A Locally Synchronous Globally Asynchronous Vertex–8 Processing Element for Image Reconstruction on a Mesh

by

Jeffrey W. Fitchett

A thesis
presented to the University of Manitoba
in partial fulfillment of the
requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba

ISBN 0-315-86005-7

Name "A Locally Synchronous Globally Asynchronous Vertex-8 Processing Element for Image Reconstruction on a Mesh."

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.

Engineering, Electronics & Electrical — SUBJECT TERM

0544 — SUBJECT CODE

U·M·I

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	0465

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

PHILOSOPHY, RELIGION AND THEOLOGY

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

SOCIAL SCIENCES

American Studies	0323
Anthropology	
Archaeology	0324
Cultural	0326
Physical	0327
Business Administration	
General	0310
Accounting	0272
Banking	0770
Management	0454
Marketing	0338
Canadian Studies	0385
Economics	
General	0501
Agricultural	0503
Commerce-Business	0505
Finance	0508
History	0509
Labor	0510
Theory	0511
Folklore	0358
Geography	0366
Gerontology	0351
History	
General	0578
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	
General	0626
Criminology and Penology	0627
Demography	0938
Ethnic and Racial Studies	0631
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	
General	0473
Agronomy	0285
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	
General	0306
Anatomy	0287
Biostatistics	0308
Botany	0309
Cell	0379
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	
General	0786
Medical	0760

EARTH SCIENCES

Biogeochemistry	0425
Geochemistry	0996
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	
General	0566
Audiology	0300
Chemotherapy	0992
Dentistry	0567
Education	0350
Hospital Management	0769
Human Development	0758
Immunology	0982
Medicine and Surgery	0564
Mental Health	0347
Nursing	0569
Nutrition	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	
General	0485
Agricultural	0749
Analytical	0486
Biochemistry	0487
Inorganic	0488
Nuclear	0738
Organic	0490
Pharmaceutical	0491
Physical	0494
Polymer	0495
Radiation	0754
Mathematics	0405
Physics	
General	0605
Acoustics	0986
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	
General	0537
Aerospace	0538
Agricultural	0539
Automotive	0540
Biomedical	0541
Chemical	0542
Civil	0543
Electronics and Electrical	0544
Heat and Thermodynamics	0348
Hydraulic	0545
Industrial	0546
Marine	0547
Materials Science	0794
Mechanical	0548
Metallurgy	0743
Mining	0551
Nuclear	0552
Packaging	0549
Petroleum	0765
Sanitary and Municipal	0554
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

Nom ____

Dissertation Abstracts International est organisé en catégories de sujets. Veuillez s.v.p. choisir le sujet qui décrit le mieux votre thèse et inscrivez le code numérique approprié dans l'espace réservé ci-dessous.

____ ☐☐☐☐ U·M·I

SUJET — CODE DE SUJET

Catégories par sujets

HUMANITÉS ET SCIENCES SOCIALES

COMMUNICATIONS ET LES ARTS

Architecture	0729
Beaux-arts	0357
Bibliothéconomie	0399
Cinéma	0900
Communication verbale	0459
Communications	0708
Danse	0378
Histoire de l'art	0377
Journalisme	0391
Musique	0413
Sciences de l'information	0723
Théâtre	0465

ÉDUCATION

Généralités	515
Administration	0514
Art	0273
Collèges communautaires	0275
Commerce	0688
Économie domestique	0278
Éducation permanente	0516
Éducation préscolaire	0518
Éducation sanitaire	0680
Enseignement agricole	0517
Enseignement bilingue et multiculturel	0282
Enseignement industriel	0521
Enseignement primaire.	0524
Enseignement professionnel	0747
Enseignement religieux	0527
Enseignement secondaire	0533
Enseignement spécial	0529
Enseignement supérieur	0745
Évaluation	0288
Finances	0277
Formation des enseignants	0530
Histoire de l'éducation	0520
Langues et littérature	0279
Lecture	0535
Mathématiques	0280
Musique	0522
Orientation et consultation	0519
Philosophie de l'éducation	0998
Physique	0523
Programmes d'études et enseignement	0727
Psychologie	0525
Sciences	0714
Sciences sociales	0534
Sociologie de l'éducation	0340
Technologie	0710

LANGUE, LITTÉRATURE ET LINGUISTIQUE

Langues	
Généralités	0679
Anciennes	0289
Linguistique	0290
Modernes	0291
Littérature	
Généralités	0401
Anciennes	0294
Comparée	0295
Mediévale	0297
Moderne	0298
Africaine	0316
Américaine	0591
Anglaise	0593
Asiatique	0305
Canadienne (Anglaise)	0352
Canadienne (Française)	0355
Germanique	0311
Latino-américaine	0312
Moyen-orientale	0315
Romane	0313
Slave et est-européenne	0314

PHILOSOPHIE, RELIGION ET THÉOLOGIE

Philosophie	0422
Religion	
Généralités	0318
Clergé	0319
Études bibliques	0321
Histoire des religions	0320
Philosophie de la religion	0322
Théologie	0469

SCIENCES SOCIALES

Anthropologie	
Archéologie	0324
Culturelle	0326
Physique	0327
Droit	0398
Économie	
Généralités	0501
Commerce-Affaires	0505
Économie agricole	0503
Économie du travail	0510
Finances	0508
Histoire	0509
Théorie	0511
Études américaines	0323
Études canadiennes	0385
Études féministes	0453
Folklore	0358
Géographie	0366
Gérontologie	0351
Gestion des affaires	
Généralités	0310
Administration	0454
Banques	0770
Comptabilité	0272
Marketing	0338
Histoire	
Histoire générale	0578
Ancienne	0579
Médiévale	0581
Moderne	0582
Histoire des noirs	0328
Africaine	0331
Canadienne	0334
États-Unis	0337
Européenne	0335
Moyen-orientale	0333
Latino-américaine	0336
Asie, Australie et Océanie	0332
Histoire des sciences	0585
Loisirs	0814
Planification urbaine et régionale	0999
Science politique	
Généralités	0615
Administration publique	0617
Droit et relations internationales	0616
Sociologie	
Généralités	0626
Aide et bien-àtre social	0630
Criminologie et établissements pénitentiaires	0627
Démographie	0938
Études de l' individu et de la famille	0628
Études des relations interethniques et des relations raciales	0631
Structure et développement social	0700
Théorie et méthodes.	0344
Travail et relations industrielles	0629
Transports	0709
Travail social	0452

SCIENCES ET INGÉNIERIE

SCIENCES BIOLOGIQUES

Agriculture	
Généralités	0473
Agronomie.	0285
Alimentation et technologie alimentaire	0359
Culture	0479
Élevage et alimentation	0475
Exploitation des péturages	0777
Pathologie animale	0476
Pathologie végétale	0480
Physiologie végétale	0817
Sylviculture et faune	0478
Technologie du bois	0746
Biologie	
Généralités	0306
Anatomie	0287
Biologie (Statistiques)	0308
Biologie moléculaire	0307
Botanique	0309
Cellule	0379
Écologie	0329
Entomologie	0353
Génétique	0369
Limnologie	0793
Microbiologie	0410
Neurologie	0317
Océanographie	0416
Physiologie	0433
Radiation	0821
Science vétérinaire	0778
Zoologie	0472
Biophysique	
Généralités	0786
Medicale	0760

SCIENCES DE LA TERRE

Biogéochimie	0425
Géochimie	0996
Géodésie	0370
Géographie physique	0368
Géologie	0372
Géophysique	0373
Hydrologie	0388
Minéralogie	0411
Océanographie physique	0415
Paléobotanique	0345
Paléoécologie	0426
Paléontologie	0418
Paléozoologie	0985
Palynologie	0427

SCIENCES DE LA SANTÉ ET DE L'ENVIRONNEMENT

Économie domestique	0386
Sciences de l'environnement	0768
Sciences de la santé	
Généralités	0566
Administration des hipitaux	0769
Alimentation et nutrition	0570
Audiologie	0300
Chimiothérapie	0992
Dentisterie	0567
Développement humain	0758
Enseignement	0350
Immunologie	0982
Loisirs	0575
Médecine du travail et thérapie	0354
Médecine et chirurgie	0564
Obstétrique et gynécologie	0380
Ophtalmologie	0381
Orthophonie	0460
Pathologie	0571
Pharmacie	0572
Pharmacologie	0419
Physiothérapie	0382
Radiologie	0574
Santé mentale	0347
Santé publique	0573
Soins infirmiers	0569
Toxicologie	0383

SCIENCES PHYSIQUES

Sciences Pures

Chimie	
Genéralités	0485
Biochimie	487
Chimie agricole	0749
Chimie analytique	0486
Chimie minérale	0488
Chimie nucléaire	0738
Chimie organique	0490
Chimie pharmaceutique	0491
Physique	0494
PolymÇres	0495
Radiation	0754
Mathématiques	0405
Physique	
Généralités	0605
Acoustique	0986
Astronomie et astrophysique	0606
Électronique et électricité	0607
Fluides et plasma	0759
Météorologie	0608
Optique	0752
Particules (Physique nucléaire)	0798
Physique atomique	0748
Physique de l'état solide	0611
Physique moléculaire	0609
Physique nucléaire	0610
Radiation	0756
Statistiques	0463

Sciences Appliqués Et Technologie

Informatique	0984
Ingénierie	
Généralités	0537
Agricole	0539
Automobile	0540
Biomédicale	0541
Chaleur et ther modynamique	0348
Conditionnement (Emballage)	0549
Génie aérospatial	0538
Génie chimique	0542
Génie civil	0543
Génie électronique et électrique	0544
Génie industriel	0546
Génie mécanique	0548
Génie nucléaire	0552
Ingénierie des systämes	0790
Mécanique navale	0547
Métallurgie	0743
Science des matériaux	0794
Technique du pétrole	0765
Technique minière	0551
Techniques sanitaires et municipales	0554
Technologie hydraulique	0545
Mécanique appliquée	0346
Géotechnologie	0428
Matières plastiques (Technologie)	0795
Recherche opérationnelle	0796
Textiles et tissus (Technologie)	0794

PSYCHOLOGIE

Généralités	0621
Personnalité	0625
Psychobiologie	0349
Psychologie clinique	0622
Psychologie du comportement	0384
Psychologie du développement	0620
Psychologie expérimentale	0623
Psychologie industrielle	0624
Psychologie physiologique	0989
Psychologie sociale	0451
Psychométrie	0632

A LOCALLY SYNCHRONOUS GLOBALLY ASYNCHRONOUS VERTEX-8 PROCESSING ELEMENT FOR IMAGE RECONSTRUCTION ON A MESH

BY

JEFFREY W. FITCHETT

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

ABSTRACT

A Locally Synchronous Globally Asynchronous Vertex–8 Processing Element for Image Reconstruction on a Mesh

Abstract. Image or field reconstruction problems from non–invasive measurements are a class of signal processing applications that demand parallel processing to achieve real–time operation. Image reconstruction from parallel beam projections, an example of which is the CAT (Computerized Axial Tomography) scan, falls within this class. This thesis considers a novel architecture suitable for WSI (Wafer Scale Integration) on which algorithms for image reconstruction from parallel beam projections or other cyclic image processing algorithms could be embedded, namely the mesh of vertex–8 processors. We will show how to map iterative reconstruction algorithms onto a fine–grained mesh of custom processors and allow the reader to infer mappings for other image transform (such as the Hough transform) and pattern recognition problems. The focus is on the hardware design of a locally synchronous globally asynchronous vertex–8 processing element (PE) that is suitable for implementation in a WSI array with one PE per pixel. The PE has several unique features, including self–timed handshaking elements for inter–processor communications and a novel asynchronous scan test structure. Test chips have been sent for fabrication.

Acknowledgements

I would like to thank my advisor, Professor R.D. McLeod, who unintimidating approach to academics and research was highly motivating and greatly appreciated. The assistance of many, if not all, graduate students was indispensable. In particular, thanks to Martin Meier, Dean McNeill, and David Blight for their skill and patience while helping me with many computer problems.

Financial support from the Natural Sciences and Engineering Council of Canada and equipment loans from the Canadian Microelectronics Corporation (CMC) are gratefully acknowledged. Access to manufacturing technology was donated by Northern Telecom Electronics Incorporated through the CMC. The Manitoba Telephone System allowed a leave of absence for me to complete my degree and provided financial support and equipment loans.

Dedication

Lastly, I would like to thank my wife Sher, for her continued support in my self indulgent endeavors. I dedicate this thesis to her and to our daughter Ariel Dawn.

Jeffrey W. Fitchett
June 18, 1993

Table of Contents

List of Figures

List of Tables

List of Charts

CHAPTER 1

Introduction

Multi–processor application specific integrated circuits (ASIC's) that make use of an entire silicon or gallium–arsenide wafer are an area of much current research because of their potential to speed up computationally intensive algorithms to real time. Real time operation is often needed in signal and image processing applications, applications where the amount of data per unit time to be processed may be too large for a conventional Von Neumann machine. One such application is image reconstruction. In particular, image reconstruction from parallel beam projections is an application that has important uses in medicine, biology, and astronomy. One example in medicine is the CAT (Computerized Axial Tomography) scan. Iterative algebraic reconstruction techniques have been shown to give superior reconstructions to the common commercial technique of Fourier filtered back projection[6], but commercial implementation of improved algorithms has been limited because of computational cost. The processing power needed and the large number of nearest–neighbor communications in the iterative algorithms make them likely candidates for parallel computing multi–processor ASIC arrays; this thesis studies an architecture suitable for image reconstruction from parallel beam projections, namely the mesh of vertex–8 processors. The focus is on the hardware implementation of a globally asynchronous locally synchronous processing element (PE) for such an array, which has several unique features that make it suitable for WSI (Wafer Scale Integration) including the locally synchronous globally asynchronous architecture, self–timed handshake components for inter–processor communications, and asynchronous scan test. These features would be useful for a fault tolerant WSI array. A processing element for the proposed array was designed in VLSI (Very Large Scale Integration) and sent for fabrication.

In the reconstruction from parallel beams problem, parallel rays are projected thought an unknown image plane and their intensities detected on the other side, as in Figure 1. The difference between the transmitted and received intensities is the ray integral, the sum

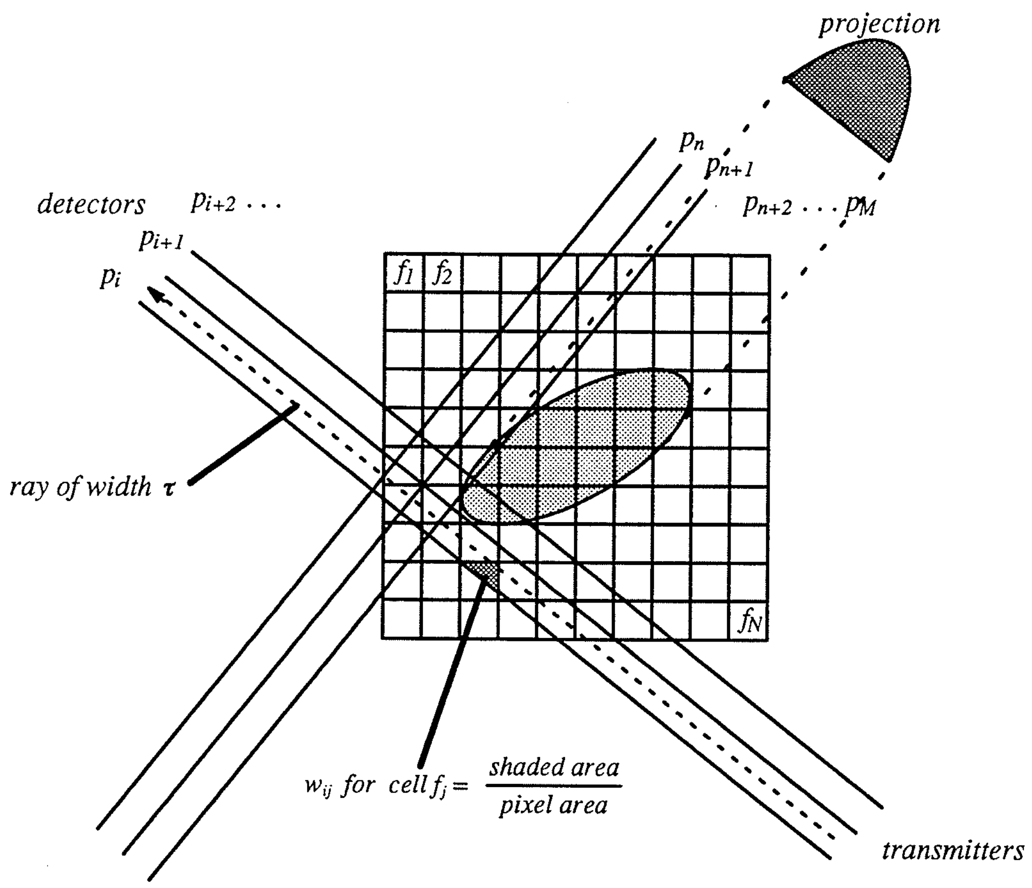


FIGURE 1 Parallel rays projected through an unknown image.[30]

of the image densities that it has passed through. By rotating the image plane or the transmitter and detector arrays, enough projection data can be obtained to reconstruct the image. Various techniques have been proposed to accomplish the reconstruction, the most popular of which is a Fourier method known as Filtered Back Projection (FBP). Other techniques such as direct matrix inversion, constraint optimization, and iterative algebraic methods can give better results but are more computationally expensive. The iterative techniques we will study are called ART (Algebraic Reconstruction Techniques).[12][13] We have performed simulated reconstructions for small images on a personal computer that have taken hours to converge using a basic ART method. A typical acquisition time for a CAT Scan machine is 1 milli–second per 1000 element projection vector, which means we must reconstruct a 1000x1000 pixel image in 1 second to do it in real time. Robotic vision applications will required even faster speeds. The reconstruction problem clearly is a candidate for application specific and/or parallel computing, such as in our proposed array.

Our array is a square mesh with one processor per pixel. The array is able to handle the bulk of the calculations in parallel, using pipelining to acheive a high degree of concurrency. Each processing element (PE) has vertex–8 connections and programmable routing tables. It is able to communicate with each of its 8 nearest neighbors and is controlled by a global command, in an SIMD (single instruction multiple data) arrangement. The large number of fine–grained processors required make it unlikely that the entire array could be manufactured on a single chip in VLSI. There are many VLSI micro–chips on the original silicon or gallium–arsenide wafer, of which a percentage are faulty; the process of trying to make use of the entire wafer is called WSI or Wafer Scale Integration. For our array, the yield of a WSI process with current technology would be unacceptable, implying that fault tolerance would have to be incorporated. Implementing a fault tolerant scheme is a synchronous WSI system would be very difficult, given that clock distribution alone in large synchronous systems is a formidable problem that has not been overcome. A globally asynchronous locally synchronous array rather than a clocked systolic one would therefore have greater potential for practical use in a WSI implementation, and that is the design philosophy we follow.

The sequential form of the iterative solutions that we wish to embed on the array will be described in the remainder of Chapter 1. In Chapter 2, *Array Architecture and Algorithm*

Simulations, we describe how the algorithms can be parallelized on our proposed vertex–8 mesh of processors. We discuss the time complexity implications for the case of one PE per pixel. Parallel simulations were done on a transputer array to verify the validity of the proposed architecture. The transputer simulations were made to closely model the envisioned hardware to provide a functional test of the PE before implementing it in VLSI. Chapter 3, *Processing Element Hardware Design,* contains details of the hardware design of the PE including a description of self–timed building blocks and major integrated components. Conclusions and recommendations for future work will be discussed in Chapter 4.

Although the array architecture we will propose can have very general uses depending on the design of the ALU (Arithmetic Logic Unit) within the PE, we have chosen to concentrate on a particular set of algorithms for reasons of succinctness and clarity. These are the iterative algorithms for image reconstruction from parallel beam projections. The reader can infer many other applications that could be embedded into our proposed array that retain the communications infrastructure used by our PE and would require only simple, if any, modifications of the PE's ALU. Our implementation differs from those in the literature[20]. Our array is fine grained and globally asynchronous with locally synchronous computing. Other implementations have used complex off the shelf processors or transputers. Our design has programmable routes, making it useful for non–algorithmic path mappings and thus suitable for many other applications such as pattern recognition. The use of programmable routing tables is a reasonable approach for cyclic algorithms. Other implementations calculate the routes dynamically, which places limits on flexibility of the paths through the array and the speed with which the array can operate.

Image and field reconstruction problems on a digital computer are currently becoming as common as they are computationally expensive. In this thesis we will examine a parallel algorithm and WSI architecture for a particular reconstruction problem, reconstruction of 2–dimensional images from parallel beam projections. The architecture could find application in other reconstruction problems as well.

1.1 THEORY

After a model for the image space is described, an iterative technique based on theory developed by Kaczmarz[30] will be developed and then approximated in a form more suitable for implementation in a digital computer. The approximated form, devised by Gordon et al.,[12][13] is called ART for Algebraic Reconstruction Technique. A modification to the algorithm proposed by Gilbert[11] called SIRT (Simultaneous Iterative Reconstruction Technique) that will improve noise performance will be explained.

1.1.1 The Problem

Image space is represented as in Figure 1. Here *f(x,y)*, the unknown image in region ***R***, is split into a grid of N square elements denoted f_j. Parallel rays of width τ pass through the image resulting in M measured rays sums p_i. Both f_j and p_i have one–index representations. The weighting factor w_{ij} represents the intersection of the j^{th} cell with the i^{th} ray, or the contribution of the j^{th} cell to the i^{th} ray sum. This geometry gives M equations in N unknowns:

$$\sum_{j=1}^{N} w_{ij}\ f_j = p_i,\ i = 1, 2, 3, . \quad . \quad . M$$

or

$$w_{11}f_1 + w_{12}f_2 + . \quad . \quad . \; w_{1N}f_N = p_1$$

$$w_{21}f_1 + w_{22}f_2 + . \quad . \quad . \; w_{2N}f_N = p_2$$

.

.

.

$$w_{M1}f_1 + w_{M2}f_2 + . \quad . \quad . \; w_{MN}f_N = p_M \qquad (1)$$

Gordon[13] points out some interesting features of equations (1), some of which must be noted if we are to optimize a solution process, i.e.:

1. The matrix $\{w_{ij}\}$ is sparse, which will ease calculations.
2. The matrix $\{w_{ij}\}$ can be huge.
3. Elements w_{ij}, f_j, p_j are always positive.
4. The data is often inconsistent. It can contain noise or errors.

Before we can begin developing the technique, we must determine how to best choose M and N in the computer's representation of the problem. M and N are dependent on the size of the projection elements and the desired resolution of the reconstructed image. The fineness of the discrete projection elements is limited by the resolution of the measured projection data, which depends on the physics of the radiation and measurement devices used. If we wish to get the maximum possible resolution in the reconstructed image with the minimum of calculations, we want the projection elements to be as large as possible without degrading the reconstructed image. Typically the projection element spacing is chosen to be half of the presumed resolution to fulfill the Nyquist criteria. A coarser choice for projection element width is limited only by the desired degree of resolution in the reconstructed image. One may want the division of $\boldsymbol{R}$ to be as fine as possible, but finer elements in the division of $\boldsymbol{R}$ (or in the projection elements) will increase calculations. One the other hand, the coarseness of the division of $\boldsymbol{R}$ is limited by the size of the projection elements. This is especially true when certain approximations to equations (1) are used; more will be said about the coarseness limitation on $\boldsymbol{R}$ after the approximations are developed in Section 1.1.3 on Gordon's appoximation.

1.1.2 The Kaczmarz method

The Kaczmarz method is an iterative procedure that can be used to solve simulaneous equations. References [30] and [12] give excellent descriptions of the development and implementation of the algorithm. The approach taken here is similar to that in [30], with a few refinements.

To begin, one must realize that the N unknown grid elements in equations (1) give an image N degrees of freedom. An image can therefore be represented as a single point in an N–dimensional space. Each individual equation in (1) would represent a hyperplane in such a space. If a unique solution exists, the intersection of these hyperplanes is a single point representing the solution. The Kaczmarz method uses successive projections of a proposed solution onto adjacent hyperplanes to iterate closer and closer to the solution. This is best illustrated by a simple 2 variable example with unknowns f_1 and f_2, written:

$$w_{11}f_1 + w_{12}f_2 = p_1,\ \ w_{21}f_1 + w_{22}f_2 = p_2 \tag{2}$$

Figure 2 gives a conceptual view of how the iteration process evolves. It can be shown via vector analysis (see [30]) that an iterative equation for a projection from the $(j-1)^{th}$ to the j^{th} hypeplane can be written

$$\vec{f}^{(j)} = \vec{f}^{\,(j-1)} - \frac{\vec{w}_j \cdot \vec{f}^{(j-1)} - p_j}{\vec{w}_j \cdot \vec{w}_j}\vec{w}_j \tag{3}$$

where

$$\vec{f} = \{f_1 f_2, \ldots f_N\},$$

$$\vec{w}_j = \{w_{j1}, w_{j2}, \ldots w_{jN}\},$$

and j ranges from 1 to M. We will forgo the derivation. Equation (3) will iteratively move the initial estimate $\vec{f}^{(0)}$ towards the solution at the intersection of the hyperplanes. After a complete pass thru all M projections the process can be repeated starting from the first pro-

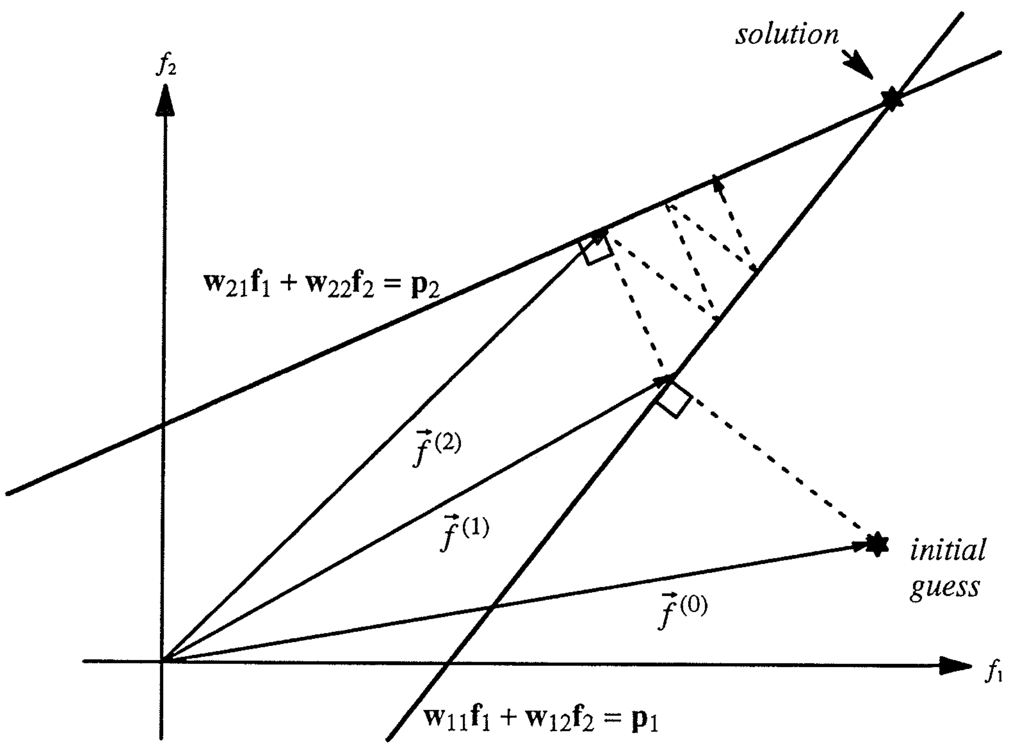


FIGURE 2 The Kaczmarz method of solving algebraic equations for 2 unknowns. From the initial guess, perpendicular projections onto adjacent hyperplanes (lines for the 2–d case) for each equation iterate closer and closer to the solution.[30]

jection using the new estimate of the solution $\vec{f}^{(M)}$, then again starting with $\vec{f}^{(2M)}$, $\vec{f}^{(3M)}$, . . . $\vec{f}^{(kM)}$ where k can be at convergence or as large as required to get the desired results.

But does the process converge? It is obvious for the 2–d example but not for systems with many more equations. Tanabe[34] has proved for the general case of equation (3) that if a unique solution to (1) exists,

$$\lim_{k \to \infty} \vec{f}^{(kM)} = \vec{f}_s \tag{4}$$

This is true if equations (1) are overdetermined and consistent, i.e. $M>N$ with no noise or distortions in the data. If they underdetermined, then (3) will converge to a solution (one of the many possible solutions) that is closest to the initial estimate.

$$\lim_{n \to \infty} \vec{f}^{(kM)} = \vec{f}_s' \quad \textit{such that} \; | \vec{f}^{(0)} - \vec{f}_s' | \quad \textit{is minimized} \qquad (5)$$

If the data in equations (1) is noisy or distorted, then (3) will oscillate in the neighborhood of the actual solution. This attribute makes the Kaczmarz method somewhat, but not completely, noise tolerant.

1.1.3 Gordon's Approximation

Equation (3), being iterative, is reasonably easy to implement in a computer program. However, the storage of and floating point calculations on what could be an enormous number of weights w_{ij} makes it cumbersome for digital computers. For example, for a 16 projection 64x64, the size of the w_{ij} matrix would be:

$$(64\text{x}64) \text{ x } (16\text{x}128) = 4096 \text{ x } 2048 = 4.2\text{x}10^6 \text{ elements} \qquad (6)$$

Even a small image with a limited number of projections is too big for matrix inversion methods. About 1/100 of these elements will be non–zero, which still leaves a large amount of numbers considering the many floating point calculations that must be performed.

Many methods to overcome these difficulties have been proposed; one efficient method is Gordon's approximation, which drastically reduces the number of non–zero weights and eliminates much floating point calculation and storage. The approach is simply that the weight is assigned a one if the centroid of the grid element is within the ray and a zero if it is not. Using this approximation, we can develop a form of (3) that can be run more easily and faster on a digital computer. Applying the approximation and again forgoing the details of the derivation, we arrive at the simple and intuitive equation below for the update of the m^{th} cell from the j^{th} ray sum.

$$\Delta f_m^{(j)} = \frac{p_j - q_j}{N_j} \qquad (7)$$

where

p_j is the measured projection for a ray,

$q_j = \sum_{k=1}^{N} f_k^{(j-1)} w_{jk}$ is the calculated projection, and

$N_j = \sum_{k=1}^{N} w_{jk}^2$ is the number of pixels in the ray.

Recall that the weight w_{jk} or w_{jm} will be 1 if the cell centroid is in the ray and zero elsewhere if we apply Gordon's approximation. One iteration for the m^th cell would be

$$f_m^{(j)} = f_m^{(j-1)} + \Delta f_m^{(j)} \tag{8}$$

Equations (7) and (8) were the equations implemented in our simulations, with some of the enhancements forthcoming. An intuitive interpretation of them is that the average error for each pixel in a ray is calculated by taking the difference between the measured and calculated pseudo projections and dividing by the number of pixels in the ray. This 'average error' is then backprojected to each pixel in the ray. We will dub this implementation of the Kaczmarz method combined with Gordon's approximation as ART, for Algebraic Reconstruction Technique.[1] This solution has been shown to be a good approximation[30] to the constrained least squares approach.

For our application, an improvement to ART would be to use a priori knowledge about the image to speed convergence. For example, we know that the image intensities are greater than zero so we can apply that to the $f_m^{(j)}$'s before proceeding to the next iteration.

1. Now that equations (7) and (8) and Gordon's approximation have been developed, it is more appropriate to discuss the limits in coarseness on the representation of space. If a calculated ray sum is to provide any representation of the image, number of cell centroid N_j must be non–zero. This limits the coarseness of the grid. Further restrictions on coarseness are required to provide an accurate representation of the density along the ray. The centroids should be evenly distributed along the ray's length. Gordon has estimated that this requirement will be sufficiently met if $N_j > n^{1/d}$ for rays that are about the diameter of $\boldsymbol{R}$, for a grid on n elements in d–dimensional space. Here $n^{1/d}$ is the average ray length in units the same size as the cell width. A grid element size of no greater than the width of the projection elements will guarantee this condition. For this reason, we chose the pixel width equal to the ray width.

We can do the same thing with the upper limit. This has been dubbed *fully constrained* ART by Gordon.

$$f_m^{(j)} = min[MAXPIX, max[0, f_m^{(j-1)} + \Delta f_m^{(j)}] \qquad (9)$$

For the purposes of this paper we will call the algorithm described thus far ART, although the term can have generic connotations in the literature. This form of ART is the most basic in a family of ART algorithms.

1.1.4 Initialization Value

Since the Kaczmarz/ART method will converge to the solution closest to $f^{(0)}$ for underdetermined equations, the starting value should be a best guess of the solution. The better a starting value chosen, the faster the algorithm will converge and the smaller unwanted artifacts will be. The mean density of f is one possibility, which is approximately

$$f_{kmean} = \frac{\sum p_j \ \ such\ that\ p_j\ belongs\ to\ a\ single\ projection}{area\ of\ \mathbf{R}\ normalized\ to\ p_j\ widths} \qquad (10)$$

or even better

$$f_{mean} = \frac{1}{K}\sum_{k=1}^{K} f_{kmean} \qquad (11)$$

where K is the number of projections. A *projection* is the set of ray sums taken over one discrete projection angle. These equations are true since the parallel rays fully partition the scanned space.

Usually the initial value is chosen as zero since the result after iterating through the first projection is roughly the same as f_{mean}. Another alternative is to use a SBP (Simple Back Projection) result as an initialization value. The problem with this is that distortions introduced by the initial estimate may be retained as ghosts. For these reasons, zero was chosen as the initialization value in our implementation.

1.1.5 Convergence criteria

Gordon, Bender, and Herman[13] suggested 3 possible measures for convergence.

1. The difference between the measured and calculated projection values: (goes to zero)
$$D^{(t)} = \sqrt{\frac{1}{M}\sum_{j=1}^{M}\frac{(p_j - q_j^{(t)})^2}{N_j}}$$
(12)
where t is the iteration number.

2. The non–uniformity or variance: (goes to a minimum)
$$V^{(t)} = \sum_{i=1}^{N}(f_i^{(t)} - f_{mean})^2 \tag{13}$$

3. The entropy: (goes to a minimum)
$$S^{(t)} = \frac{-1}{\ln N}\sum_{i=1}^{N}\frac{f_i^{(t)}}{f_{mean}}\ln\frac{f_i^{(t)}}{f_{mean}} \tag{14}$$

One could suggest that a form of method 1 would be the most efficient since most of these calculations must be done anyway. If noise is present there will be a problem with detecting convergence since the algorithm will oscillate near the solution. Thresholding must be used appropriately in the programming to watch for divergence and terminate execution at a satisfactory time. Another important measurement of convergence when an original test image f_i is used for simulations is the Euclidean distance:

$$\delta^{(t)} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(f_i^{(t)} - f_i)^2} \tag{15}$$

The Euclidean distance from the original image is a measure of the degree of accuracy of the reconstruction.

In our programs, a minimum{maximum Δf} was chosen for a convergence criteria. This is a factor in calculating most of the above values, and is accurate enough for our purposes since only clean projection data was used. As an option, we can stop the simulation after a certain number of iterations and calculate the Euclidean distance to check the degree of convergence.

1.1.6 Variations

The ART technique developed so far has been shown to perform poorly when compared to Fourier techniques when the projection data is noisy and/or inconsistent. This is mainly because it only looks at one projection at a time before performing an iteration. A plethora of variations have been proposed to reduce the effects of bad projection data and to eliminate artifacts inherent in the algorithm.

To reduce artifacts from geometric distortions caused by the discrepancy between the actual ray integral and the centriod technique, Gilbert[11] replaced equation (7) with

$$\Delta f_m^{(j)} = \frac{p_j}{L_j} - \frac{q_j}{N_j} \tag{16}$$

where L_j is the length of the j^{th} ray through the reconstruction region, normalized to the cell width. Gordon et al. have applied similar modifications to ART.

Another modification proposed by Gilbert is SIRT, or Simultaneous Iterative Reconstruction Technique, which makes the algorithm more noise tolerant. In this technique, an iteration is performed using the data from all projections simultaneously rather than one by one. Only at the end of each complete pass through all projection elements for the K projection angles are cell values changed, by the average value of the K Δf's belonging to a pixel. The SIRT variation will average out bad projections:

$$\Delta f_{m_{av}}^{(j=nK)} = \frac{\sum_{j=(n-1)K}^{nK} \Delta f_m^{(j)}}{K} \quad \text{where } n = 1, 2, 3.... \tag{17}$$

$$f_m^{(nK)} = f_m^{((n-1)K)} + \Delta f_{m_{av}}^{(nK)} \tag{18}$$

With ART some pre–filtering of projection data must be done or modifications to the algorithm must be incorporated to enhance noise performance. The drawback with SIRT is that it would require more memory; with the improvements that have been made to ART, the performance of ART and SIRT would be comparable. In algorithm tests of Chapter 2, the PC simulations implement ART and the parallel transputer simulations implement a form of SIRT.

In [12] Gordon reviews numerous other improvements to the basic ART technique which include:

1. ART2, which uses an intermediate unconstrained estimator to calculate the next iteration in constrained ART. This speeds convergence.
2. ART3, which uses a preset error threshold that makes the iterative procedure more tolerant to noise and inconsistent data.
3. ART with a damping factor. This slows convergence but improves noise performance.

Some of these improvements must be applied if algebraic techniques are to compete with Fourier techniques, especially with real rather than simulated data. One must carefully select those to be implemented since they all have some computational cost. Although our simulations were done using the basic constrained ART algorithm and the SIRT modification, the architecture that we will propose can accommodate many of the aforementioned improvements since they can be performed by the host or a complementary chip set. The ASIC array performs the ray summing and update, or projection and backprojection, portions of the algorithms. These are the most computationally intense parts of the algorithms.

CHAPTER 2

Array Architecture and Algorithm Simulations

Before committing to a hardware architecture, simulations were done with the sequential reconstruction algorithms and with a parallel implementation of an ART. The serial simulations were necessary to ensure that the ART or SIRT algorithms could out–perform the other techniques in reconstruction quality, to show the need for parallelization of the iterative algorithms, and to gain an understanding of the image reconstruction algorithms. Our results confirm the first two items. The parallel simulations were done on a transputer array to test a novel parallel version of the ART algorithm, including improvements and modifications such as SIRT, and to verify that the algorithms could be mapped onto our proposed array architecture. The proposed architecture and mapping will be discussed after the results of the sequential simulations are presented, followed by a description of the parallel simulation results at the end of the chapter.

2.1 SEQUENTIAL SIMULATIONS

For the sequential simulations, three reconstruction algorithms were programmed and tested using pseudo–projections generated artificially from a test image. They include:

1. *Simple Back Projection (SBP).* The ray sums for each projection angles are backprojected, summed, and scaled with no further correction. This is a very crude reconstruction technique.

2. *Filtered Back Projection (FBP).* A Fourier technique. A filters is applied to the 1–D Fourier transforms of each of the projections to create a 'slice' of the 2–D transform of the original image. The 2–D transform is then used to reconstruct the image; see [30] for details. FBP or Fourier reconstruction is a common technique because of its computational efficiency and noise tolerance. Our implementation of of FBP could be improved,but literature[5] supports the overall reslts.
3. *Algebraic Reconstruction Technique (ART).* The constrained ART algorithm was implemented as described in Chapter 1. The ART simulations used equations (7), (8), and (9).

The sequential programs were written in *C* and compiled and run on a 286 personal computer without a co–processor. The programs for SBP and FBP were based on previous work[21]. The program listing for ART is in the *Technical Reference Addendum;* input to the ART program for a simulation would be as follows:

1. A measured/simulated projection data file. For our simulations this file contained artificially generated tomographic projection data from a black and white test image with 256 gray levels. The format of the file is:

```
SIZE OF PROJECTION
NUMBER OF PROJECTIONS
ANGLE
PROJECTION ELEMENT/RAY SUM 1
PROJECTION ELEMENT 2
ETC.
.
.
NEXT ANGLE
PROJECTION ELEMENT 1
PROJECTION ELEMENT 2
.
.
K^th ANGLE
PROJECTION ELEMENT 1
PROJECTION ELEMENT 2
.
.
M^th PROJECTION ELEMENT
```

2. Number of desired iterations.
3. The dimensions of the reconstructed image.

The output of the program is the reconstructed image. Support programs not shown in the *Technical Reference Addendum* include a program to generate projection data and one to calculate the Euclidean distance between the original and the reconstructed image per equation (15).

The sequential ART algorithm is very slow, since for each sub–iteration every pixel must be visited once to calculate the pseudo projection q_j and pixel count N_j and once to perform the update. Here we define a 'sub–iteration' as one pass of the update equation (8) through all grid elements, equivalent to one pass through rays sum belonging to a particular projection angle. A complete iteration would be one pass through all rays sum data for the K projection angles. Ray membership is determined using the formula:

$$ray\ number = \lfloor \varrho \rfloor = \lfloor x \cos\theta y \sin\theta \rfloor \qquad (19)$$

The transformation from x and y to ϱ and θ required in each *project* phase is equivalent to the Hough transform[8], an image transform commonly used in industry for edge and line detection. The *cos* and *sin* functions are calculated by slow iterative means [35] on most computers. Even for a small test image of 64x64 pixels the ART algorithms took hours to converge on our PC programs. The degree of convergence was measured by calculating the Euclidean distance between the reconstructed image and the original test image, after a chosen number of iterations.

The main storage requirements of the sequential program were N integers for the reconstructed image and arrays of size $\sqrt{2N}$ for Δf, q_j and N_j. The image was initialized to a guess of *255/2=127* rather that zero. In retrospect, this choice sped convergence (as opposed to a zero initialization value, but contributed to unwanted artifacts.

2.1.1 Sequential Results

The sequential ART reconstruction program was run many times for a varied number of iterations from sets of projections generated from several test images. The results were analyzed pictorially by looking at the images and numerically by plotting combinations of the number of iterations, the mean squared error and the convergence time against each other. We will present 4 examples of simulated reconstructions from 64x64 pixel images:

1. *Saturn,* 4 projections. (Figure 3)
2. *U of M Administration Building,* 16 projections. (Figures 4 & 5)
3. *U of M Administration Building,* 32 projections. (Figure 6)
4. *Saturn,* 64 projections. (Figure 7)

Pictorial Analysis

Some heuristic judgements are necessary when analyzing visual data; we will later confirm our observations with numerical results. The original test images are shown for comparison in the last plate of each of Figures 3, 4, 5, 6, and 7. By examining combinations of pictures in the figures, we can make observations about the performance of ART, SBP, and FBP:

ART improves with increasing iterations or time. Sequentially examining image sets by iteration number shows the reconstructed image appearing more and more like the original image with each iteration. This can be seen particularly in Figures 4 and 5, where each sub–iteration for the first iteration is shown as well as subsequent full iterations.

ART improves with increasing projections. Comparing the converged reconstructions in figures 3 and 7 and figures 5 and 6 demonstrates improvement in the reconstruction quality with increasing projections. This is true when the data is not noisy.

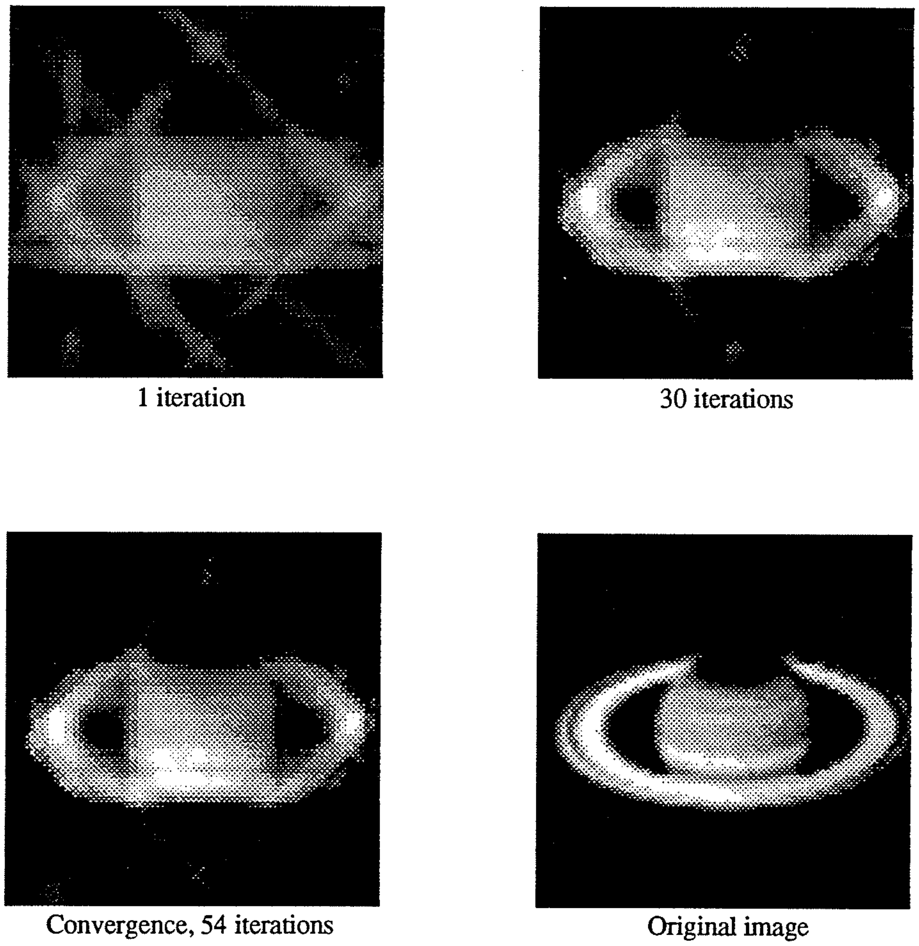


FIGURE 3 *Saturn* reconstructed from 4 projections using the ART algorithm.

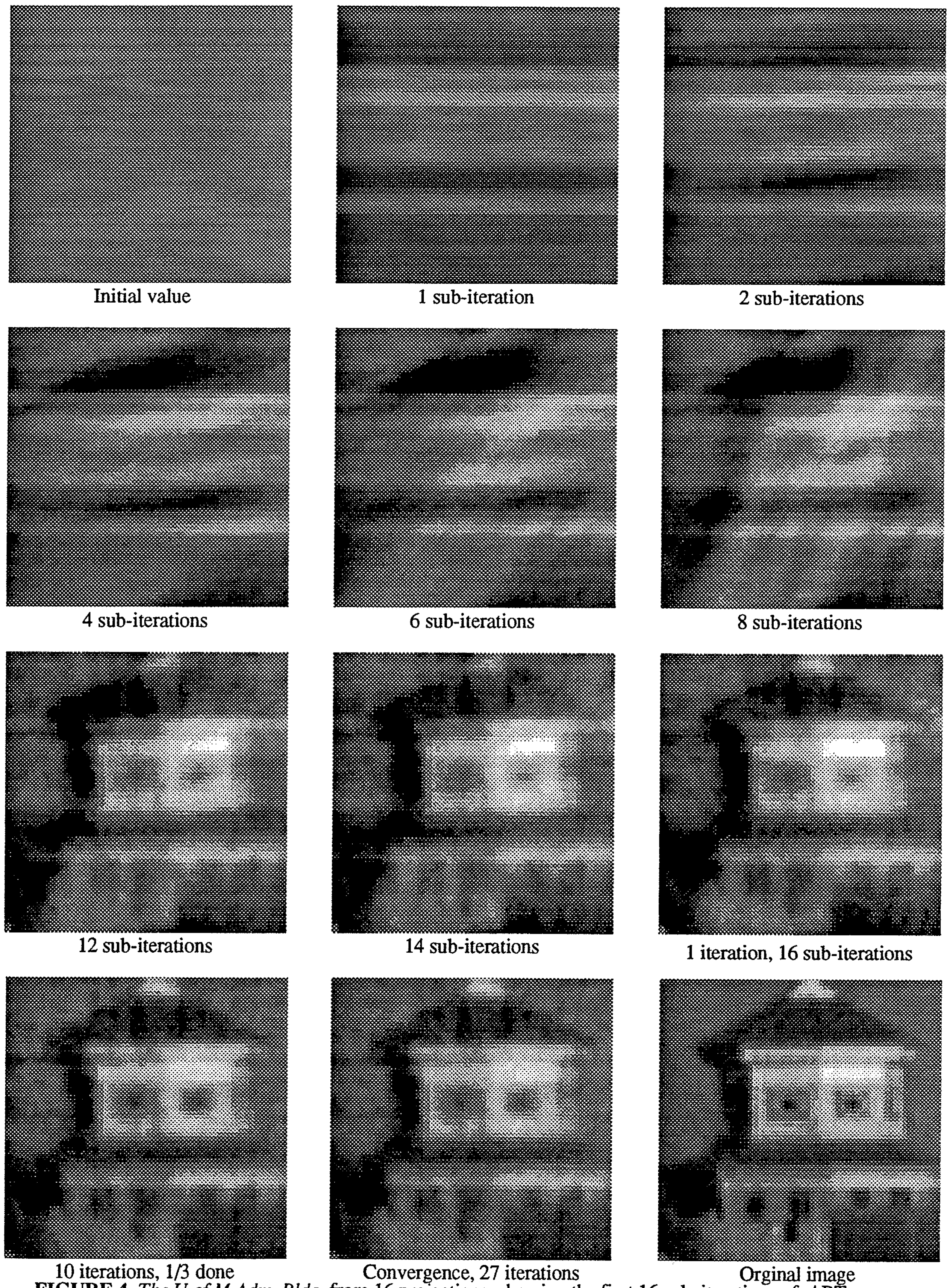
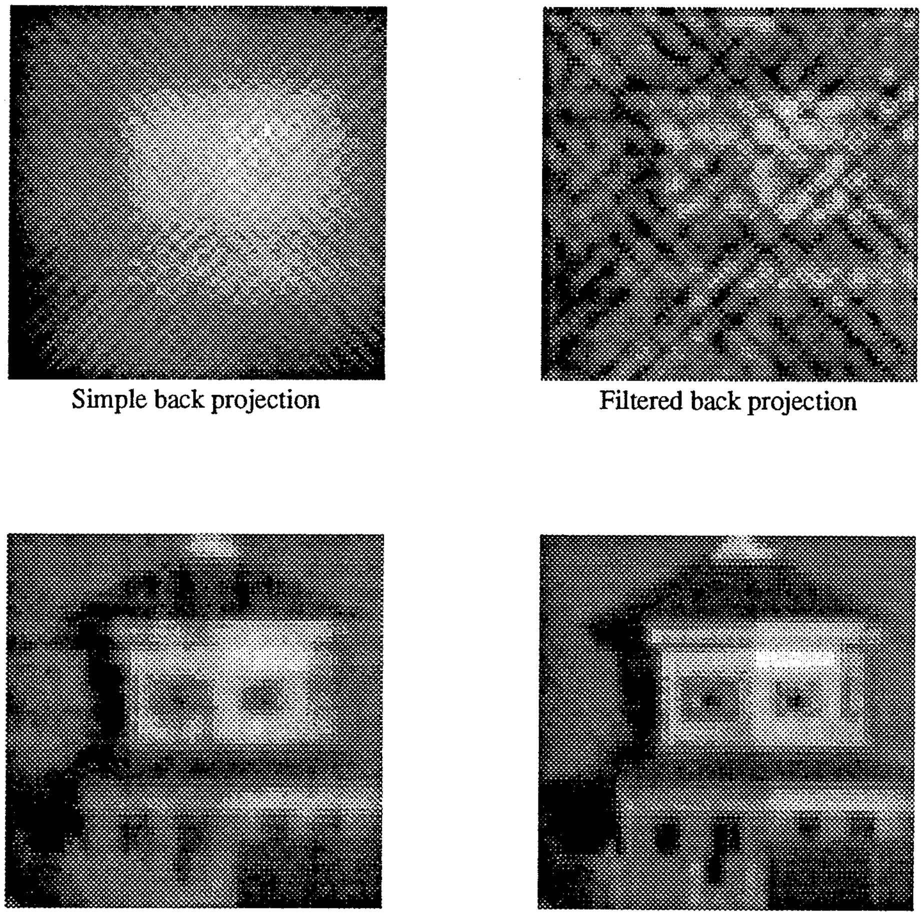


FIGURE 4 *The U of M Adm. Bldg.* from 16 projections, showing the first 16 sub–iterations of ART.



Simple back projection

Filtered back projection

Convergence, 27 iterations

Original image

FIGURE 5 *The U of M Administration Building* reconstructed from 16 projections, showing the ART algorithm at convergence and SBP and FBP reconstructions.

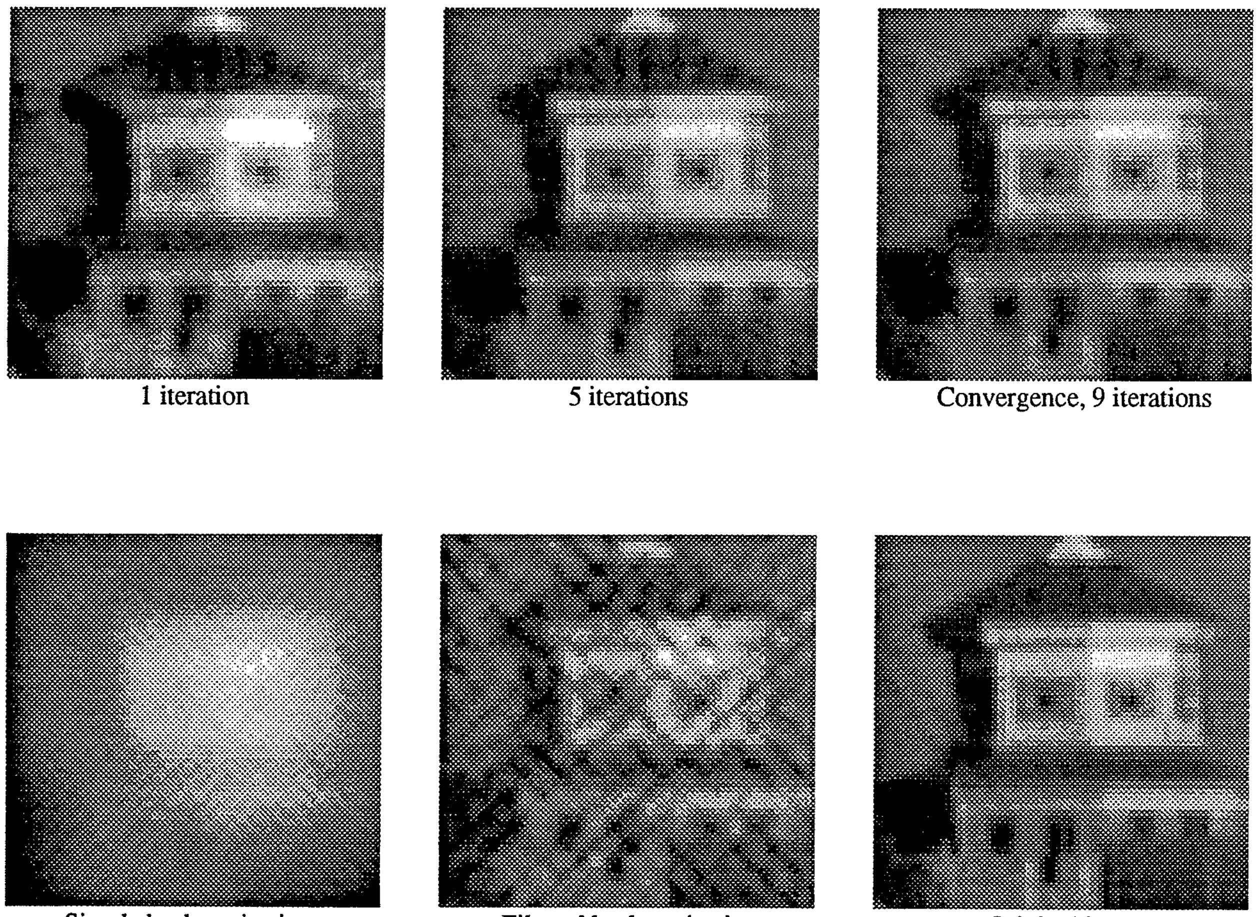


FIGURE 6 *The U of M Administration Building* reconstructed from 32 projections, showing the ART algorithm converging and SBP and FBP reconstructions.

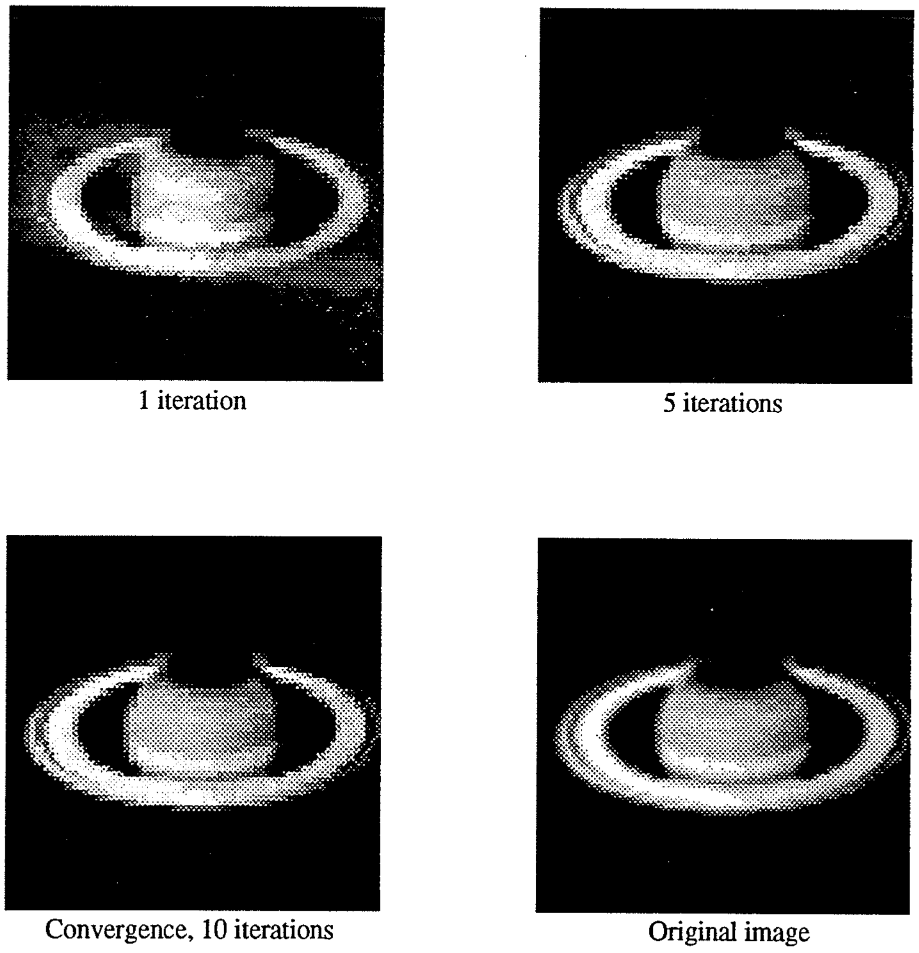


FIGURE 7 *Saturn* reconstructed from 64 projections using the ART algorithm.

Numerical Analysis

Numerical analysis verifies our visual observations. Charts 1, 2, 3, 4, 5 plot the mean squared error (MSE), number of iterations, and execution time against each other in different combinations and scenarios for the reconstructions of Figures 3 to 7.

Chart 1 is a plot of the MSE versus time. It shows that ART can take a large amount of computing time and that ART can out–perform the FBP algorithm in reconstruction accuracy. Note that the MSE for ART decreased monotonically with time; this would not be true if noisy data was used.

Chart 2 is MSE versus iterations. This plot does not add much information to the observations on Chart 1, except to compare it with Chart 1 and note that more projection data required more time per iteration so that the relationship between time and iterations is non–linear. The relationship between time and sub–iterations would have been linear.

Chart 3, the MSE at convergence versus the number of projections, shows that for ART the MSE at convergence monotonically decreased with increasing projections and that ART at convergence had a lower MSE that the FBP and SBP examples.

Chart 4 shows the time to convergence versus the number of projections. The time to convergence generally increases with an increased number of projections, but not necessarily monotonically. The time to convergence for ART was greater than the FBP or SBP run–times in all cases.

Chart 5, the time needed for ART to achieve a lower MSE than either the FBP or SBP algorithms, shows that the time required for ART to outperform FBP or SBP is comparable (less than a factor of 2) to the FBP and SBP executions times. The time for complete convergence is much longer.

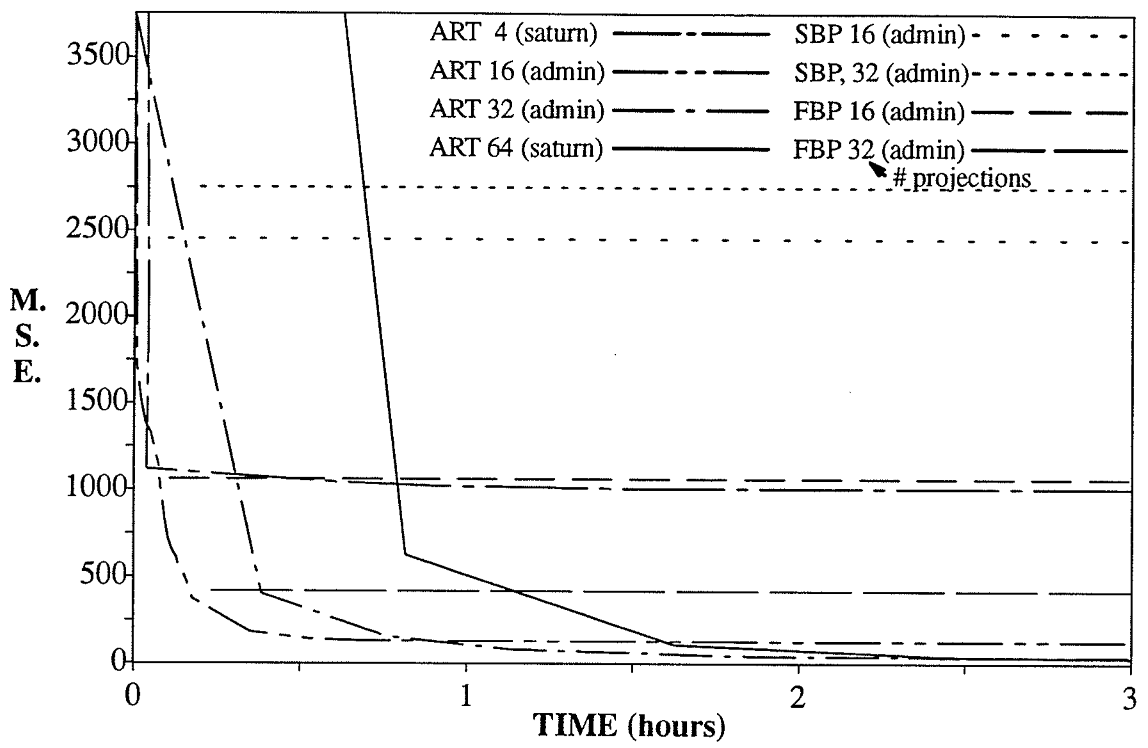


CHART 1 The mean squared error versus time for the ART, FBP, and SBP algorithms.

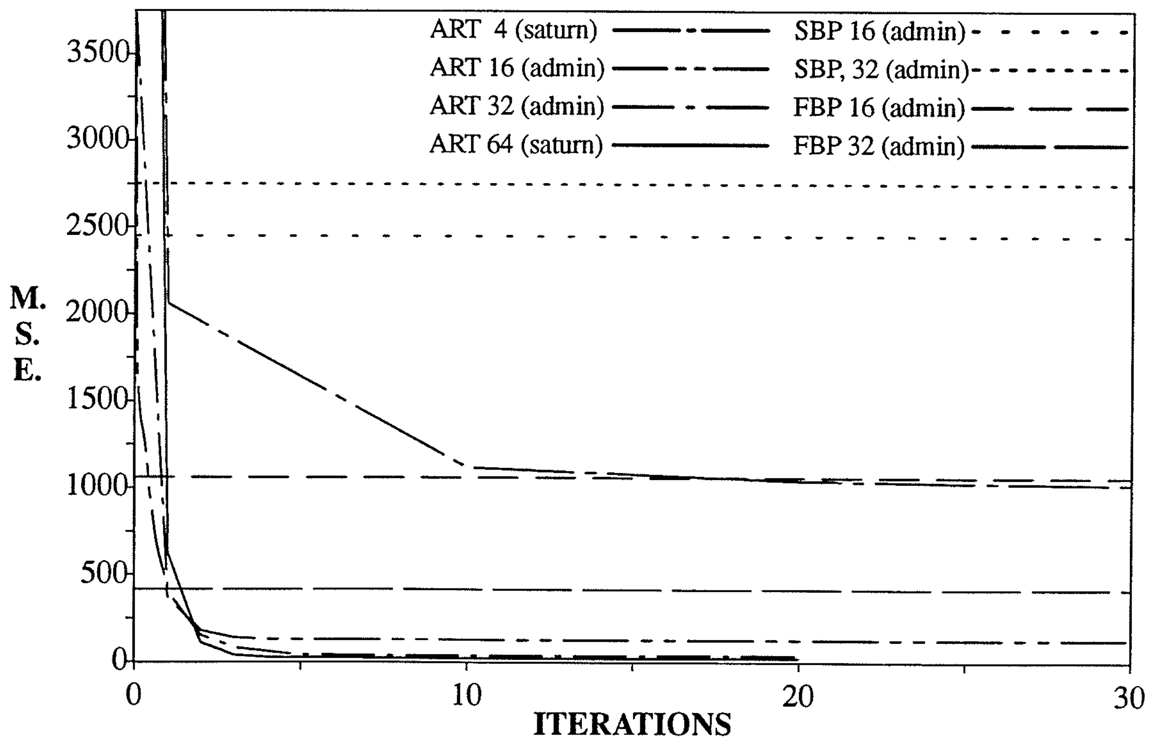


CHART 2 The mean squared error versus the number of iterations for ART.

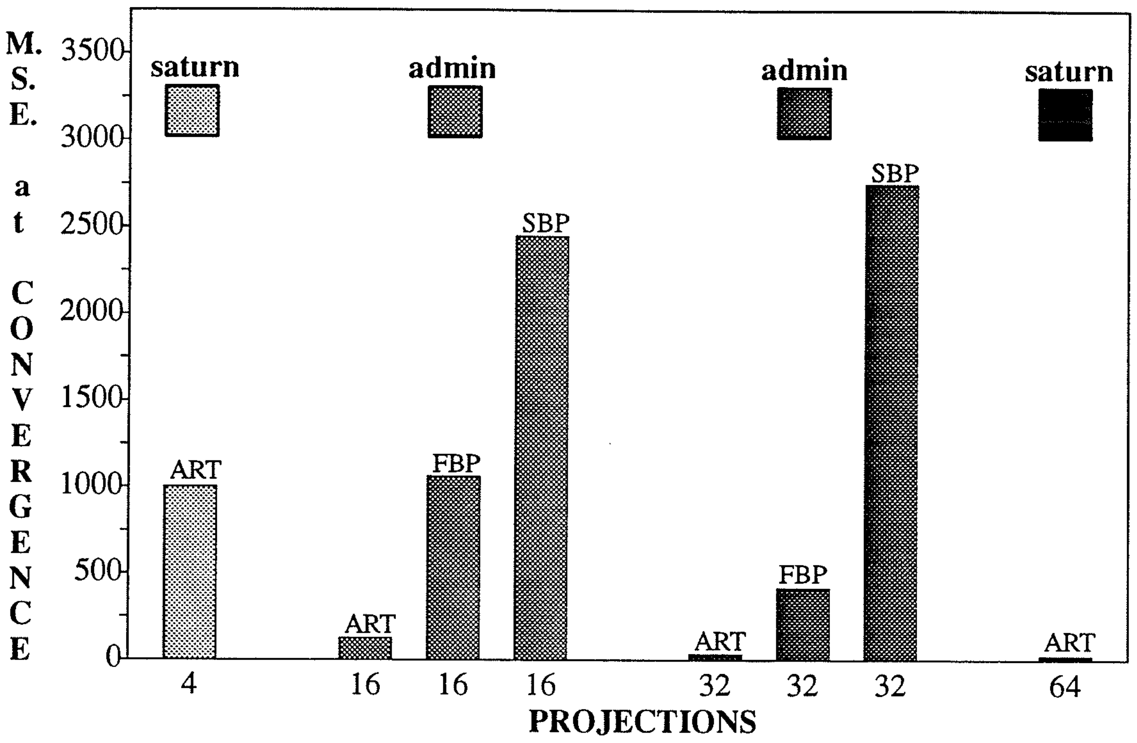


CHART 3 The mean squared error at convergence versus the number of projections used to reconstruct the image for the ART, FBP, and SBP algorithms.

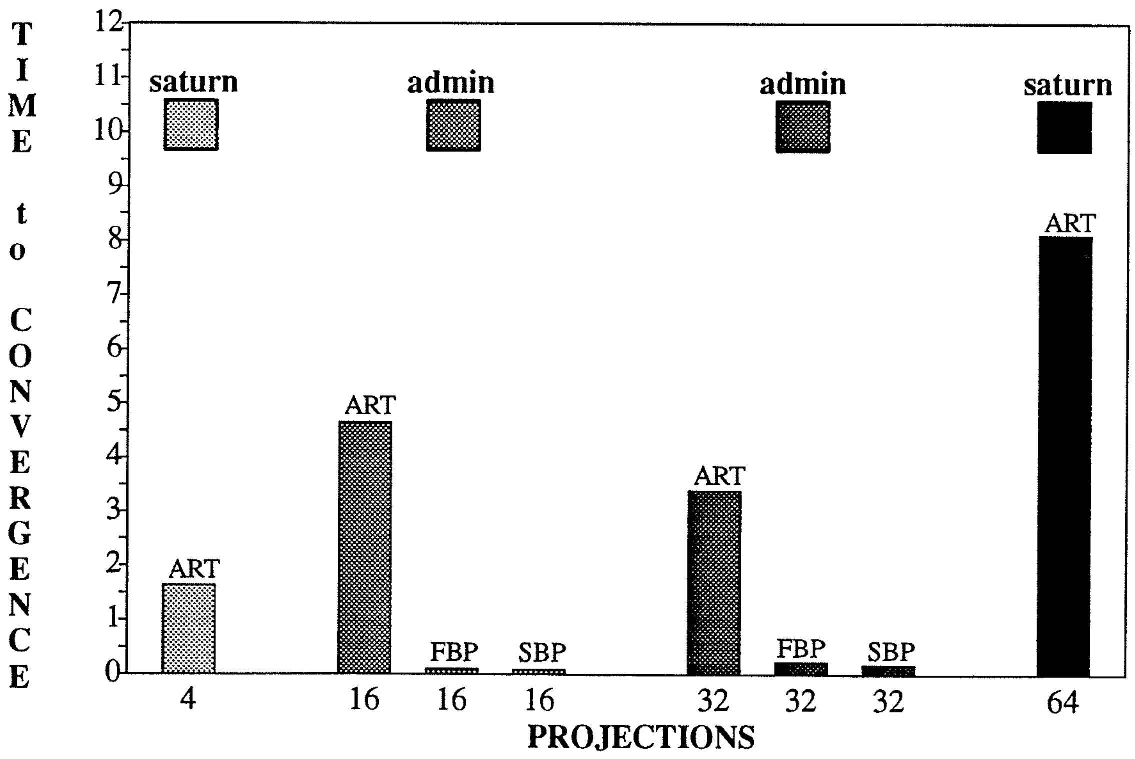


CHART 4 The time to convergence versus the number of projections.

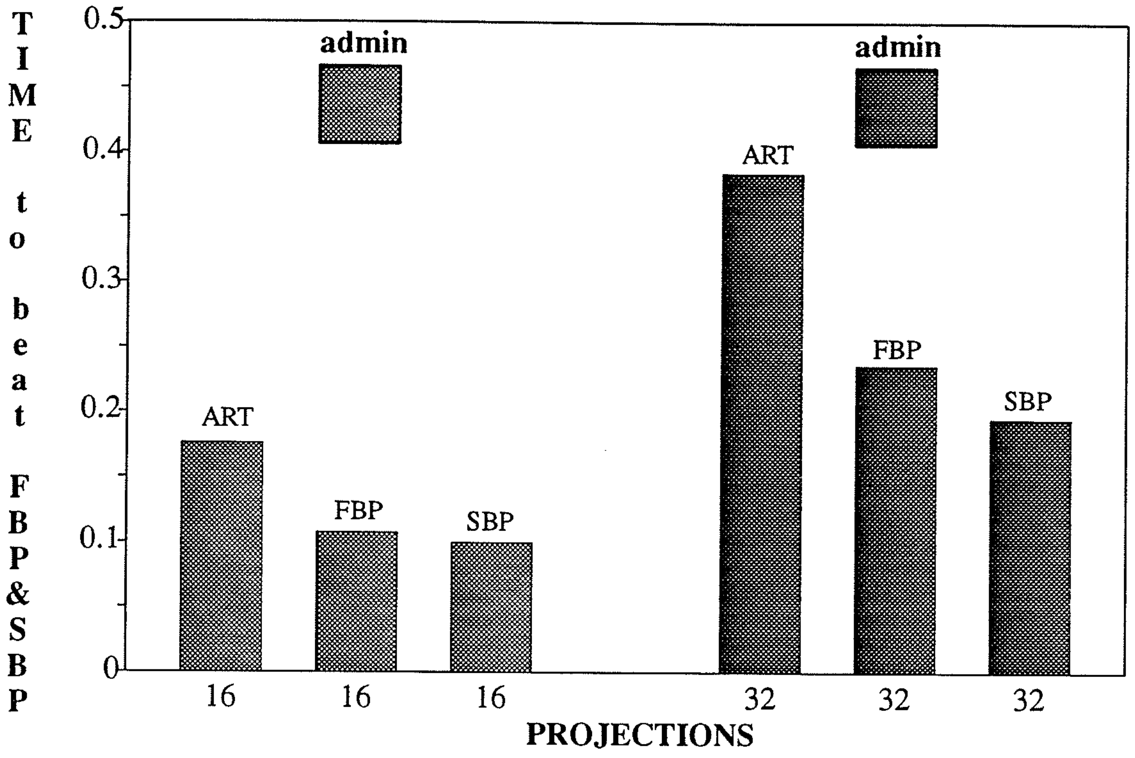


CHART 5 The time required for the ART algorithm to give a better reconstruction than the FBP or SBP algorithms. For our test images, only one iteration was needed; the time shown is for the complete iteration.

Overall Analysis of the Sequential Algorithm

The pictorial and numeric results correspond, indicating that ART can provide reconstructions superior to FBP and SBP with a penalty in computing time. The time to convergence of the ART algorithm can be quite large, too large for unspecialized hardware and much larger than commercially common algorithms. Although our results show that the time required for ART to out–perform the FBP algorithm is less than an order of magnitude, this result would be much worse for larger images and if actual ray integrals[11] or noisy data had been used. Our sequential simulations verified that a parallel architecture is needed for algebraic reconstruction algorithms; the vertex–8 array of processing elements is the architecture we propose.

2.2 ARRAY ARCHITECTURE

Our proposed vertex–8 mesh is shown in Figure 8 and the processing element (PE) in Figure 9. The array is a fine–grained SIMD data–flow machine, with one processing element (PE) per pixel. There are horizontal, vertical and diagonal connections between PE's so that each PE is connected to its 8 nearest neighbors. Each PE contains an ALU with summing functions and pixel storage and input/output routers with programmable pre–calculated routing tables. In addition to the 8 I/O routes, *initiate* and *terminate* route codes were added to the

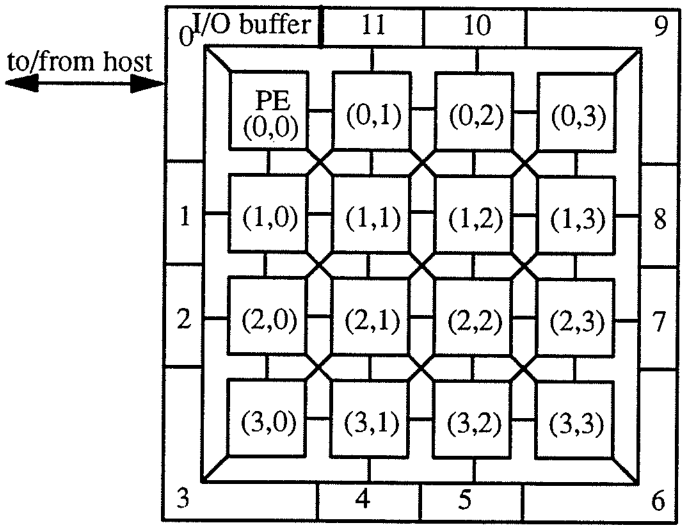


FIGURE 8 The proposed array, shown for a 4x4 image. The array can be controlled by a host or complementary on or off–wafer chip set.

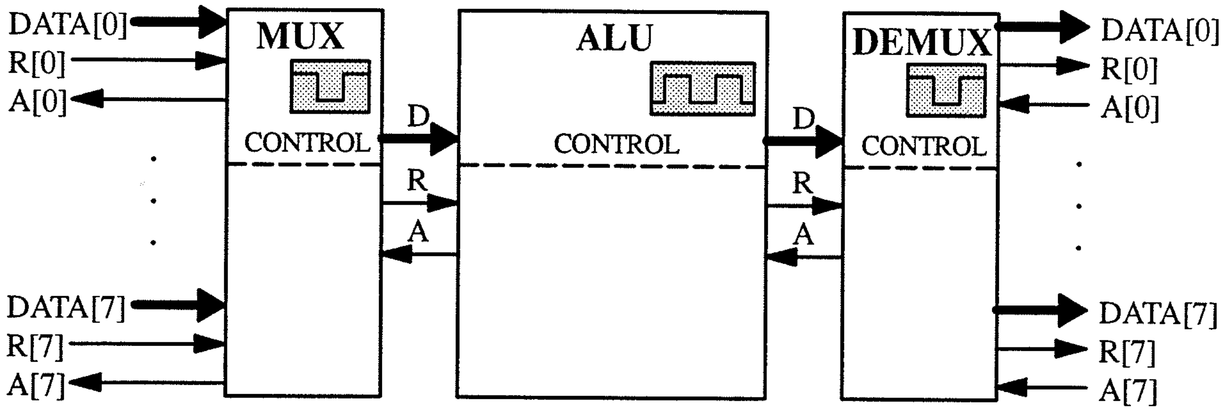


FIGURE 9 The processing element for the proposed array, with vertex 8 asynchronous communications. The PE's components are locally synchronous.

PE to make the array more flexible. Equations (7), (8), (17) and (18) for the ART and modified SIRT algorithm form the premise for our parallel mapping; recall that the modified SIRT algorithms are similar to ART except that the average Δf is applied to a pixel after a set of K Δf's (one for each of K discrete projection angles) have been calculated. The algorithms have in common a *project* mode, a *backproject* mode, and an *update* mode. Convergence detection will not be addressed in detail here; in our proposed array convergence would be detected by the host or the machine would be left in a continually iterating mode.

The parallel SIRT algorithm for one iteration would proceed as follows:

1. Pipeline one set of ray sums after another through the array in the *project* mode. Paths for the rays are already stored at the PE's. The input and output routing tables are calculated according to the ray membership function *ray number* = $\lfloor \varrho \rfloor = \lfloor x \cos\theta y \sin\theta \rfloor$.
2. As the ray sums exit, the host calculates and stores the change in pixel value Δf_m for each ray per equation (7). In place of the host, a linear array of custom processors of size $\sqrt{2N}$ could be used to calculate the updates.
3. Pipeline one set of pixel changes after another through the array in the *backproject* mode. For SIRT, the pixel change for a ray is scaled down by K to produce an average at the PE. The scaling means only summing need be performed locally. This will result in equation (17), $\Delta f_{m_{av}}$, being stored in each m^{th} PE.
4. *Update* each pixel value in parallel as in equation (18), using the average pixel change which is stored in the PE's to calculate the new pixel value f_m.
5. *Read* out the image if done. A *clear* command is needed prior to a *read* to place the pixel in the pipeline buffers. In the *read/write* mode, an image can be shifted in or out of the array horizontally much faster than could be done by *scan* mode register shifts.

Only 5 commands, *project, backproject, update, read/write,* and *clear,* are needed for ART, SIRT and a large number of other algorithms. For the reasons of resolution of the reconstructed image discussed in Chapter 1 we constrained the ray width τ to be less than or equal to the pixel width, which has important hardware implications since in the approximated algorithms it ensures that the next PE or pixel in a ray path will always be one of the 8 nearest neighbors of the current PE. Hence the choice of a vertex–8 PE. The result is that only a simple routing strategy is needed; the incoming and outgoing routes are pre–calculated and no queuing is needed for intra–mesh communications.

The mesh is a natural choice for image processing applications because the conceptual similarities between the physical and computational problem make this architecture efficient and make programming of the parallel machine easier. Many image processing algorithms, and especially reconstruction algorithms, require a large number of nearest neighbor computations. The reconstruction problem is a difficult one because it requires both local and global knowledge. The best time complexity that has been shown for the Hough transform on a vertex 4 mesh of size $\sqrt{N} \times \sqrt{N}$ [8] has been $O(\sqrt{N})$. We can achieve a similar time complexity for the SIRT algorithm described above, assuming the host (or associated $\sqrt{2N}$ linear array) can calculate the updates in *O(1)* time. The time complexity for the SIRT algorithm on our vertex–8 $\sqrt{N} \times \sqrt{N}$ array is also:

$$O(\sqrt{N}) \tag{20}$$

where K is the number of projections. The sequential algorithm would have at least $O(N)$ complexity. The processors in the mesh of vertex 8 processors must be simple and inexpensive since only $O(\sqrt{N}+K)$ time complexity is achieved with $O(N)$ processors.

2.3 PARALLEL SIMULATIONS

Simulations to verify that the parallel ART or SIRT algorithms could be mapped onto our proposed hardware were performed on a 4x4 array of Inmos 4 port T805–30 transputers, mounted in a box made by Transtech Inc. and running under the Ohio State *Trollius C* operating environment[27] and hosted by a unix workstation. Each transputer had 4 megabytes of memory. The box, called the MPC1000, allowed for a user configurable connection pattern. For our simulations the most efficient connection pattern is a mesh. The simulation programs were designed to closely model the intended hardware to provide a functional test of the PE's.

Lattard and Mazare [19] have performed image reconstruction on a transputer array. Our purpose is not to implement the reconstruction algorithm for permanent use on a general purpose transputer array, but rather to use that array for simulations that verify the functionality of our proposed custom hardware.

2.3.1 Program Implementation

Sixteen transputers with 4 physical ports each were available; thus the goal was to embed our proposed architecture onto the fixed number of transputers. Each software process running on a transputer represents in hardware a PE with programmable routing , with each PE process being able to receive and transmit data from or to its 8 nearest neighbors. Up to 32 processes can run on a transputer. The software processes were programmed to communicate in a 4 phase asynchronous software protocol by using a particular communications command set (*tsend* and *trecv*) provided by the *Trollius* operating system, which is important since this is the protocol we wish to implement in hardware. Routing tables for the PE's and I/O request and acknowledge patterns for the buffer were pre–calculated using the ray membership function $ray\# = \lfloor x\cos\theta + y\sin\theta \rfloor$ and loaded into the parallel programs before execution. The asynchronus nature of the receiving and sending processes combined with knowledge of pre–calculated routes guarantees that collisions cannot occur.

Figure 10 shows how the embedding was done, with an example of a 4x4 array of processes embedded on a 2x2 transputer array. We were not overly concerned with the effi-

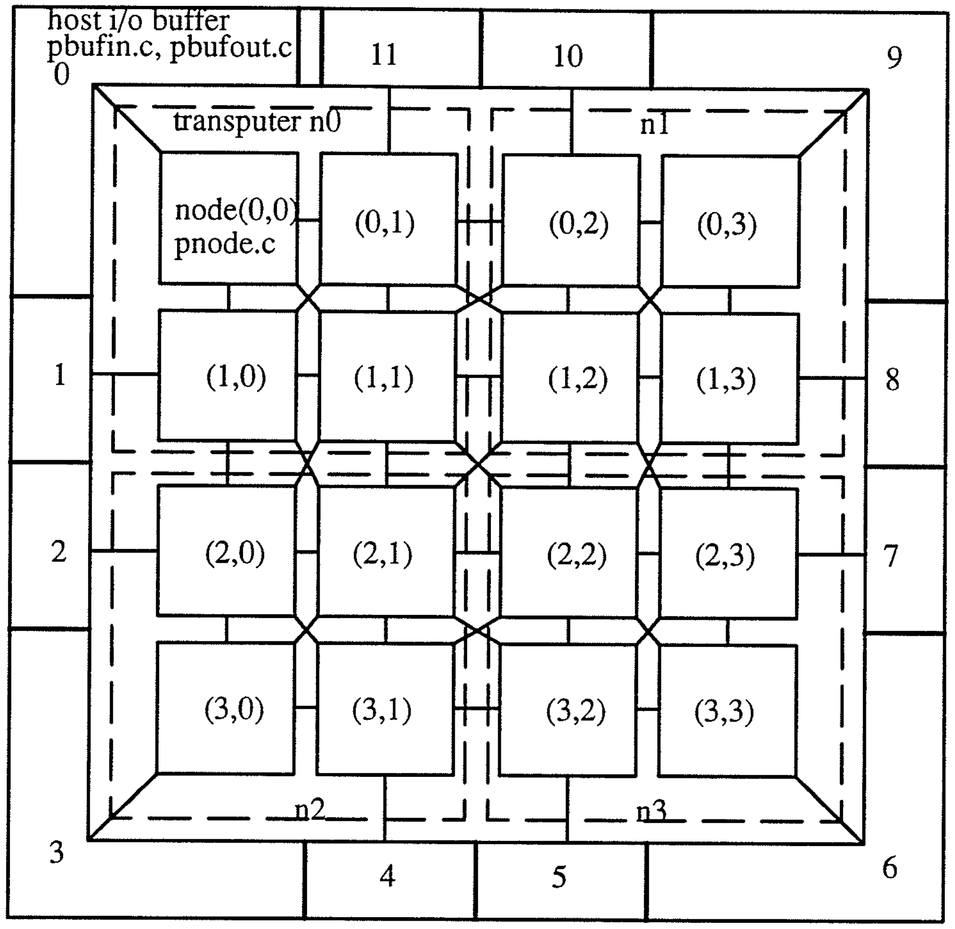


FIGURE 10 Typical software embedding of the processing elements in the transputer array.

ciency of the implementation on the transputer array, but in fact the mesh of transputers is the most efficient structure on which our 8 connected mesh could be embedded. Several processes had to be run on each transputer because of the limited number available. The programs *pnode.c*, *pbufin.c* and *pbufout.c* were written to accommodate a square image of size up to 128x128 on a square transputer array of size up to 4x4. Complimenting the node program *pnode.c*, the programs *pbufin.c* and *pbufout.c* perform the input and output buffering and also operate as a pseudo host for the algorithm. Our proposed hardware includes the node portion and the buffering, but not the host functions. In the planned hardware implementation, the host could be a general purpose computer or another dedicated chip set. The functions of the three programs and how they relate to the planned hardware implementation are described in the following.

2.3.2 Node

Each node process represents a processing element (PE) with 5 commands. The commands are PE functions that must be implemented in hardware. In hardware, the PE's would consist of a simple ALU containing registers to store f and Δf (pixel and delta pixel) and routers with local RAM to store the routes. The routes are loaded from a pre–calculated file and control a local multiplexer and demultiplexer. The *pnode.c* program takes x and y co–ordinates as its input, and must be loaded on the transputer node number that represents the quadrant (or more generally the tile) containing x and y. The *pnode.c* program was not designed to run 'outside the box' on the workstation with the other programs, but rather only on the transputers. The modes of the node process are:

<u>INIT</u>

Scan in mode. Loads in the processors registers and pseudo–RAM. The RAM contains the route programming. Global values such as the size of the PE array and the size of the transputer array are also read in; these would not be required in hardware.

<u>PROJECT</u>

Step through each of the processor routes and perform the project function, i.e. add the pixel value to the ray sum passing through the node. The data is passed according to the pre–calculated routes.

<u>BACKPROJECT</u>

Step through each of the processor routes and perform the backproject function, i.e. add the ray delta value passing through the node to the local delta value. The ray delta value has already been divided by the number of projections so that it will contribute only a portion to the local delta value and no division has to be performed locally to find the average.

UPDATE

Add the local delta pixel value to local pixel value.

STOP

Stop iterating. Received a DONE message from host. Otherwise repeat PROJECT, BACK-PROJECT, and UPDATE.

WRITE

Scan out mode. Write the final pixel value to a file.

2.3.3 Input Buffer

The input buffer program *pinbuf.c* performs both input buffering and host functions. It pipelines data into the mesh by initiating messages containing the ray sums and delta pixel values to the edge nodes of the arrays. Its pre–calculated input pattern is read from *bufin.pat.* The *pinbuf.c* program must also correctly reorder the stored delta pixel values before backprojecting them to the array. The modes are the same as for the *pnode.c* program. *Pbufin.c* is is STOPPED by a message from the *pbufout.c* program.

2.3.4 Output Buffer

Pbufout.c also has a pre–calculated output pattern. Besides acting as an output buffer for the ray sum, it acts as a host in calculating and storing equation (7). The Δf's are divided by the number of projection angles before they are stored for the *pbufout.c* program's use. This will result in an average update at the local PE's, as required by the SIRT algorithm. For our simulation, only rounded integer values were passed back to the PE's. The *pbufout.c* program also checks for convergence and sends a DONE message to all other running processes to tell them to stop iterating.

2.3.5 Needed Additions

Each of the three programs *pnode.c, pbufin.c, pbufout.c* require pattern files to control their routing. For the SIRT algorithm, these patterns are deterministic and programs can be written to calculate them. For our simulations, they were calculated by hand because only a 4X4 image was being tested. However, for larger images this would not be practical. External calculation of the routes is not as cumbersome as it seems. This is an *application specific* array that would likely be loaded only once, and externally calculating the routing gives you great flexibility in the number and type of algorithms that could be implemented on the array.

2.3.6 Parallel Simulation Results

Numerous simulations were done to prove the concept. Reconstructions of a 4x4 image from 2 and 4 projection angles using the SIRT algorithm are shown in Figures 11 and 12. The samples shown are for a 4x4 vertex 8 computational mesh, run on the 1x4 linear array of transputers. Recall that the 4x4 vertex–8 computational mesh is what we are interested in, not the underlying transputer hardware, since we plan an application specific VLSI implementation of a vertex 8 mesh. For the 4x4 simulations 16 software processes of *pnode.c* were running on the transputer array and one process each of *pbufin.c* and *pbufout.c* on the host workstation, all simultaneously. Undocumented compiler differences and hardware or operating system faults on the transputer machine were uncovered during programming.

For these simulations, only rounded integers were backprojected to contribute to the $\Delta f_{m_{av}}$ update values at each PE. This presented a problem since scaled fractional contributions of less than .5 would not contribute to the average update, making convergence not as accurate as it would be if the division required to calculate the average update value were done locally. To correct this, fixed point data could be backprojected in the *backproject* mode. This can be done in our proposed architecture because we have a 16 bit data buses. The 16 bit bus would provide enough resolution to backproject contributions resulting from a difference in 1 between the measured and calculated ray sums for a 256x256 image; for a larger image one would have to choose between a larger bus or less convergence accuracy.

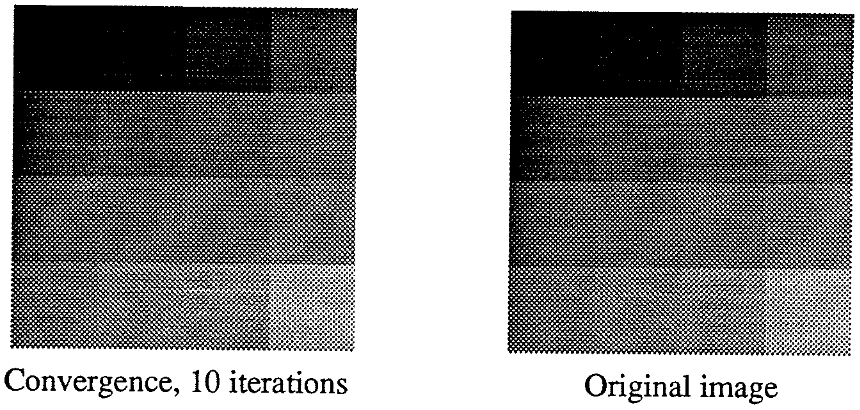


FIGURE 11 A parallel reconstruction of the image on the left using SIRT from simulated projection data from 2 angles.

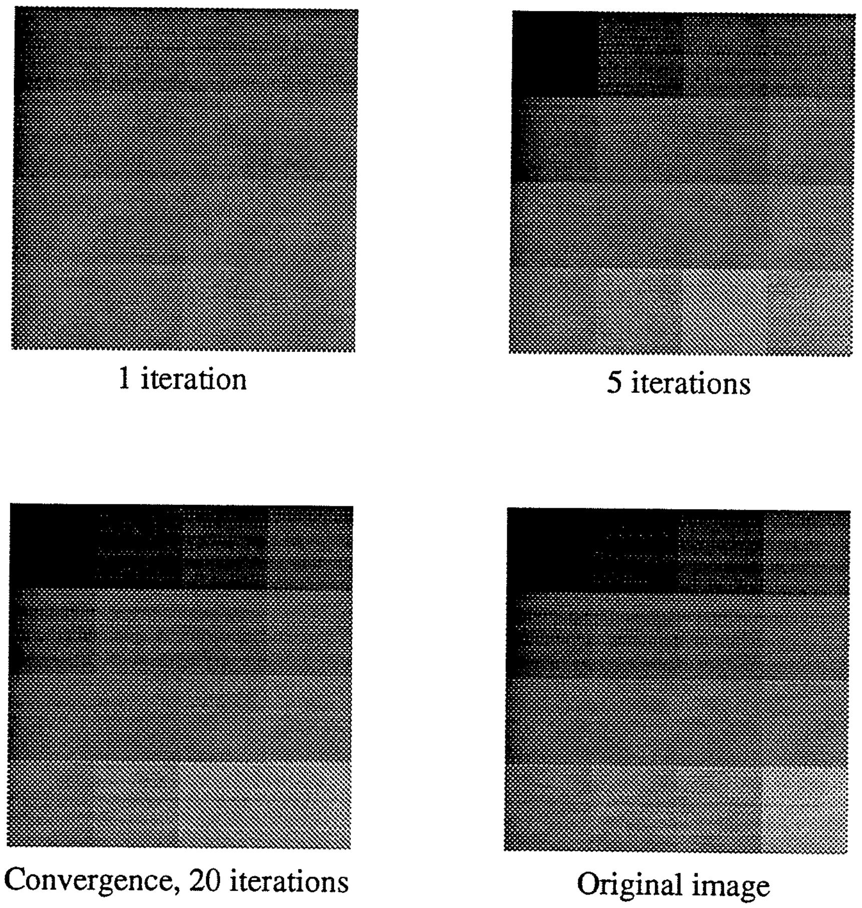


FIGURE 12 A parallel reconstruction from projection data from 4 angles.

The simulations showed the concept to be feasible and delineated the commands that would be required in hardware. Future simulation plans include reconstructing larger images (at least 32x32) from realistic projection data; the drawback is the limited size of the transputer array available. The parallel simulations identified improvements to our proposed hardware and were sufficient to provide confidence that mappings of the ART and SIRT algorithms onto the vertex–8 mesh of custom processors could be done.

CHAPTER 3

Processing Element Hardware Design

A processing element with the commands described in Chapter 2 was designed in a VSLI CMOS 3 micron technology and prototype chips have been sent for fabrication. The PE has several unique features that make it suitable for WSI, including a locally synchronous globally asynchronous communications protocol and a scan test pipeline overlay that connects an asynchronous scan chain to all registers for testing. As designed it would be suitable for use in an SIMD array, where the instruction can be changed when the array is in a steady state.

A globally asynchronous locally synchronous architecture was chosen because of well–known hardware limitations for WSI circuits[26]. The limitations are mainly due to clock distribution problems in synchronous design. Clock skew can result in data transfer errors and make fault tolerance, a necessity in WSI, difficult to implement. For example, in the the sketch on the following page (Figure 13) the data could be latched before it is valid due to the skewed clock between the distant spare and the local processing elements. The amount of skew and thus the locality of the spare elements must be taken into account in WSI reconfiguration architectures and algorithms. Other problems with large globally synchronous integrated circuits are the power spikes caused by simultaneous computation and that the entire chip may only operate as fast as the slowest part on it. As well, data is usually asynchronous in nature.

The globally asynchronous locally synchronous architecture is a compromise between the two design styles. It consists of clocked computation blocks with asynchronous

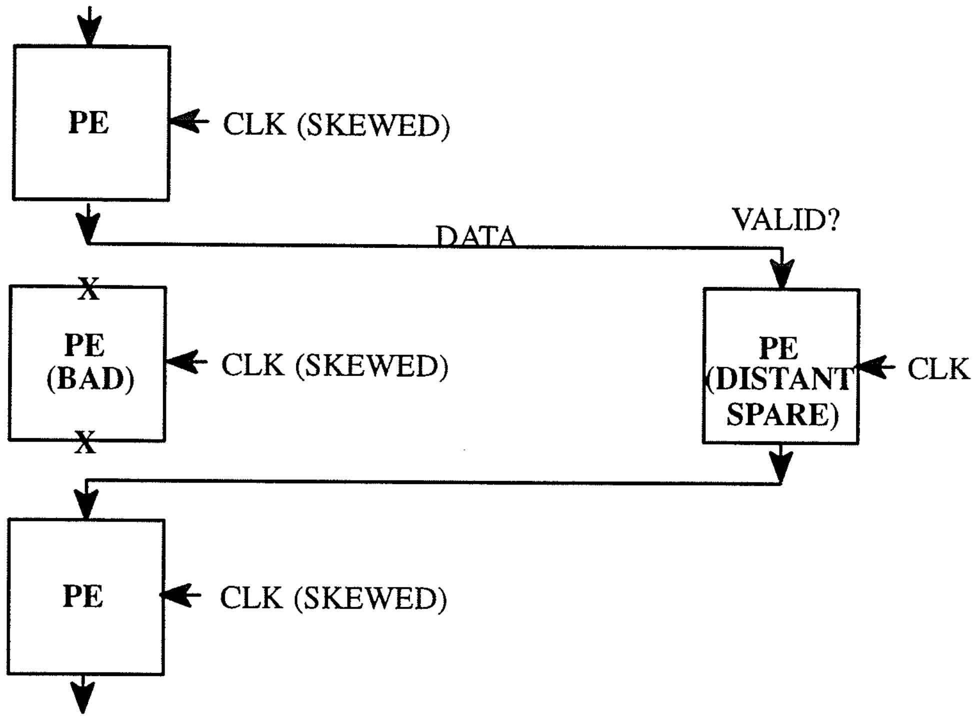


FIGURE 13 Problems with clock skew and fault tolerance. If the clock at the distant spare is leading the clock at the local PE's by too much, data will be latched too soon at the distant spare and be invalid. Similar problems will occur if the distant clock is lagging more than a threshold.

communication between them. The clock itself can be distributed without concern for clock skew since them synchronous islands can be made to approximate equipotential regions[31]. Global clock distribution is not necessarily required; internal clocks within the synchronous regions could be used, with the benefit of elimination of the power peak problem. Synchronous design procedures can still be used to a large extent, making the design process faster because of the existing knowledge base. Hybrid asynchronous/synchronous circuits require much less overhead than totally self–timed asynchronous designs. Such a system would have greater throughput since each part could operate as fast as possible (assuming local internal clocks), and would be easier to upgrade since a new faster part could be clocked at a different rate. This upgradability is important for integrated circuits as well as board level designs since with modern CAD tools upgrading a single component or unit on an integrated circuit for subsequent fabrication is commonplace.

For these reasons, the PE we designed was made to operate in a globally asynchronous locally synchronous environment. The drawback is in circuit overhead and the cost is in silicon area. As well, the much touted performance gain of self–timed circuits may be negligible because of the overhead; we will discuss these issues further in Section 3.12, *Area and Performance Considerations*, when we analyze the chip being manufactured.

The micro–chips were constructed out of a standard cell library, namely the cmos3dlm standard cell library supplied by CMC[2] for the Northern Telecom 3 micron double–level metal CMOS process[3]. A *standard cell library* is a set of primitive logic elements that contain physical, electrical, and behavioral descriptions suitable for simulation and layout of integrated circuits. The particular standard cells used here belong to the static CMOS logic family and contain *SILOS* and *hspice* descriptions. *SILOS* and *hspice* are two circuit simulators. Most of the simulations were done with SILOS using gates level models, and a few smaller parts with *hspice* at the transistor level. The design and simulation was done within the *Cadence 2.1*™ design framework, which uses a schematic capture design methodology. From a circuit schematic, one can generate a layout via semi–automatic placement and routing or a program listing suitable for simulation by software that has been integrated into the *Cadence* framework.

Asynchronous handshaking circuits were taken from Meng et al.[24] and modified to provide building blocks for communications circuits between the synchronous computation blocks. The synthesis procedure used by Meng guarantees hazard–free components, if one ensures that the modifications do not introduce any. We ensured this by design and veri-

2. CMC is the Canadian Microelectronics Corporations, established by the Natural Sciences and Engineering Research Council of Canada to promote and support microelectronics research in Canada.

fied it in simulations. The building blocks were cascaded and combined with *rendezvous* or *C* elements to produce more complex handshaking circuits without the need to repeat the synthesis process. The most efficient implementation does not result, but a more structured framework of asynchronous design is allowed than is presently available. Meng's synthesis procedure and original parts will be reviewed in the preamble, as well as the modifications to them.

Chips submitted for fabrication include an input router, output router, a 16 bit ALU with serial–to–parallel/parallel–to–serial converters, and a miscellaneous test chip. Serial operation was necessary here due to pin–out constraints of the prototype, this problem would not exist in WSI. One additional chip, the processing element shown in Figure 49, was designed but not submitted for fabrication. Each of the chips will be described in detail. The PE is an amalgamation of the aforementioned parts into one chip, and is capable of performing nearly all of the functions described in Chapter 2. In its present form, the PE would be suitable for calculating the Hough transform and for pattern recognition problems. Minor modifications to the ALU would be needed to perform image reconstruction, i.e. the addition of a temporary register *delta pixel* and logic to perform the *backproject* and *update* commands. This would not be difficult, as it involves only well–known synchronous design procedures. An additional needed part, the I/O buffer, was designed but not entered or simulated.

During design, a number of fundamental concerns arose that affect the potential of the PE to be used in a commercially acceptable WSI array. These are performance issues that fall into two categories, namely speed and area usage. The wafer must perform fast enough to achieve real–time reconstruction of images and must take only a reasonable amount of silicon. It will be shown that these are attainable goals for this architecture.

3.1 PREAMBLE: ASYNCHRONOUS BUILDING BLOCKS

The asynchronous building blocks were modified from circuit designs synthesized by Meng et al.[24] They use the common 4–phase communications protocol. Here we will briefly review the 4–phase protocol, review Meng's synthesis procedure to show that the circuits provide hazard–free operation and maximum concurrent computation, and explain modifications we have made to her circuits . The modifications added reset, completion detection and test capabilities to the building blocks. The building blocks include a pipeline handshake element, a multiplexer, and a demultiplexer. Many variations of the basic building blocks were used in our design.

The 4–phase signalling protocol can be best understood by examining the pipeline in Figure 14. *Rin*, request in, represents a completion signal from the previous computation

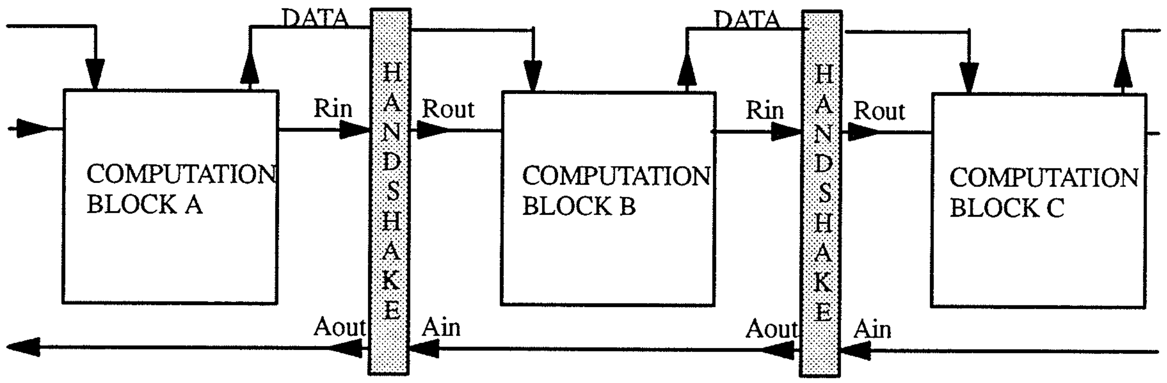


FIGURE 14 A pipeline.

block and *Rout* a request to the next computation block. *Ain* and *Aout* are the acknowledge signals. A typical handshaking cycle for an empty pipeline would look as in Figure 15. Transition 1, Rin^+, is a request form the previous computation block. Transition 2, $Aout^+$, means the handshake state has latched in data and the requesting computation block can continue with another computation. Transition 3 indicates that the previous handshaking block has received the acknowledgement and that the computation block is prepared to begin the next

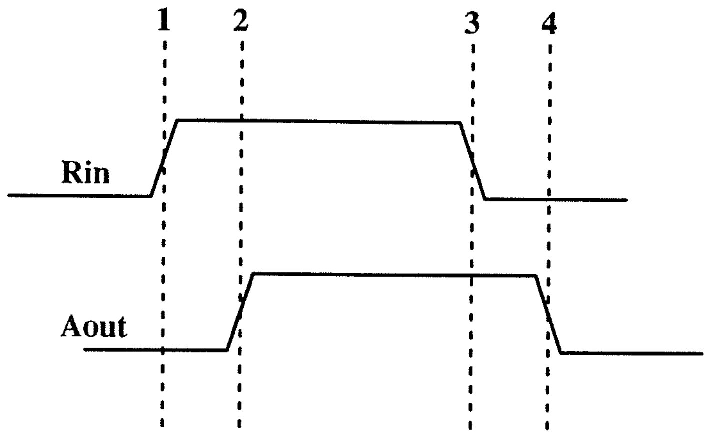
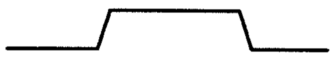


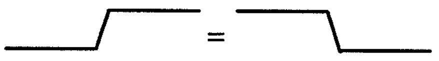
FIGURE 15 Typical 4–phase handshaking cycle.

computation. Transition 4 completes the cycle, dropping *Aout* to indicate that the handshake block is again empty and ready to latch in new data.

The 4–phase protocol is level–sensitive, in contrast to 2–phase signalling which is transition triggered. For example, for the 4–phase protocol

R/A = 

and for the 2–phase

R/A = 

The 4–phase protocol was chosen over the 2–phase because it has less circuit overhead. Although it requires an "extra trip" for the return to zero portion of the protocol, this time could be used to reset the clock counter in the computation blocks. Our design was not optimized to this extent. Two–phase signalling is more appropriate for long distant communications [31] than for the local communication our PE's will be performing.

3.1.1 Synthesis

In [24], Meng and her colleagues synthesize several handshake circuits using the 4–phase protocol with standard logic gates and RS latches. Their synthesis methodology is shown in Figure 16. This synthesis procedure is very effective for small systems but can become cumbersome for larger ones because of the difficulty in specifying the guarded commands, whose accuracy greatly affects circuit performance.

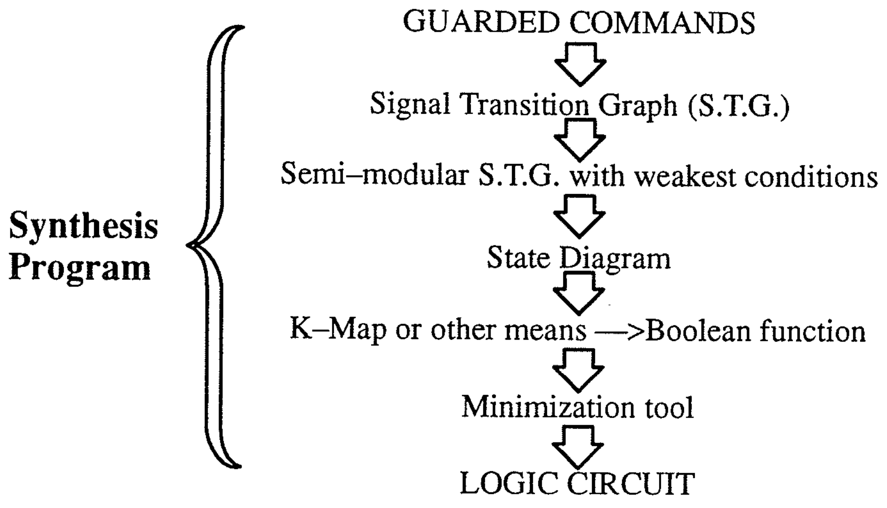

Synthesis Program
GUARDED COMMANDS
Signal Transition Graph (S.T.G.)
Semi–modular S.T.G. with weakest conditions
State Diagram
K–Map or other means —>Boolean function
Minimization tool
LOGIC CIRCUIT

FIGURE 16 Meng et al.'s synthesis methodology.

Guarded Commands

Guarded commands[9] are the input the the synthesis process. Their syntax is as follows:

Basic	$[C \rightarrow S]$	*C is a pre–condition of S, T/F*
And	$[C_1 \cap C_2 \rightarrow S]$	
Or	$[C_1 \cup C_2 \rightarrow S]$	
Sequential	$[C_1 \rightarrow S_1 ; C_2 \rightarrow S_2]$	
Parallel	$[C_1 \rightarrow S_1 \mathbin{//} C_2 \rightarrow S_2]$	
Alternative	$[C_1 \rightarrow S_1 \mid C_2 \rightarrow S_2]$	*Only 1 of C_1 & C_2 true at a time*
Repeat	$*[C \rightarrow S]$	

A set of guarded command specifications for the handshake block in Figure 14 is:

L.H.S.
$Rin^+ \rightarrow Aout^+ \rightarrow Rin^- \rightarrow Aout^-$

Relate input to output
$*[Rin^+ \rightarrow Rout^+]$ OR $*[Aout^+ \rightarrow Rout^+]$

R.H.S.
$Rout^+ \rightarrow Ain^+ \rightarrow Rout^- \rightarrow Ain^-$

Two possible specifications relating the input signal to the output signal of the handshake block are given, both of which satisfy the 4–phase signalling protocol.

Signal Transition Graph

Signal transition graphs (STG's) can be generated from the guarded command specifications. Examples from [24] of signal transition graphs from the above specifications are shown in Figure 17; many other STG's can satisfy them. The difficulty is in finding the STG with the 'weakest conditions' that will make the circuit hazard free while allowing for maximum concurrency. The optimum STG for a set of guarded commands can be found by deterministic means, and Meng has written a program to do this in polynomial time. Before and after STG's are shown in Figure 17. The 'after' STG will give a hazard–free circuit implementation, and is known as a *semi–modular* STG. Examining the 'before' STG's , you can see that the signal transitions on the left loops could happen many times before those on the right happen once; this indicates that the STG is not semi–modular and that a hazard could exist in a circuit implementation for the STG.

In the 'after' STG's, the computation cycles are shown in black. The STG on the left generated from the input specification *[$\text{Rin}^{+} \longrightarrow \text{Rout}^{+}$] has only 50% concurrency, while that on the right generated from *[$\text{Aout}^{+} \longrightarrow \text{Rout}^{+}$] has 100% concurrency. The handshake blocks in our design have 100% concurrency.

State Diagram, Karnaugh Map, & Logic Minimization

After the STG is obtained, the synthesis process is straight forward. A signal state diagram can be derived from the STG, and standard boolean minimization is used from there.

The logic circuit resulting from this particular synthesis procedure is shown in Figure 18, as implemented in our design. Muller C elements resulted from the synthesis, and appear as RS latch and NAND gate combinations as in Figure 19. As Meng notes, an RS latch cannot be a hazard–free implementation of a Muller C element in <u>any</u> circuit with unbounded delays, but only in circuits with unbounded delay derived as above from a semi–modular

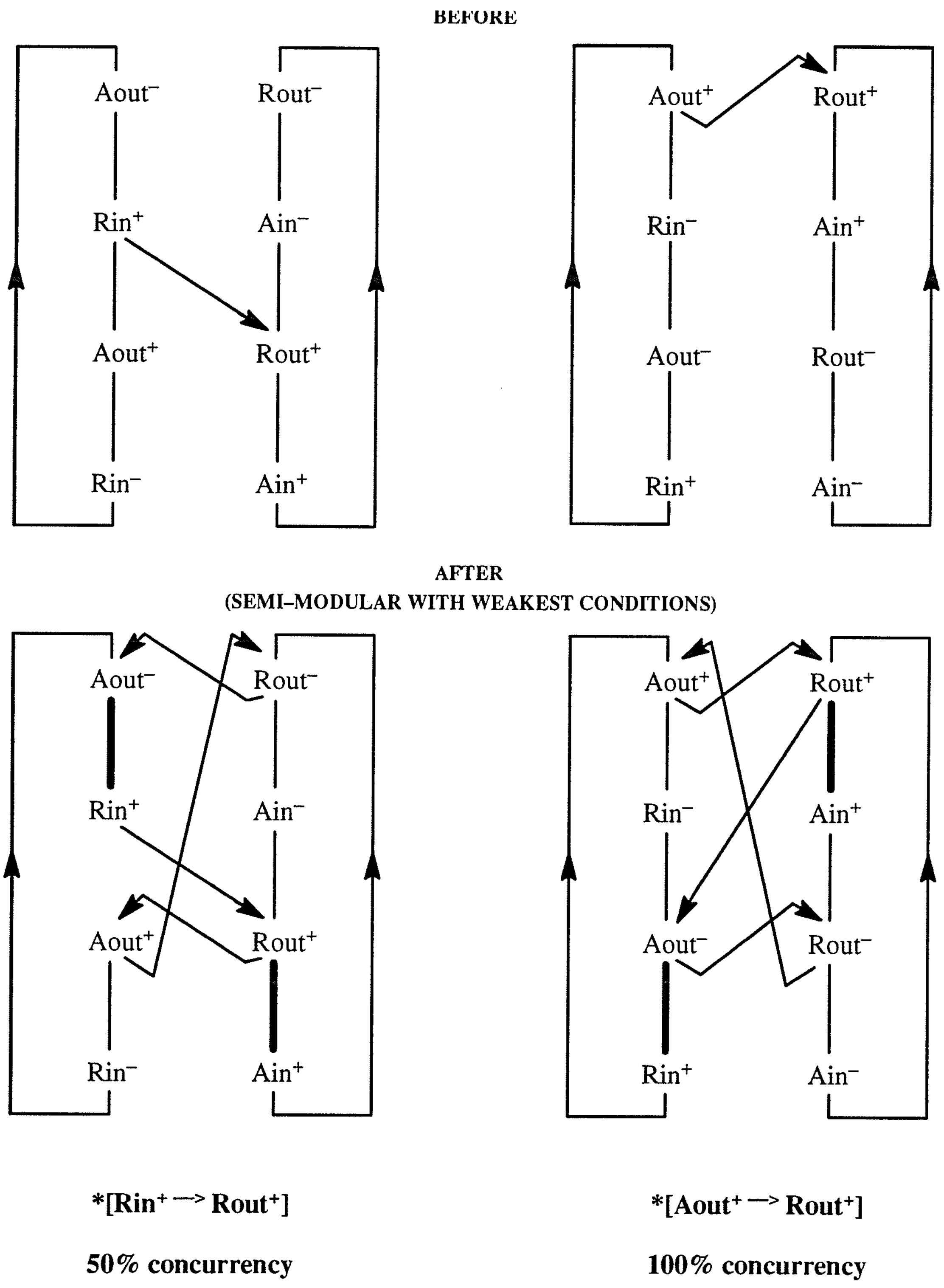
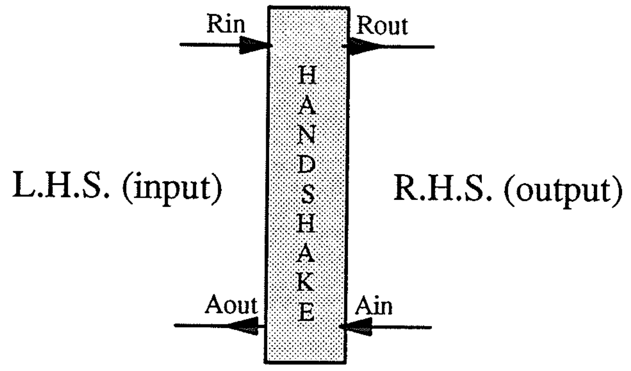


FIGURE 17 Signal transition graphs for two different guarded commands specifications, before and after conversion to a semi–modular form. The dark lines are computation cycles.[24]



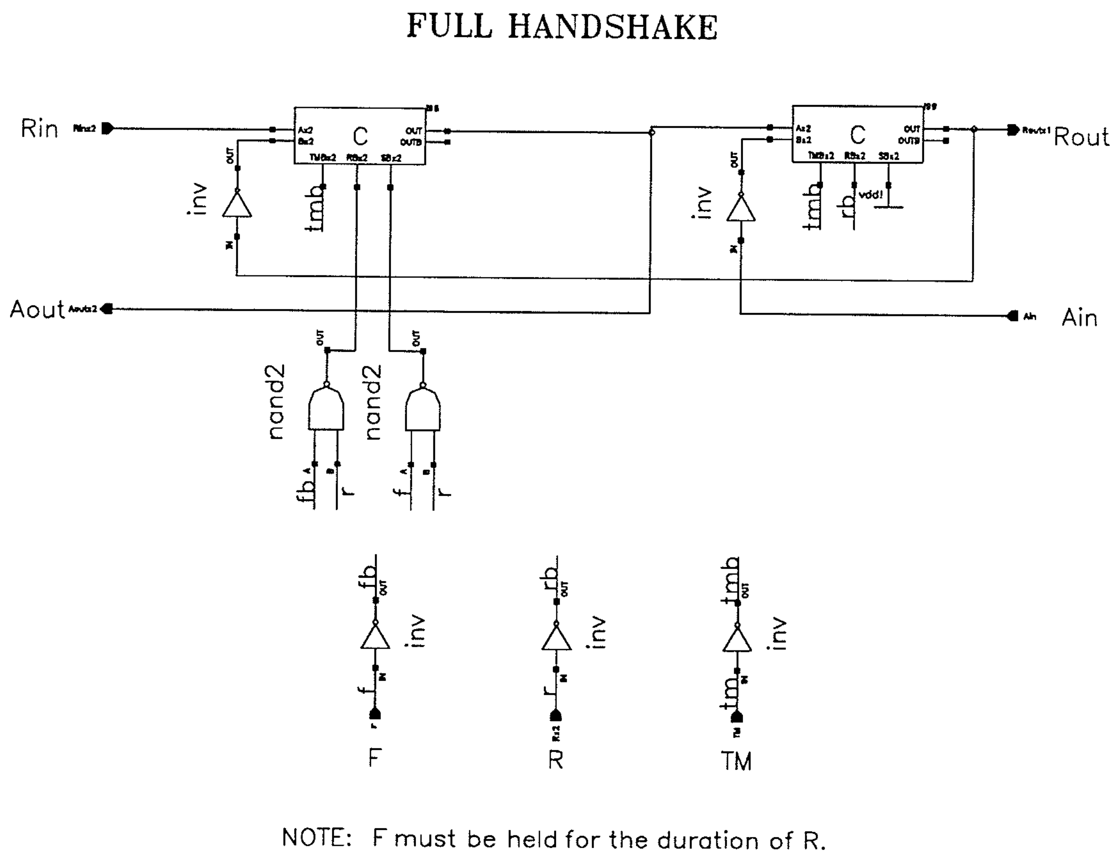
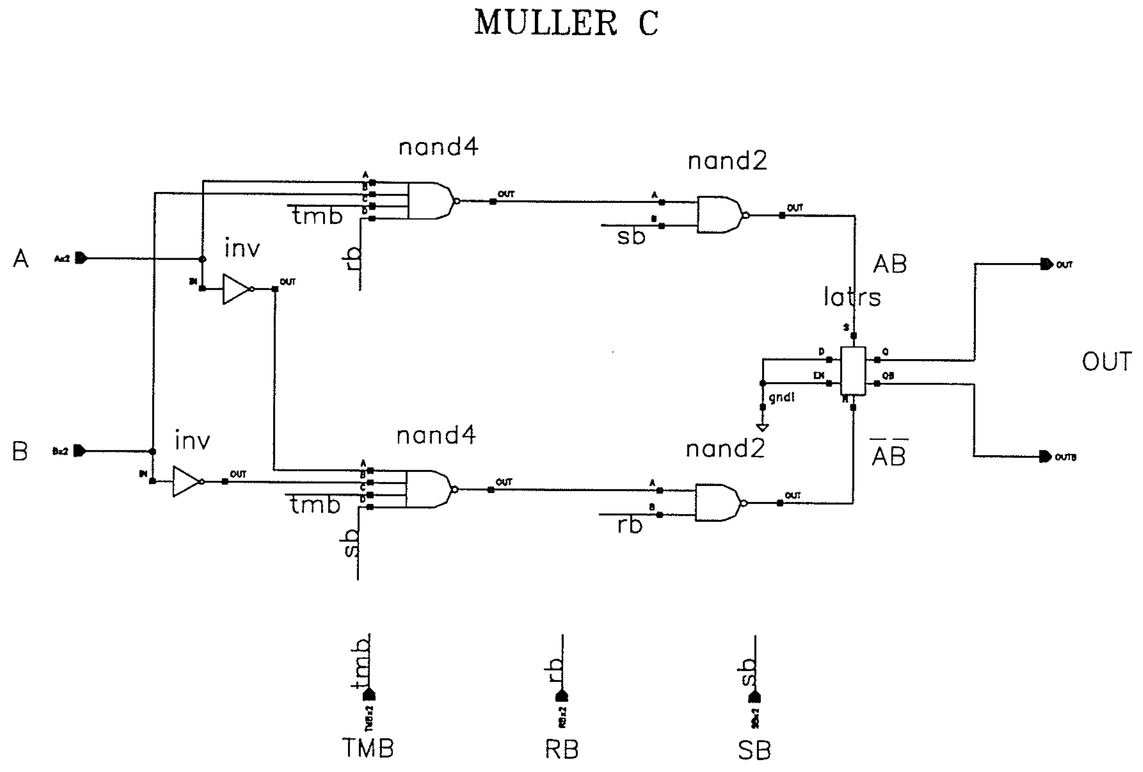


FIGURE 18 Handshake circuit for a pipeline. The circuit was modified to include initialization and test signals. Delay elements were added to this circuit to provide register latch completion signals.



NOTE: R=S=1 invalid.

FIGURE 19 C element implemented with SR latches.

STG. This implementation of the Muller C element is usually okay in circuits with realistic delays. Signals that have been added for initialization and testing do not effect the circuits dynamic behavior and thus have no effect on the validity of the synthesis.

<u>SIGNALS</u>

Rin	Request in.
Aout	Acknowledge out.
Rout	Request out.
Ain	Acknowledge in
F	Full. Initializes the pipeline to full on reset when set, i.e. *Aout*=1, *Rout*=0. Empty is *Aout*=0, *Rout*=0.
R	Reset.
TM	Test mode. Disables *R* and *S* of the RS latches.

Register Addition

The handshake circuit thus far would suffice as a pipeline element, that is, it would propagate the correct states, but it would not be very useful without valid data to pass to and from computation blocks. Pipeline registers must be added to the handshake elements. $Aout^+$ is the only signal (in concurrently operating pipelines) that can be used to latch the data without corrupting the input or output data of the computation blocks.

Heuristically, one can see this could be done without adding hazards by naming the original *Aout* a sub–signal, *Lout*, and using a latch completion signal as the new *Aout.* Meng has done this formally by adding *Lout* as a latch signal and *Lin* as a completion signal and repeating the synthesis process. We have included *Lout* in our design to trigger positive edge trigger flip–flops, *Lin* as a return signal, and used a delay element to provide the completion signal for a single register latch operation as shown in Figure 20. The separation of *Lin* and *Lout* in our design allows registers to be chained together. The trigger from the last flip flop or set of flip flops to be triggered becomes *Lin,* so that any number of registers can be triggered from the same $Lout^+$ transition without upsetting the handshake signalling and without the need for various sized delay elements. The delay element should be roughly equal to the clock hold time for the register in question, 18 ns for the cmos3dlm D flip flop. This concept is the same as getting a completion signal from the last register to be latched in the chain and is important for operation of the unique asynchronous scan chain in our design, which will be discussed in Section 3.4 *Asynchronous Scan Test.*

Building Blocks

Asynchronous parts that were synthesized by Meng and modified for use as building blocks in the same manner include a pipeline handshake element, a 2–input multiplexer, and a 2–output demultiplexer. Block diagrams and circuit implementations of the most basic im-

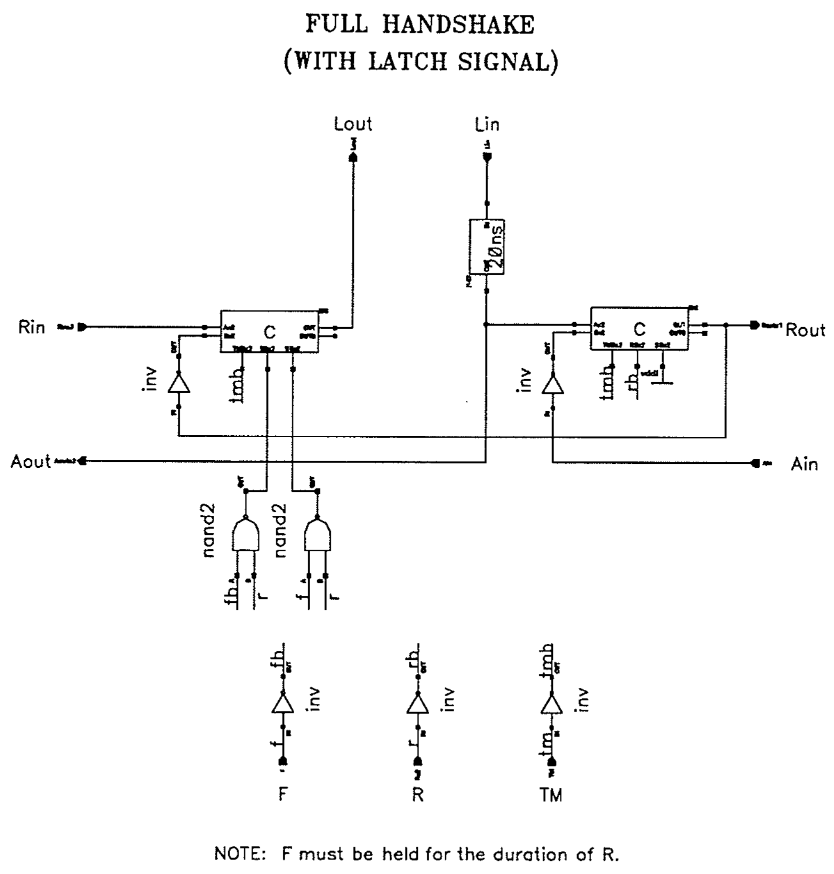


FIGURE 20 The handshake element with latch signals and a delay element to provide a completion signal.

plementations of the multiplexer and demultiplexer are shown in Figures 21 and 22, and a typical simulation in Figure 23. The mux and demux each have an additional asynchronous signal interface, consisting of *Cin, T,* and *Cout,* that interconnects with a controller. *T* selects the input or output port and must remain valid through the handshake with the controller, i.e. from Cin^{+} –> $Cout^{+}$ –> Cin^{-}. Data can be latched via *Aout* as was done for the handshake element, or occasionally by *Cout* in certain portions of our design that are sequential.

The handshake, multiplexer, and demultiplexer and variations of them were used extensively in our design. They are self–timed asynchronous parts that are hazard free. By

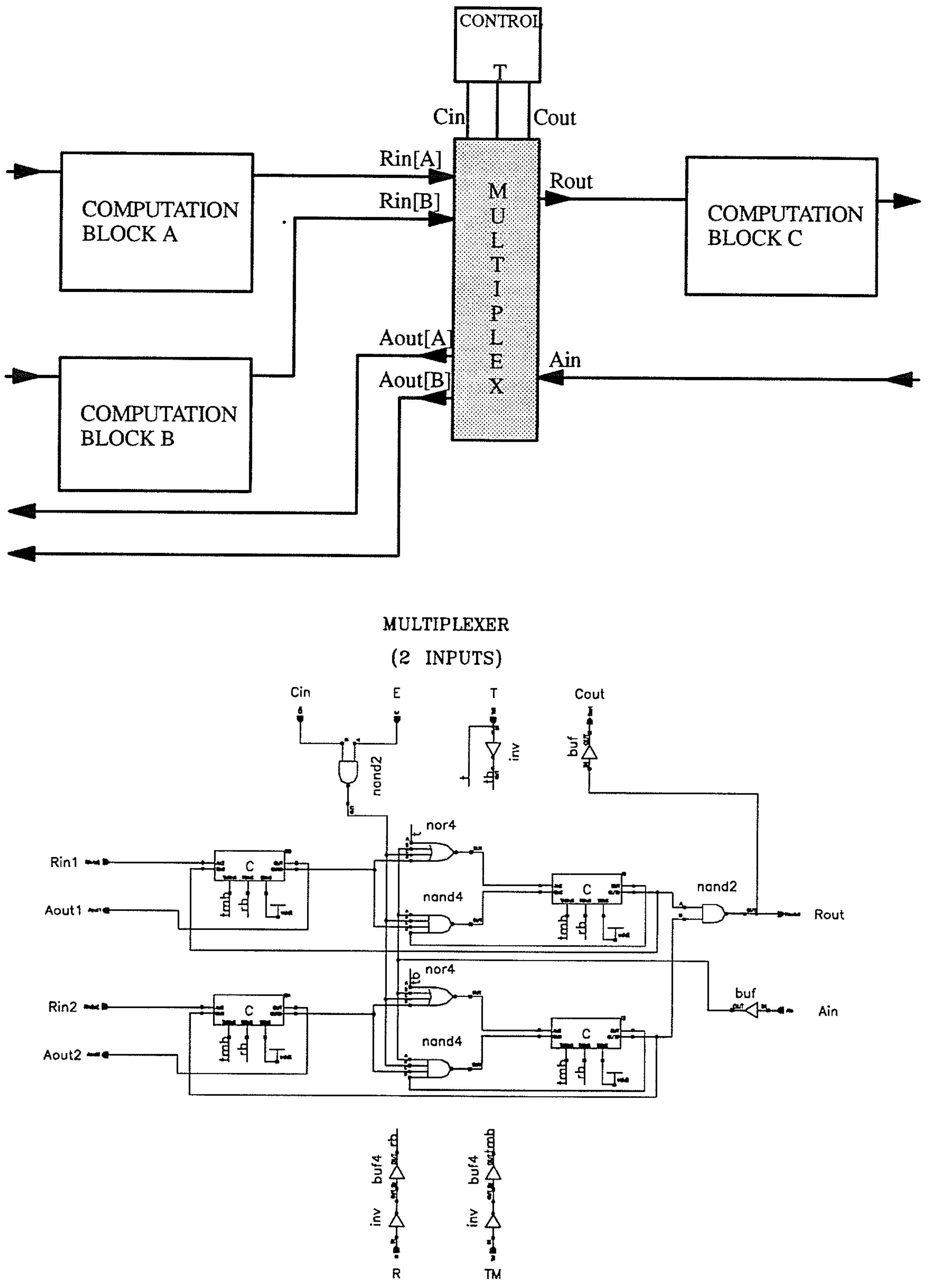


FIGURE 21 The multiplexer. The block diagram is from [24]; the circuit has been modified to add reset, enable, and test mode controls. Variations allow data to be latched by either *Aout1*, *Aout2*, or *Cout*.

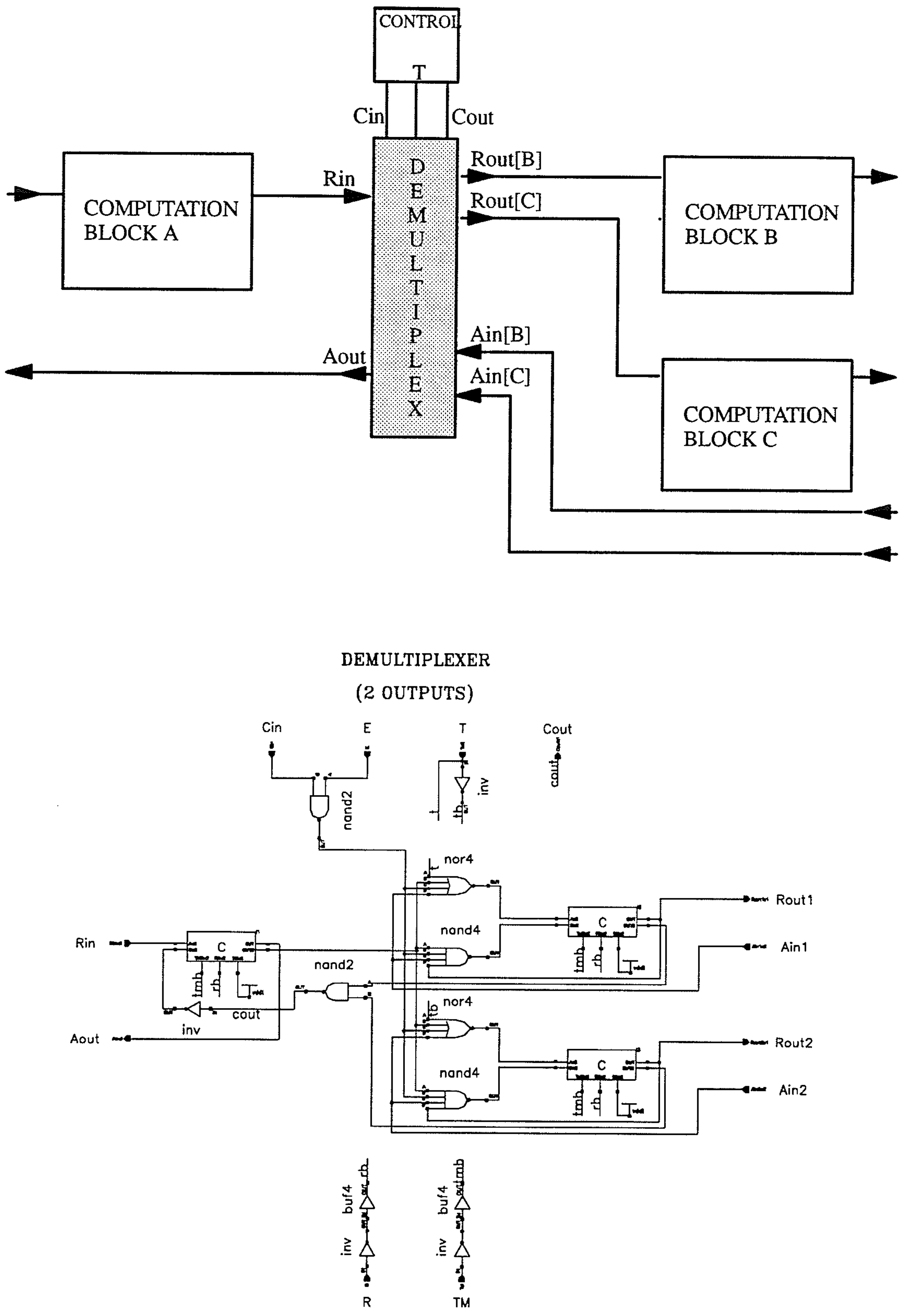


FIGURE 22 The demultiplexer. The block diagram is again from [24]; the circuit has been modified to add reset, enable, and test mode controls. Variations allow data to be latched by either *Aout* or *Cout*.

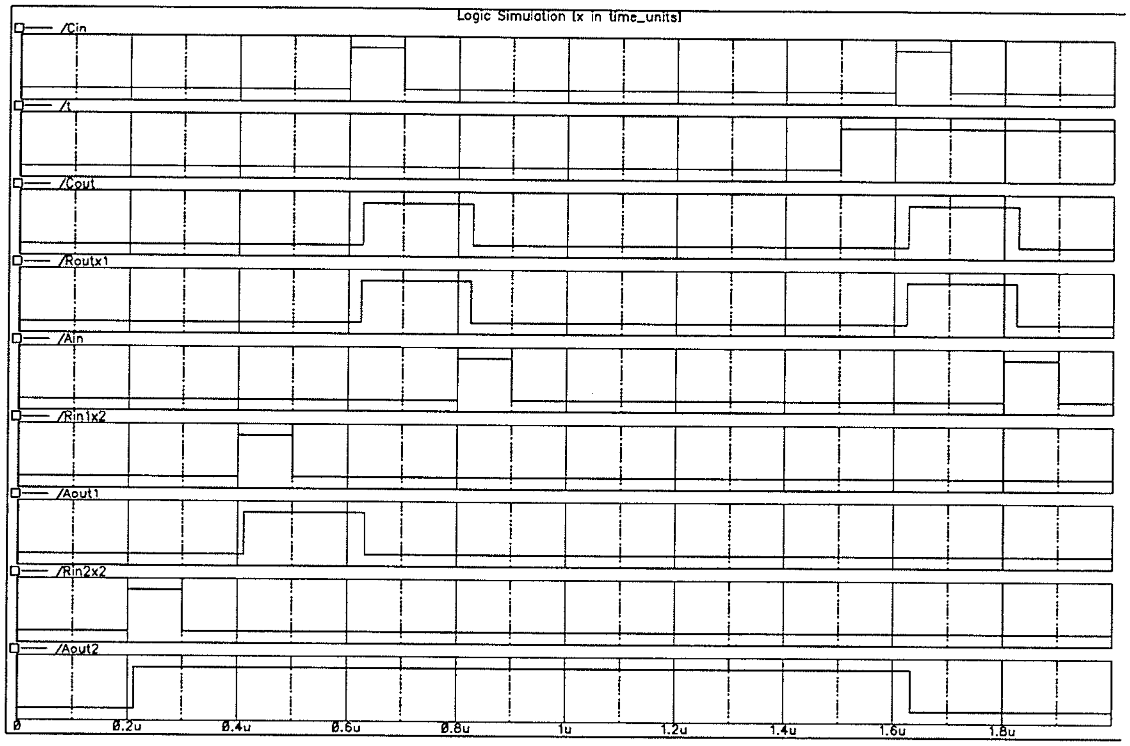


FIGURE 23 A simulation for the 2 input multiplexer handshake circuit. The request/acknowledge cycle follows the 4–phase protocol. *Cin* and *Cout* are the controller handshake signals and *t* selects input 1 or 2.

cascading and combining them, communications links suitable for a processing element in an 8–connected mesh can be constructed.

3.2 C & DELAY ELEMENTS

Although these are relatively small components, the C element and delay (or completion) circuitry bear further discussion since they are used so frequently in the design and thus have a big effect on area usage. They were implemented inefficiently in our design and alternatives must be consider for future work. There are many alternatives.

C Element

The C element is a common component in self–timed designs. Its truth table is shown in Table 1 . The building blocks use RS latches and standard boolean logic gates to make C

A	B	OUT
0	0	0
0	1	OUT(*t–1*)
1	0	OUT(*t–1*)
1	1	*1*

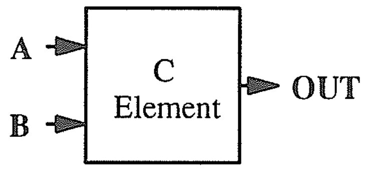


TABLE 1 C element truth table.

elements. In these circuits, a true integrated C element could be substituted for the RS latch version.[3] Our implementations used only the standard logic cells available in the cmos3dlm library. A much more area efficient and faster C element could be made at the transistor level; there are many suggested implementations in the literature. The C element should be added to the standard cell library.

Delay Element

The delay element was chosen to be 20ns after extensive simulations of an pipeline with *Rout* and *Ain* shorted to mimic a external pipeline stage that would run as fast as physically possible. The size of the delay is a limitation in itself, but for now we will only consider its implementation. For our design , the delay element was constructed from buffers and inverters for the standard cell library. This is a very inefficient means since the standard cells are optimized for speed, but was done because the specifications for the standard cells are more accurate than what could be assumed for a newly designed delay element that has not been verified by fabrication run tests. The amount of the delay using standard cells will still vary somewhat with each fab run, but the delay variation be relatively equal with respect to other

3. Although the reverse is not necessarily true in a circuit with unbounded delays. See Section 3.1. As well, the C element for these circuits must be an atomic C element; see [24] page 1197 for further references on integrated C element designs.

gates. Guard time was allowed to account for fabrication run variations and other factors relating to the automatic place and route tools.

A better way to implement the delay element could certainly be found. One simple way to increase the accuracy of the delay would be to use a fixed sized metal capacitor at the output of each buffer, but this also would not be area efficient. We propose that *clock stealing* be used. The concept is shown in Figure 24. Clock stealing would require 2 D flip flops, or could be integrated into a new standard cell. The delay would be of varying lengths, with a minimum of $^1/_2$ clock cycle and a maximum of 1 $^1/_2$ clock cycles assuming a 50% duty cycle. Clock skew would not be a concern. The fact that the amount of minimum delay could

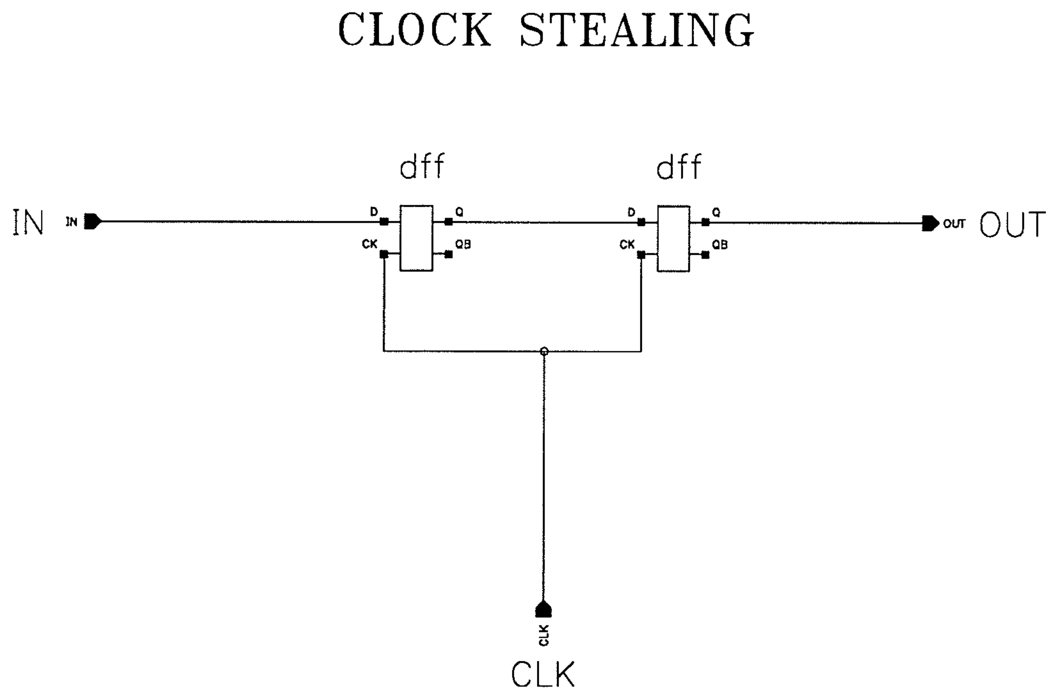


FIGURE 24 Clock stealing circuit. If the clock has a 50% duty cycle the minimum delay would be $^1/_2$ cycle and the maximum 1 $^1/_2$.

be varied would eliminate dependence on fabrication process parameters. If clock stealing is used, one would want to minimize the number of asynchronous communications links or use a separate faster clock for the delay elements since there would be a loss in performance.

Completion Signal

One must remember that the purpose of the delay element is solely to provide a completion signal for register latching. Other methods could be used to provide a completion signal for static CMOS registers, such as quiescent current monitoring. If the overhead penalty is not large, this could be a promising technique. Another alternative would be to use another type of register logic that provides a completion signal. This is an area of active research.

3.3 PIPELINE

Construction of a pipeline from the handshake element may appear straightforward, but there are some concerns for an integrated circuit implementation. These are the selection of the delay element and data/request bundling.

To decide on how large of a delay element should be used, the test pipeline shown in Figure 25 was entered for simulation, with *Rout* and *Ain* essentially shorted. The handshake circuit used in the pipeline is shown in Figure 26, and Figure 27 shows a simulation with the pipeline running at 5 MHz; successful operation was observed up to nearly 10 MHz.[4] The pipeline appears infinitely long to the input, with the buffer at the output looking like a very fast handshake element. To find the correct amount of delay, an initial value was estimated from the standard cell timing parameters. Only the cells in one handshake block were taken into account and not those in adjoining blocks since our goal was to make each

4. Note that in simulation illustrations, some signal transitions may appear to fire before others when they actually do not because they are not shown 'zoomed in'. Accurate timing measurements can be taking within the *Cadence* waveform window by zooming in on the desired signal transitions.

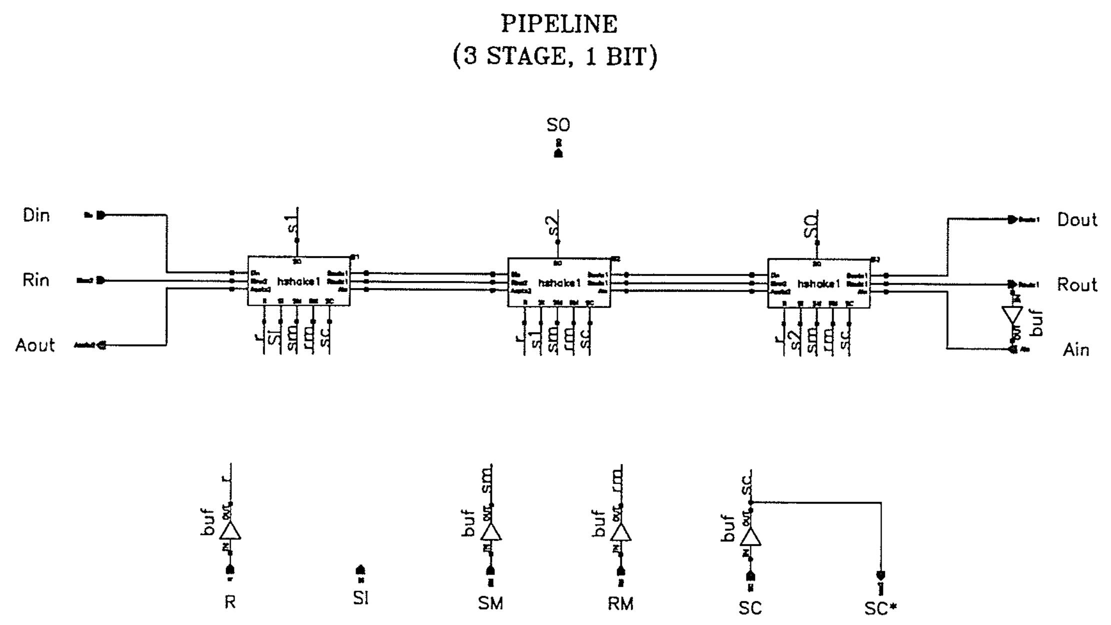


FIGURE 25 A 3 stage by 1 bit test pipeline.

stage independent of timing variations outside of its own domain. The delay element was then adjusted, and the simulation re–run. Finally, a minimal delay value was chosen such that $Rout^+$ occurred 5 ns after the data the data was valid. Only one fixed size delay element is needed for all the handshake circuits regardless of the number of registers chained together due to the use of a latch trigger return signal, *Lin*.

A distinction between a *self–timed asynchronous* circuit and just an *asynchronous* circuit is necessary. At first glance a guard time of 5 ns seems rather small for an integrated circuit where process parameters can cause variations in timing specifications, but this is in fact a conservative figure. That amount of delay was chosen do make the handshake block *self–timed*, that is its correct operation is not affected by the timing parameters of adjoining circuit elements. Surrounding pipeline stages could be infinitely fast and not cause a failure. In reality surrounding stages are not infinitely fast so their delays and wire delays could be taken into account to optimize performance. The delay element could be made considerably

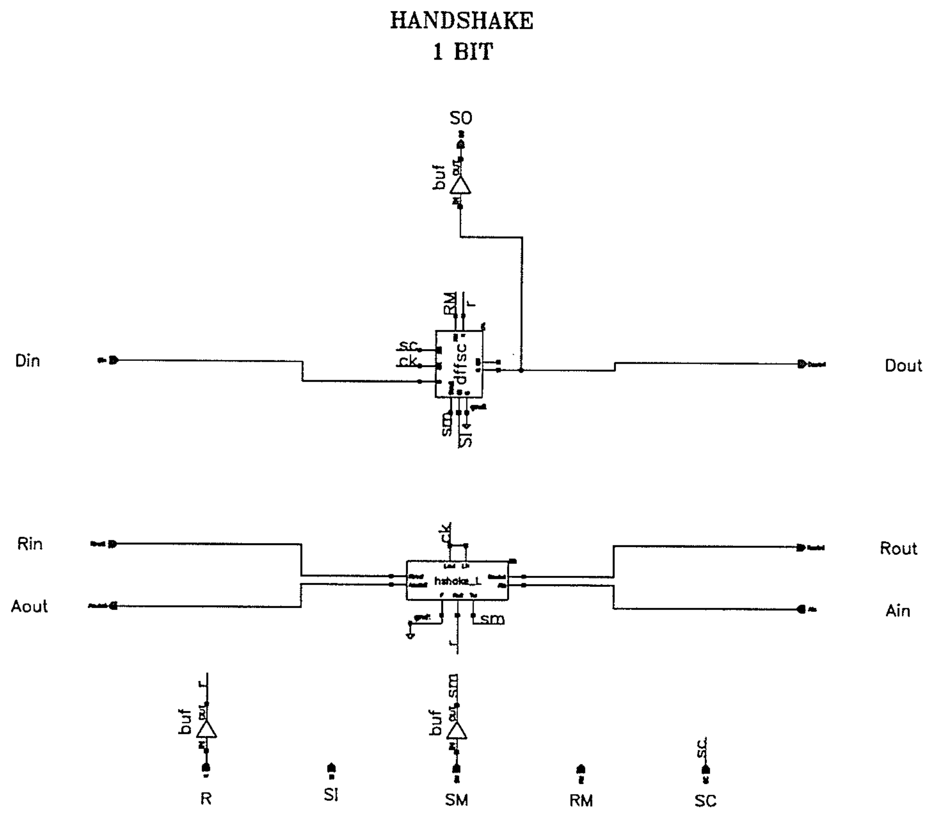


FIGURE 26 The handshake block for the 3x1 bit pipeline.

smaller and the pipeline would still be a valid asynchronous circuit; simulations indicate that the delay value of 20 ns is a least 1 $^1/_2$ times the required value. The delay value in our design was not optimized further out of concern for fabrication process parameter variations. The delay value greatly affects performance since for each handshake, double the delay value is added to the latency.

Another potential problem, akin to clock skew, is with data and request bundling and the automatic place and route tools used to generate the layout. The *data out* and *request out* signals of pipeline stages or computation blocks should be treated as a bundle; that is the delay of *Dout* in reaching the next block must be less than or equal to the delay of *Rout* plus the guard time. Wire capacitance will affect signal delays, and with automatic place

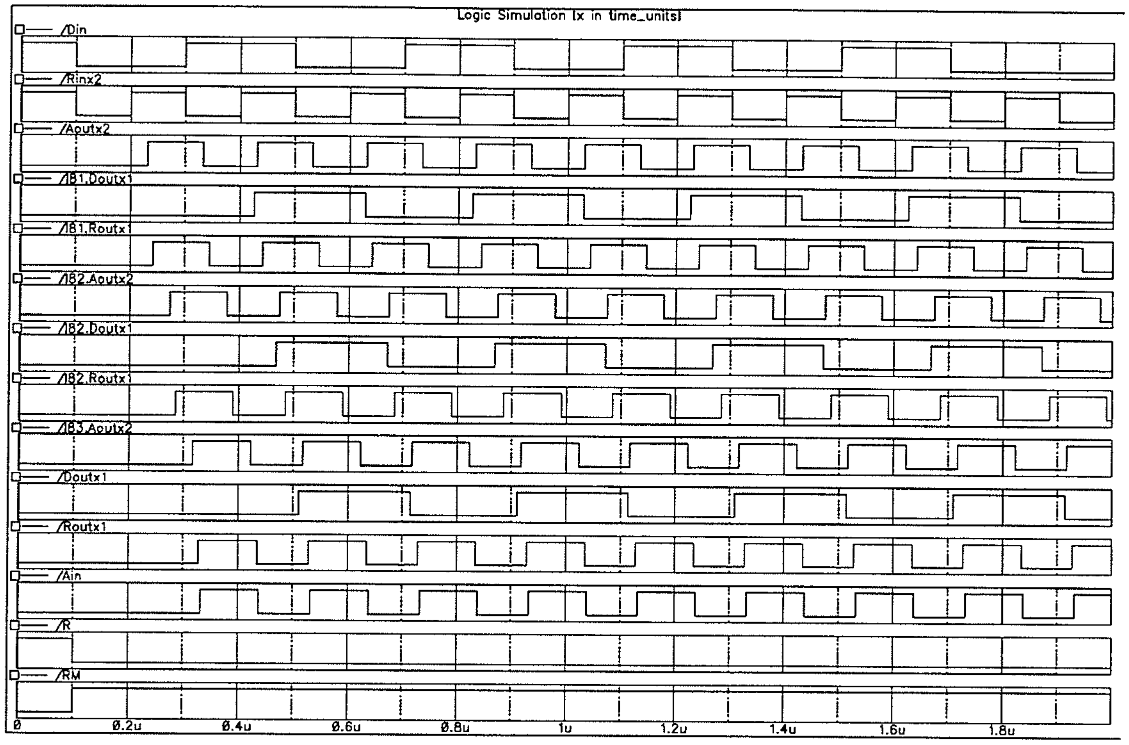


FIGURE 27 A simulation of the 3x1 pipeline with 20ns delay elements. Alternating data was applied to the input and passed to each successive stage. The signal *Din, I81.Dout, I82.Dout,* and *Dout* show the data rippling through the pipeline.

and route tools there can be no guarantee that the wire lengths for *Dout* and *Rout* will be approximately the same. Wire capacitances were not take into account in in the *SILOS* simulations, but calculations using timing parameters from the standard cell specifications indicate that with the guard times used, there is not likely to be a problem. From the specifications for a standard cell RS latch the added delay due to wire length for $Rout^{+}$, $Dout^{+}$, *and* $Dout^{-}$ would be:

$$\Delta t_{Rout^{+}} = 20.11 l C_w \tag{21}$$

$$\Delta t_{Dout^{+}} = 20.17(l + \Delta l) C_w \tag{22}$$

$$\Delta t_{Dout^-} = 22.56(l + \Delta l)C_w \qquad (23)$$

where l is the length of the wire, Δl is the additional wire length on *Dout,* and C_w is the capacitance per unit length. C_w is $2.7\text{x}10^{-5}$ pF/um^2 for the *metal 1* layer, making C_w $2.7\text{x}10^{-5}$ pF/um for the 3 micron process. Taking a nominal value of 3000 microns for the length l of the shorter wire, *Rout*, and setting $\Delta t_{Dout^+} - \Delta t_{Rout^+}$ and $\Delta t_{Dout^-} - \Delta t_{Rout^+}$ equal to the tight guard time of 5 ns gives $\Delta l \approx$ 3000 and 2400 microns respectively. This means the *Dout* wire would have to be 2400 microns longer than the *Rout* wire for the circuit to be unreliable. Since the maximum chip dimensions are 10000x10000 microns this is unlikely to happen, but more attention would have to be given to the routing in a WSI environment.

To further reduce the probability that wire length between *Rout* and *Dout* will differ greatly, *macros* were used in our design and the *Dout* and *Rout* pin were placed near each other on the macros. Macros are just chunks of standard cells that are placed and routed separately before the whole design is done. This localizes the placement and routing for logical blocks and makes major differences in the wire lengths less likely, as well as allowing the various parts to be physically identified on the layout. The use of macros also makes circuit overhead calculations easier.

More study is needed to see how much the delay element can be reduced or if some other completion detection method is more appropriate. Our design was very conservative, with each handshake component being self–timed. The 3 stage 1 bit pipeline was sent for fabrication to verify the simulations and analysis.

3.4 ASYNCHRONOUS SCAN TEST

All components in our design include control and data lines for asynchronous scan test. Figure 28 shows a 3 stage by 16 bit pipeline with scan test, and Figure 29 shows a simplified view of its internal scannable registers in *run* mode and *scan* mode. The addition of another handshaking element, *hshake_sc,* creates a second pipeline overlay that takes control of all register latching when scan mode is entered. This can include registers in clocked computation blocks. Figure 30 shows a test configuration for the circuit and Figures 31 shows simulation results.

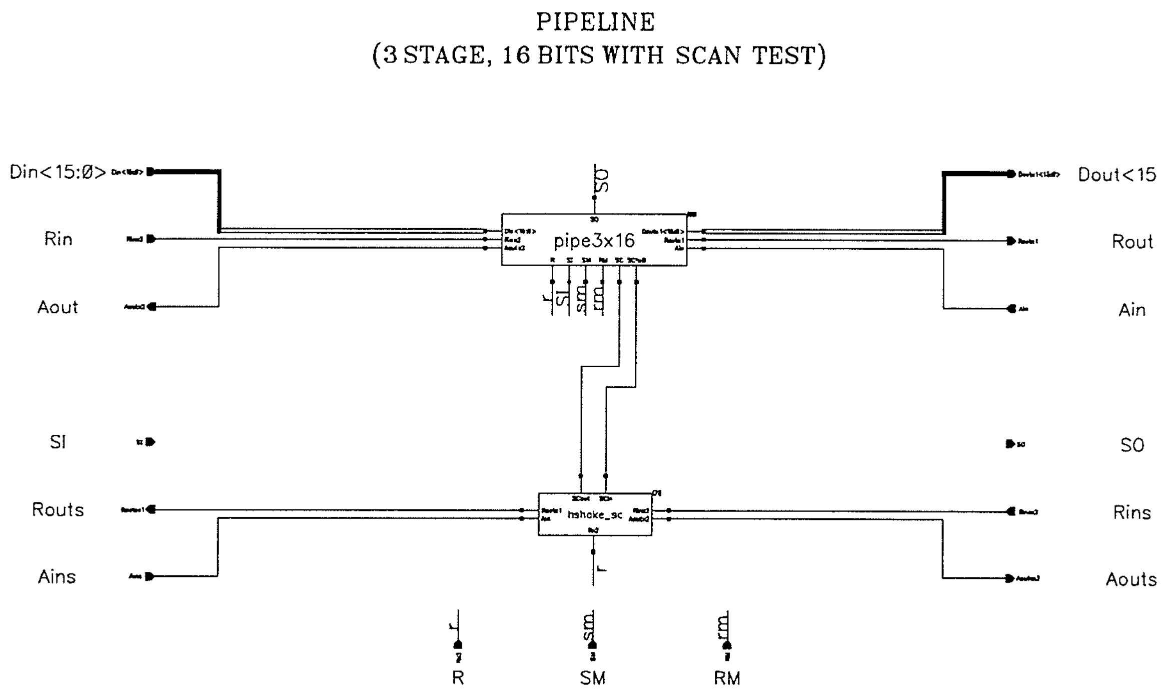


FIGURE 28 A 3 stage 16 bit pipeline with scan test. The *hshake_sc* block is a pipeline handshake block that shifts the register chain right with each request acknowledge cycle. The signalling in the scan pipeline flows to the left.

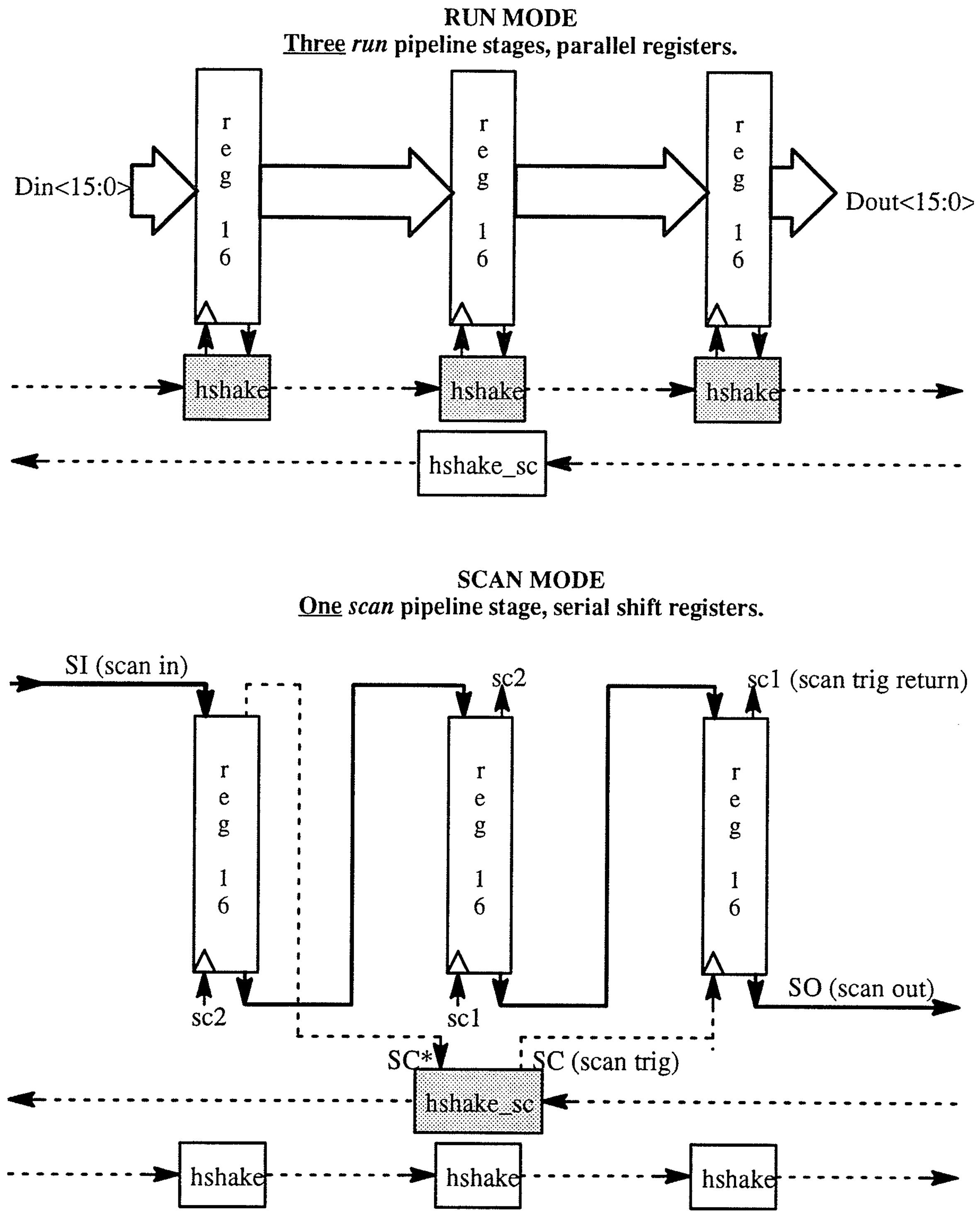


FIGURE 29 The internal registers of a 3 stage 16 bit pipeline in *run* mode and in *scan* mode. In *run* mode, data flow is controlled by signalling in the *hshake* pipeline and the registers are in a parallel mode. In *scan* mode, data flow is controlled by signalling in the *hshake_sc* pipeline and the registers are in a serial mode. The solid lines are data flow and the dotted lines handshake signal flow.

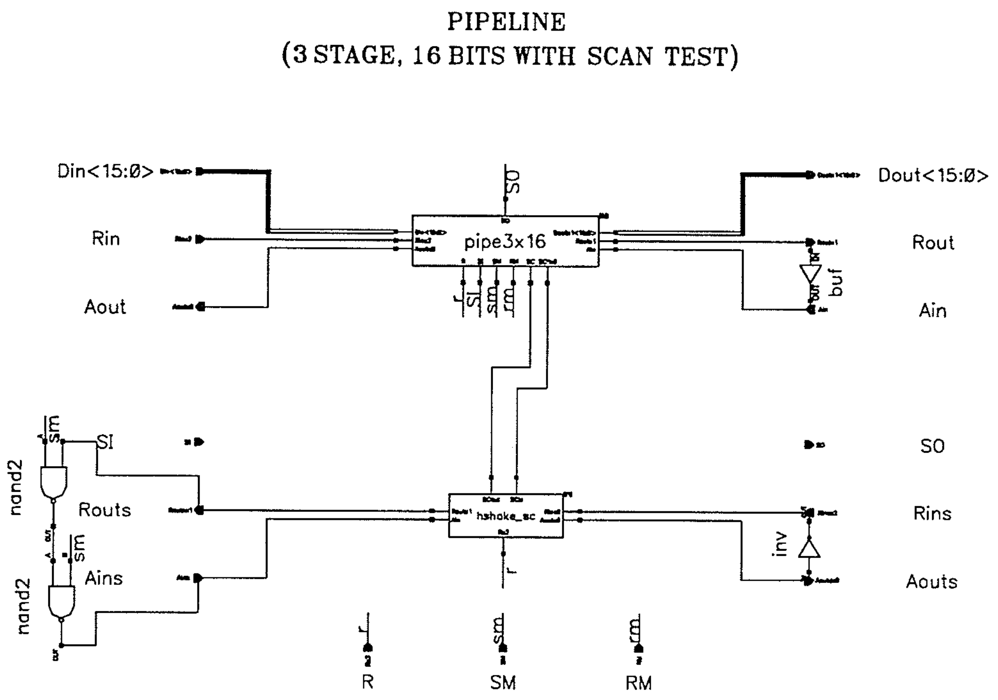
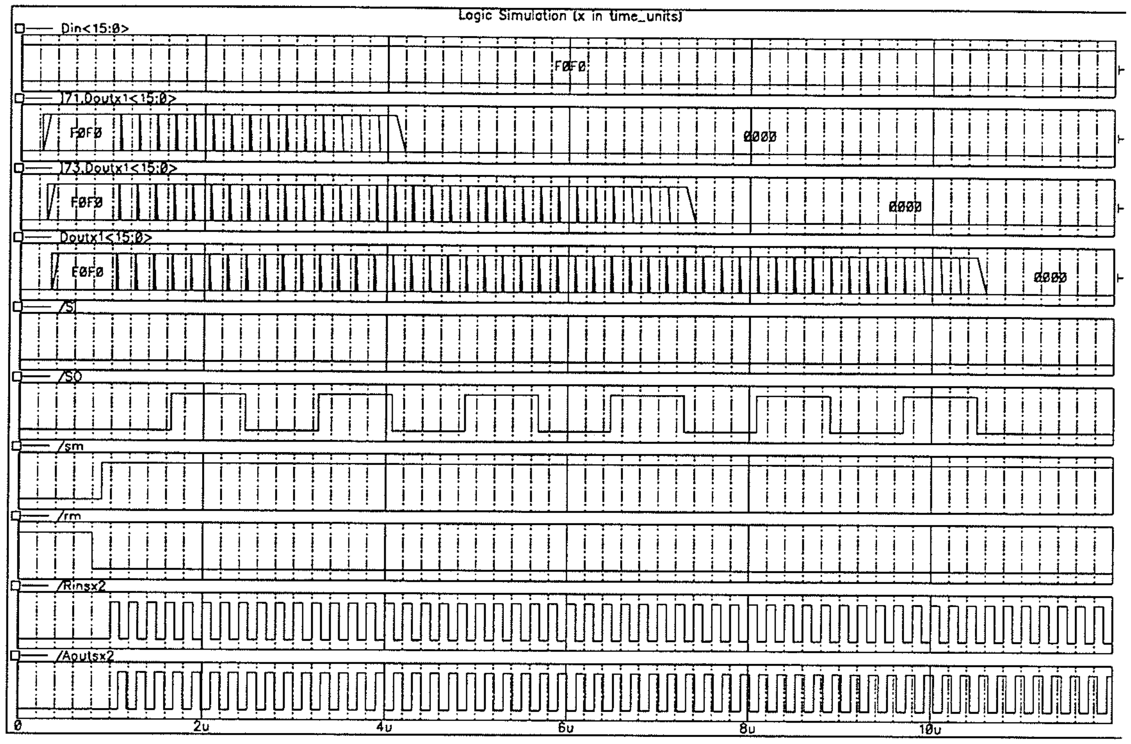


FIGURE 30 The test setup for the 3 stage by 16 bit scannable pipeline. The scan handshake element looks like it is in an infinite pipeline connected to very fast adjacent stages when scan mode is entered.

Scan test signals in Figures 28 and 30 are defined as follows:

<u>SIGNALS</u>

SI	Scan data in.
SO	Scan data out.
Routs, Ains,	
Rins, Aouts	Scan pipeline handshake signals.
R	Reset.
SM	Scan mode.
RM	Run mode.
SCout	Shift trigger signal to registers.
SCin	Shift trigger signal return.



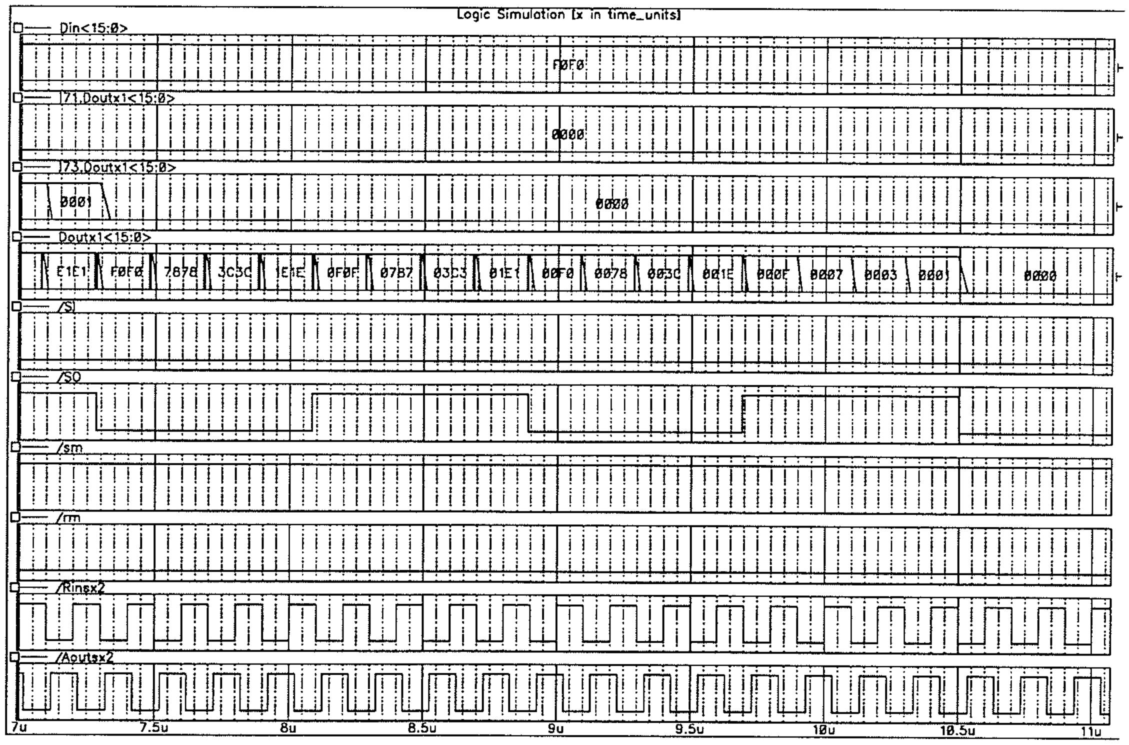


FIGURE 31 Waveforms from a simulation run performing scan test on a 3 stage 16 bit pipeline. Stages 1,2, and 3 (registers *I71.Dout, I73.Dout,* and *Dout*) are filled will F0F0 hex and then serially scanned out at *SO* with the scan input *SI* set to 0. The lower chart is a zoomed view of the upper one, showing the stage 3 register *Dout* shift its data out.

The *hshake_sc* component in Figures 28 and 30 is identical to the previous pipeline handshake element, it is just used in a slightly different way. When the registers are in scan mode, they form a shift register triggered by $SCout^+$. The *SCin* signal ensures that only after the most significant bit in the shift register chain has been shifted will the scan pipeline handshake initiate a request to the next scan stage. The operation is identical to that of *Lout* and *Lin* except that for *Lout* and *Lin* the registers would usually be in parallel with only a slight delay between $Lout^+$ and Lin^+, while there is usually a large delay between $SCout^+$ and $SCin^+$. In either case this delay is unbounded.

The scan pipeline overlay has been shown in the opposite direction, i.e. right to left, to indicate that handshake signal flow in the scan pipe should be visualized as right to left with data flow left to right. This perception is necessary because it prevents overwriting data in adjacent scan stages, and allows for trigger signal consistency when switching from run mode to scan mode. In this perceptual framework the scan pipeline must be empty when switching to scan mode. Another way to approach the problem would be to initialize the scan pipeline to full, use the complementary signal levels, and pull data off of the end of the pipeline. Data flow would then be in the same direction as scan signal flow, and the end result is the same. Note that this 'scan pipe overlay' can be used for both the asynchronous and synchronous blocks. The concept is the same as clocked scan test, except that instead of using a clock to shift the data the *SCout* handshake signal is used. The *SCin* return ensures integrity of the scan shift.

A typical scannable positive edge triggered D flip flop from our design is in Figure 32. The sequence of events to enter and exit scan mode must be carefully orchestrated to prevent unwanted positive edge transitions on the D flip flop trigger inputs. The following describes signals in the D flip flop schematic:

<u>SIGNALS</u>

D	Data in
SI	Scan data in.
Q	Data out. Connects to *SI* out of the next stage.
CK	Asynchronous or synchronous trigger.
SC	Scan trigger.
RM	*Run* mode.
SM	*Scan* mode.
R	Reset.
S	Set.

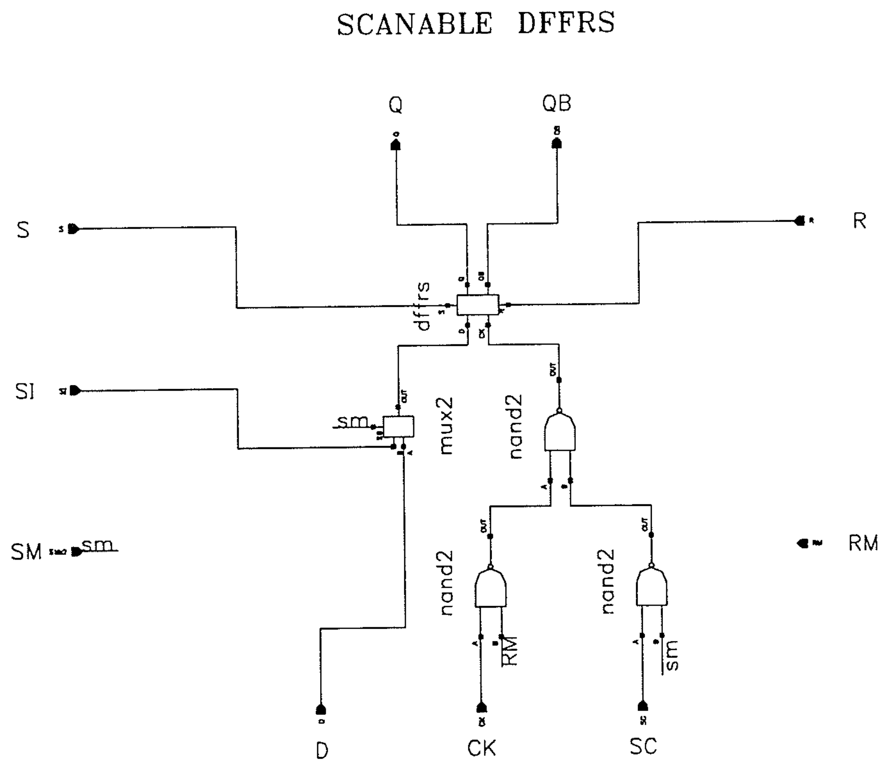


FIGURE 32 A asynchronously scannable D flip flop.

The sequence of events to enter and exit scan mode is as follows:

1. Before entering *scan* mode, the clock signal can be either in a high or low state, but it must be in a steady state. this means an asynchronous pipeline stage can be scanned in a full or empty state, as long as the pipeline has been allowed to settle. For computation blocks, the clock must be stopped or the computation allowed to complete. The scan pipe must be initially empty. (*SCout* = 0)
2. Enter *scan* mode as follows to prevent transient spikes in the D flip flop trigger:

RM	SM
1	0
1	1
0	1

TABLE 2 Control signal sequence to enter scan mode.

3. Scan.
4. Finish scan with the scan pipe full.
5. Switch back to *run* mode as follows:

RM	SM
0	1
1	1
1	0

TABLE 3 Control signal sequence to exit scan mode.

6. Now empty the scan pipe. This will have no effect on the registers, but will prepare the scan pipe for the next scan mode. This can be done concurrently with normal computations in the *run* mode.

Following this sequence of events exactly will prevent accidental latching of data. During scan mode, all asynchronous sets or resets of latches, such as those in Muller C elements, are disabled just as in fully synchronous scan test systems. Figure 33 shows how the scannable D flop flop, *dffsc,* could be used in a 4 bit register. An alternative would be to use only one set of control circuitry (the *nand* and *mux2* cells) for the 4 D flip flops.

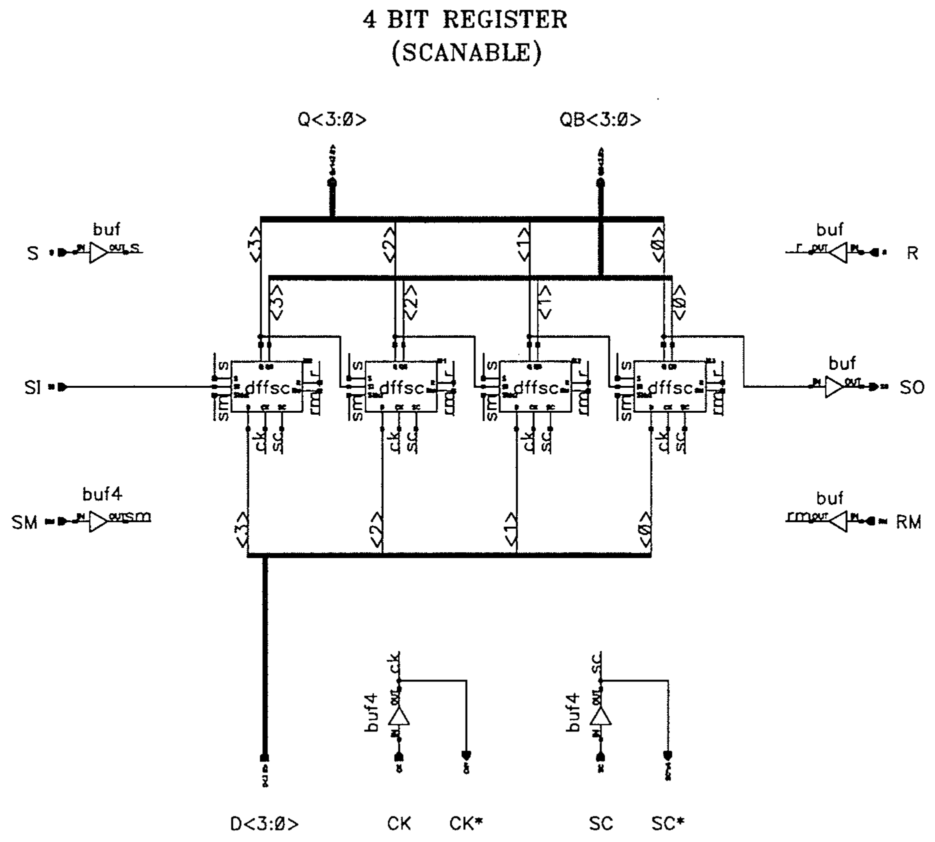


FIGURE 33 A scannable 4 bit register.

The final processor array would have one (or more if desired) scan handshake blocks for each processing element, as in Figure 46. The array would have one or more scan pipe overlays to test for faulty processors so that other fault tolerant techniques could be employed, such as switching in a spare PE for a bad PE.

3.5 ASYNCHRONOUS CLOCK

The title of this component seems to be a contradiction in terms, but its function is relatively simple. The circuit, modified from [7], produces a selectable number of clock pulses upon receiving a request and then puts out a request. It would be used to provide clock pulses for the clocked computation blocks between asynchronous handshaking blocks. The modifications we applied allowed the number of pulses to be selectable. Modules with ranges from 1 to 15 and 1 to 255 pulses were built. An *Rout*$^+$ transition will occur only after the final clock period is completely over. Glitch prevention was added to prevent an *Rout* glitch when adjusting the number of clock cycles during the inactive phase. Figure 34 shows the asynchronous clock circuit and Figure 35 shows how it would be used in a 3 by 16 bit pipeline.

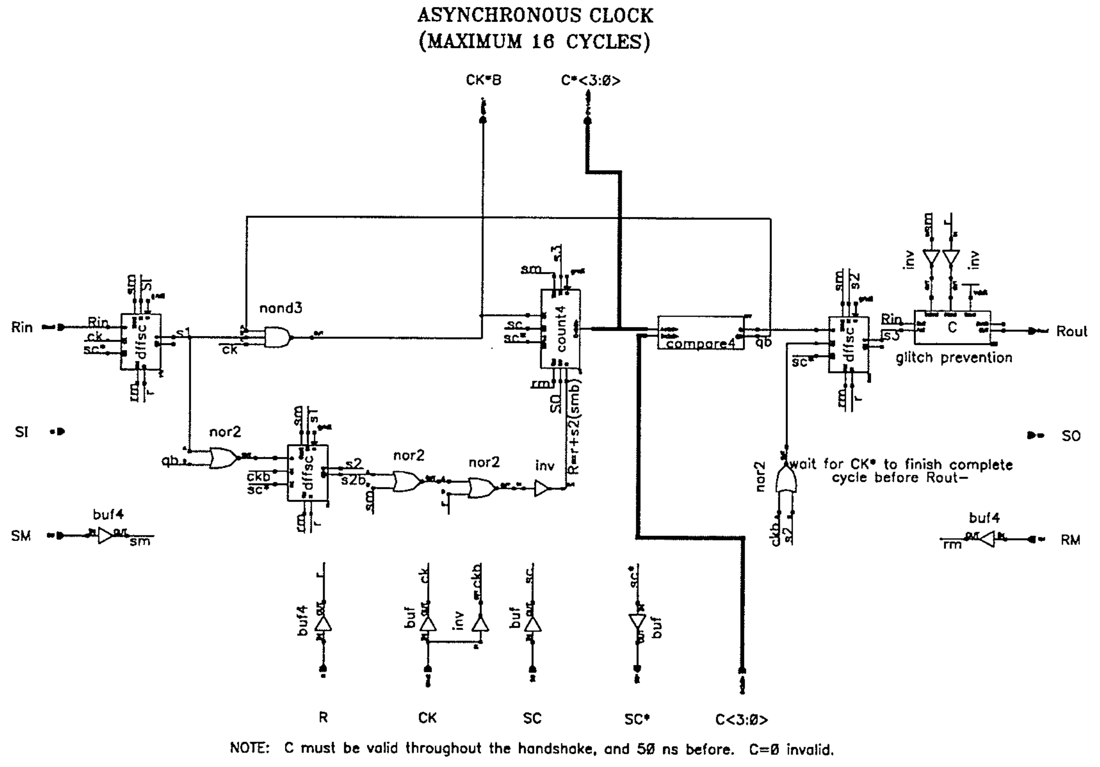

ASYNCHRONOUS CLOCK
(MAXIMUM 16 CYCLES)
CK*B
C*<3:Ø>
Rin
SI
SM
nand3
nor2
count4
compare4
glitch prevention
Rout
SO
RM
wait for CK* to finish complete cycle before Rout-
buf4
R
CK
SC
SC*
C<3:Ø>
NOTE: C must be valid throughout the handshake, and 5Ø ns before. C=Ø invalid.

FIGURE 34 An asynchronous clock circuit. The number of clock pulses is selectable from 1 to 15.

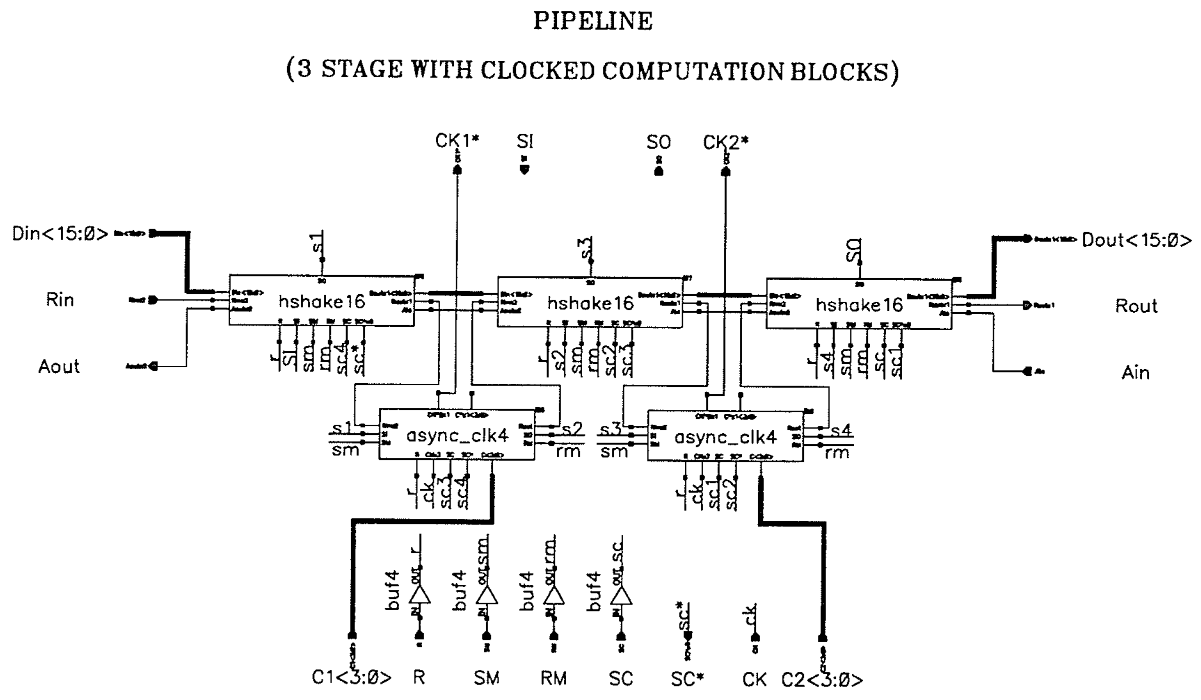


FIGURE 35 A pipeline with clocked computation blocks using the asynchronous clock units.

3.6 I/O ROUTERS

The input and output routers are the heart of the processing element's design. Most of the computational power of the PE for the target algorithms is contained not in the ALU but in the routers. The I/O routers consist of 3 macro blocks each as shown in Figures 36. The routers each have a controller to calculate the next route, handshaking logic to implement the asynchronous 4–phase signalling protocol, and multiplex or demultiplex logic to multiplex or demultiplex the data.

The routers were designed to make the PE's *non–algorithmic* and *escapement* machines, as opposed to *algorithmic* and *unsynchronous*. Both of these choices make the PE's

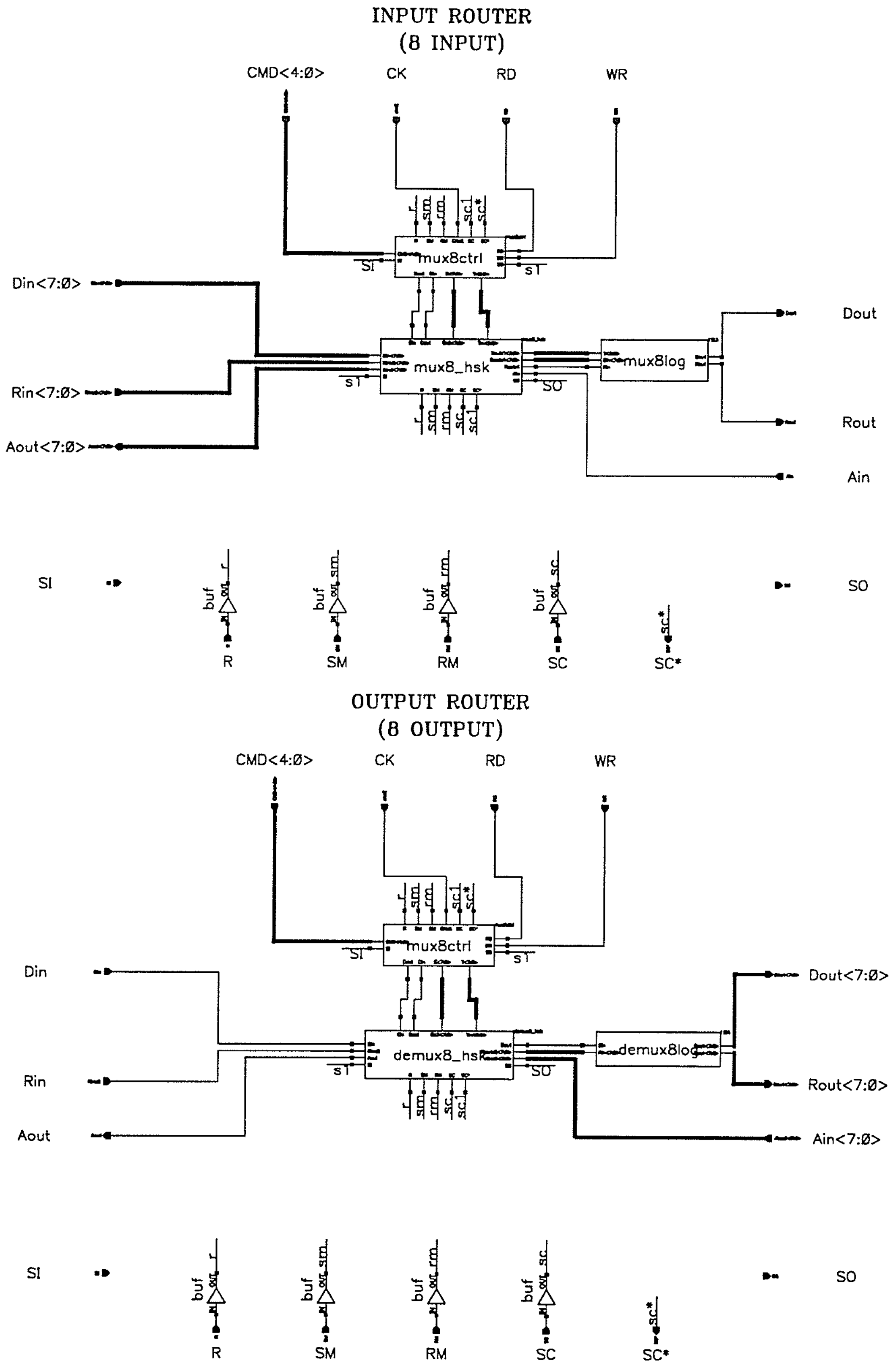


FIGURE 36 The input and output routers for a processing element in an 8 connected mesh. The routers have self–timed asynchronous communications and use locally synchronous controllers.

flexible and powerful but have hardware consequences, and so we will discuss them before describing the hardware.

By *non–algorithmic* we simply mean that the incoming and outgoing routes are stored in a pre–calculated look–up tables rather than calculated by some deterministic means based on the incoming data. Lattard and Mazare [19][20], for example, pass the ray projection angle and input point with the data and use a deterministic algorithm within a transputer to calculate the next route. Iterative register logic techniques such as the Cordic [35][36] implementation are used to calculate the *sin* and *cos* functions with a minimal of area. The advantage of the Cordic implementation is in silicon area at the expense of speed. The advantage of a look–up table technique such as ours is that it is very fast and can be used with many different algorithms, especially those which repeat the routing pattern over and over again so that only a small routing table is needed. Image reconstruction, pattern recognition, and neural network problems fall into this category. The disadvantages is in increased area usage and in the time it would take to load the routing tables for a large WSI array. Once loaded, the array would be expected to run with the same table for a considerable amount of time. A few simple calculations show that our approach is not unreasonable for a wafer scale implementation. As designed, our PE contains a 256x4 bit input routing table and a 256x4bit output routing table, which would be sufficient to calculate the Hough transform or perform image reconstruction on a 256x256 image. This would amount to 16 megabytes of storage with one processing element per pixel, which is far from an unreasonable amount for a wafer when already micro–chips alone have approached this number. A more area efficient means of implementing routing could certainly be found, depending on how application specific and how fast you want the array to be. The method of route calculation in our design could be changed to some standard deterministic means with little difficulty since the controller is a synchronous component.

The other important feature of the routing structure is that the PE is an *escapement* machine[4]. An escapement machine knows where the next request is coming from, but not when. Since the *unsynchronous* machine does not know where or when the next event is coming, it can have trouble with metastability[31]. For an unsynchronous machine, additional circuitry is required to perform synchronization arbitration and queuing. The disadvantage of our escapement machine is that both input and output route calculation must be performed.

Multiplex and Demultiplex Controller

The mux/demux controller performs route calculation and controls the handshaking blocks for the input and output router. A routing scheme for the PE is shown in Figure 37, and the mux/demux controller itself in Figure 38. It is a clocked component with 5 valid commands, sufficient to perform image reconstruction, the Hough transform, or pattern recognition problems that use line integrals. The three major groups of circuits shown in the figure perform clocking, address calculation, and route storage functions. The asynchronous clock unit interfaces with the handshake circuit and provides clock pulses to drive the controller. If dictated to do so by the current command, the route calculation circuitry increments the RAM address once every 16 requests, which make the routers 16 bit serial. The routes are stored in a RAM table of variable size, up to 256 (default) in our design. The table repeats itself over and over, which is very suitable for the image reconstruction and pattern recognition problems we wish to solve.

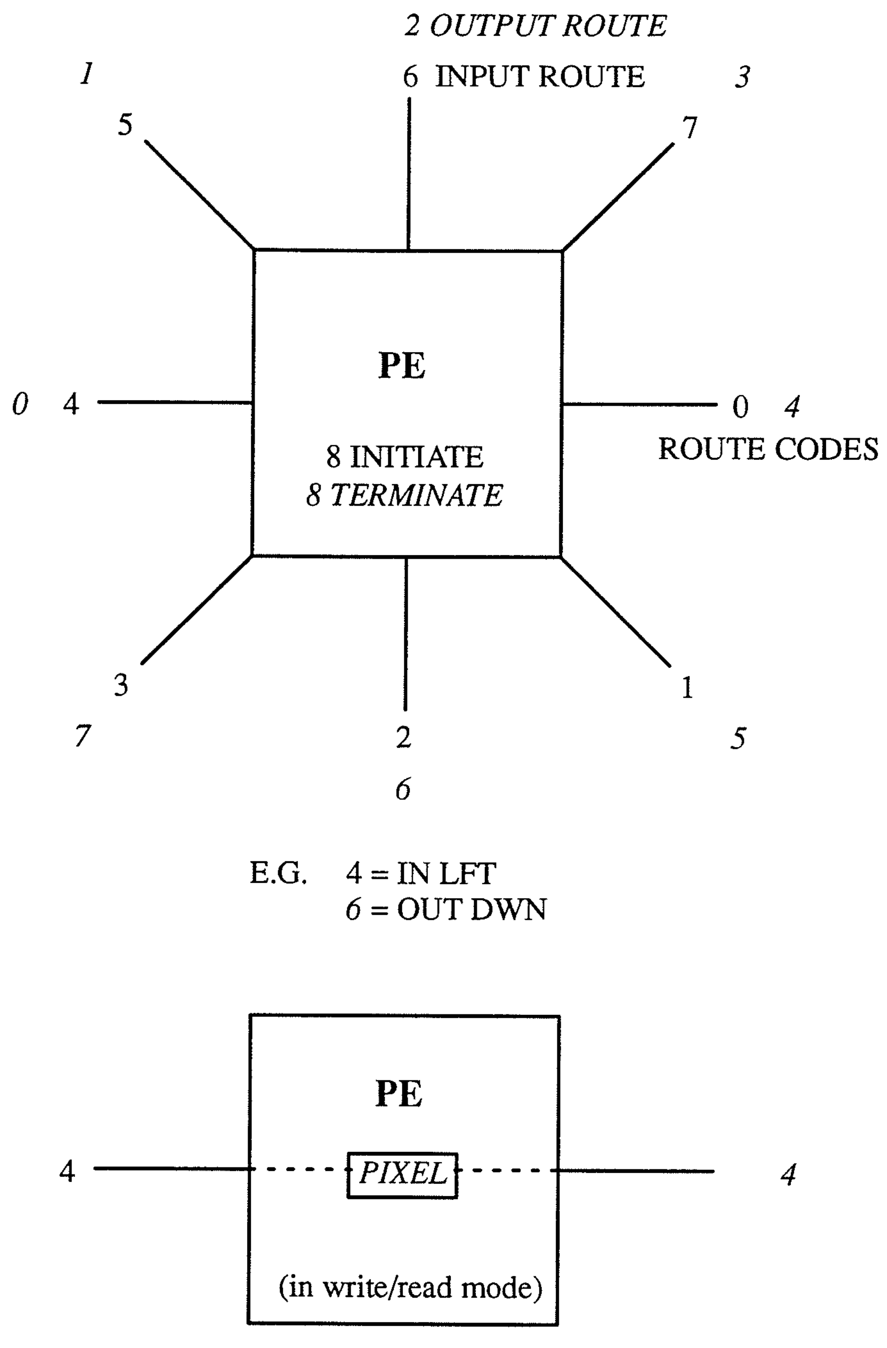


FIGURE 37 Processing element routing scheme. There are 9 route codes including those for initiate and terminate.

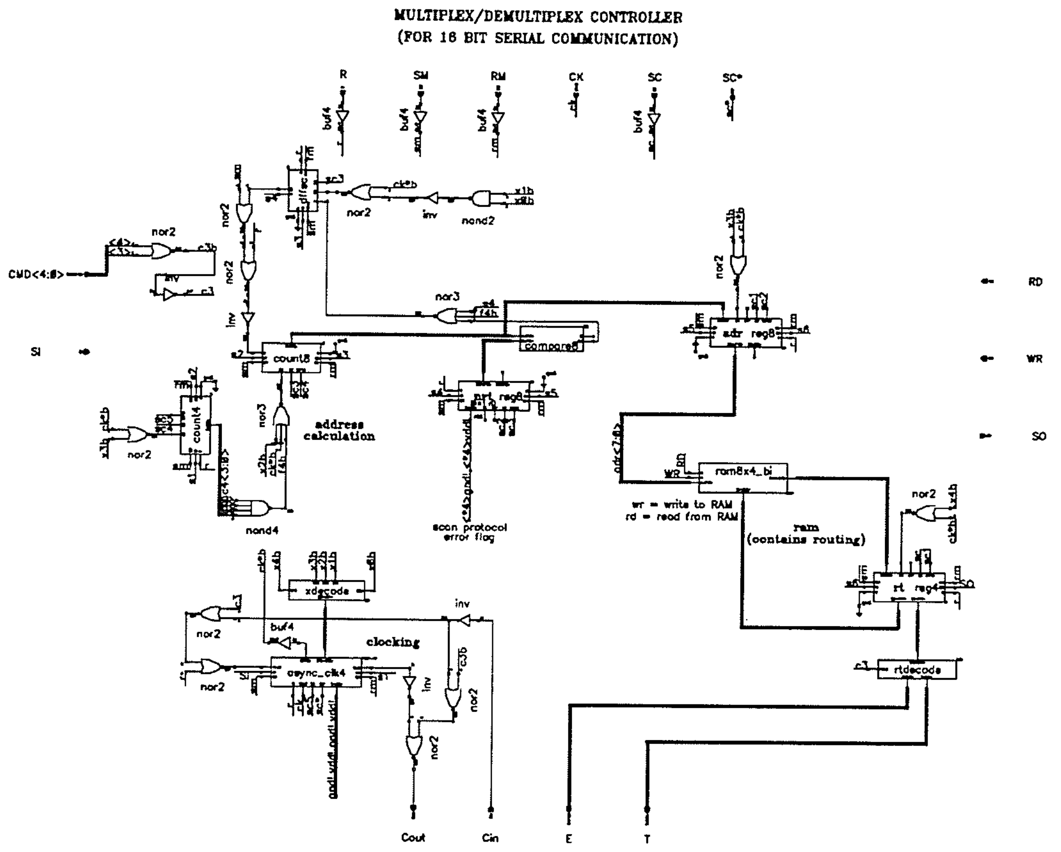


FIGURE 38 The multiplex and demultiplex controller.

The following describes the commands, registers, and important signals in the controller:

COMMANDS
(Decoded, e.g. 00001 = *Project*).

CMD<4>	*Clear* command. Routing set as in *CMD<3>*.
CMD<3>	*Write/read.* Set the route to IN LEFT and OUT RIGHT. This makes a linear array out of each row for reading and writing the image data.
CMD<2>	*Update.* Not applicable to the controller, since the multiplexer or demultiplexer will not receive a request in this command mode.
CMD<1>	*Backproject.* Step through routing table.
CMD<0>	*Project.* Step through routing table.

REGISTERS

nrt	Tells the controller how many routes are in the table.
addr	The address of the current route.
rt	The route code, 0 to 8 hex.

SIGNALS

CMD	Decoded command, 5 bits.
RD	Ram read (from ram).
WR	Ram write (to ram).
Cin	Request out, controller completion signal.
Cout	Acknowledge in, from handshake circuit.
E	Mux/demux enable byte.
T	Mux/demux select nibble.

The asynchronous clock unit is used to generate the clock pulses, generating the first set of pulses immediately after reset and a new set of pulses after receiving the Cin^{-} transition from the handshake circuitry. The controller looks like a clocked computation block in a pipeline flowing from top to bottom, with *Cout* being its completion signal. This 'controller pipeline' was made to have a continuous request, beginning to calculate the next route as soon as the handshake block has received the current route. It must have a fixed number of clock cycles(5), since it is never latent and thus there is no opportunity to change the clock cycles for the next instruction. This will not affect performance since only the instructions *project* and *backproject* use the clock and each require the same number of clock cycles. Referring to Figure 38, a typical instruction sequence for the *project* mode would be as follows:

1. Toggle the *count8* reset latch to 1 if the number of routes *nrt* equals *count8*.
2. Toggle the *count8* reset latch and increment the 8 bit counter if the 4 bit counter is done.
3. Increment the 4 bit counter.
4. Load the address from the 8 bit counter.
5. Load the route, and put out a request.

The *backproject* command would execute identically. The *write/read* and *clear* commands do not step through the routing table since the asynchronous clock is bypassed in these modes. The *update* command is a *don't care* state for the mux/demux controller since the controller will never receive handshaking signals from the handshaking blocks in this mode.

The RAM for the mux/demux controller must be loaded via the scan chain. The RAM used in our design was generated using the University of British Columbia's *Module Maker* tool[10], which is integrated into the *Cadence* design framework. The tool generated a static RAM of the desired size and used transistor–level *hspice* simulation on a delay model to provide gate–level delay parameters for the *SILOS* representation. Because of minor problems with the module maker tool, the *hspice* simulations had to be re–run manually and the delay parameters extracted from the resulting waveform. The delay parameters were scaled up before they were entered into the *SILOS* model of the ram to provide a safety margin, since the accuracy of the transistor–level delay model automatically generated by the tool is limited.

Multiplex and Demultiplex Handshake

The multiplex handshake circuit is shown in Figure 39 and the demultiplex handshake circuit in Figure 40. We will describe only the mux handshake in detail, since the demux is nearly a mirror image.

The multiplexer handshake has 4 stages, 3 to select one of the 8 inputs and one to allow for an *initiate* command. A 4 input C elements collects *Cout* events from each of the stages and acknowledges the controller when all the stages are done. The 4 input C element *event and* forces the stages to operate as a unit; only after *Cout*$^+$ or *Cout–* has occurred for all the stages will it occur for the entire unit. The stages do not operate concurrently. Each of the 2 input multiplexer elements is controlled by one bit of the enable byte *E<7:0>* and

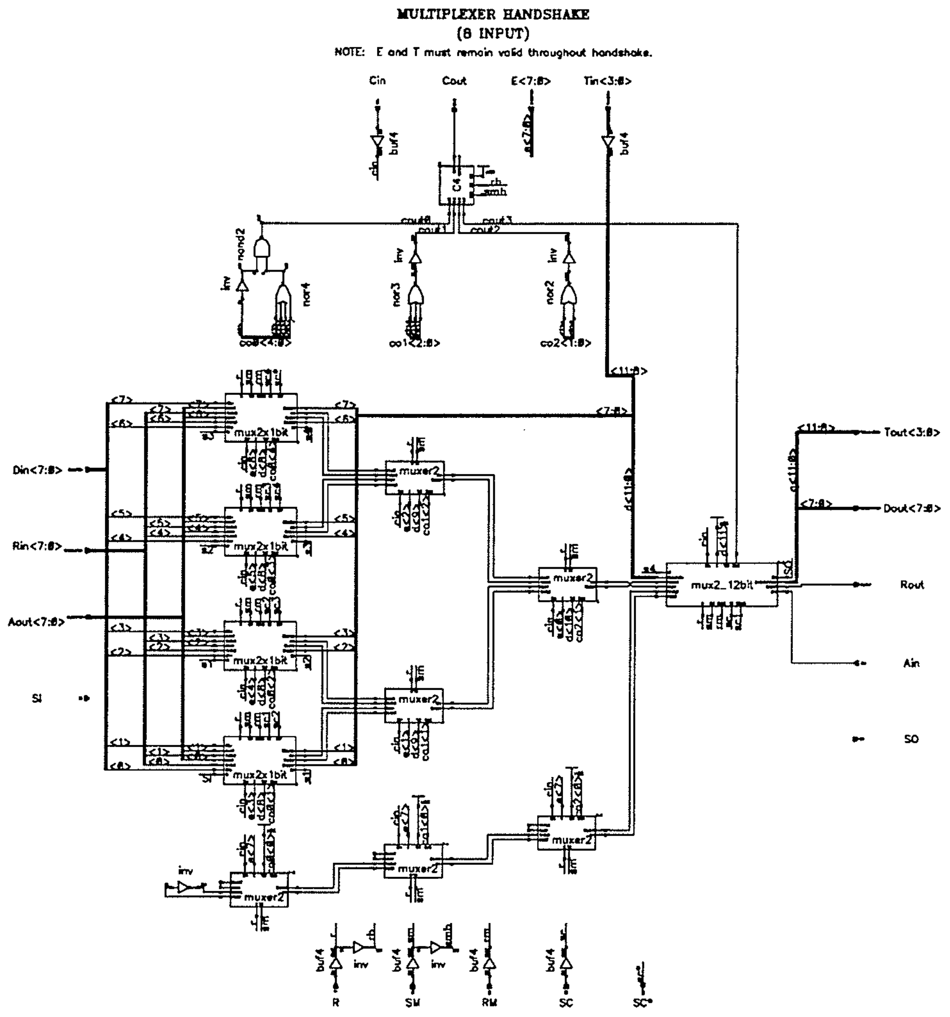


FIGURE 39 The multiplex handshake circuit. It has four stages and one interface to the controller.

E<7:0>	T<3:0>	Selected Input
≥80	≥8	8 (initiate or terminate)
45	7	7
45	6	6
25	5	5
25	4	4
13	3	3
13	2	2
0B	1	1
0B	0	0

TABLE 4 Enable and select patterns for each route code. Values are in hexadecimal.

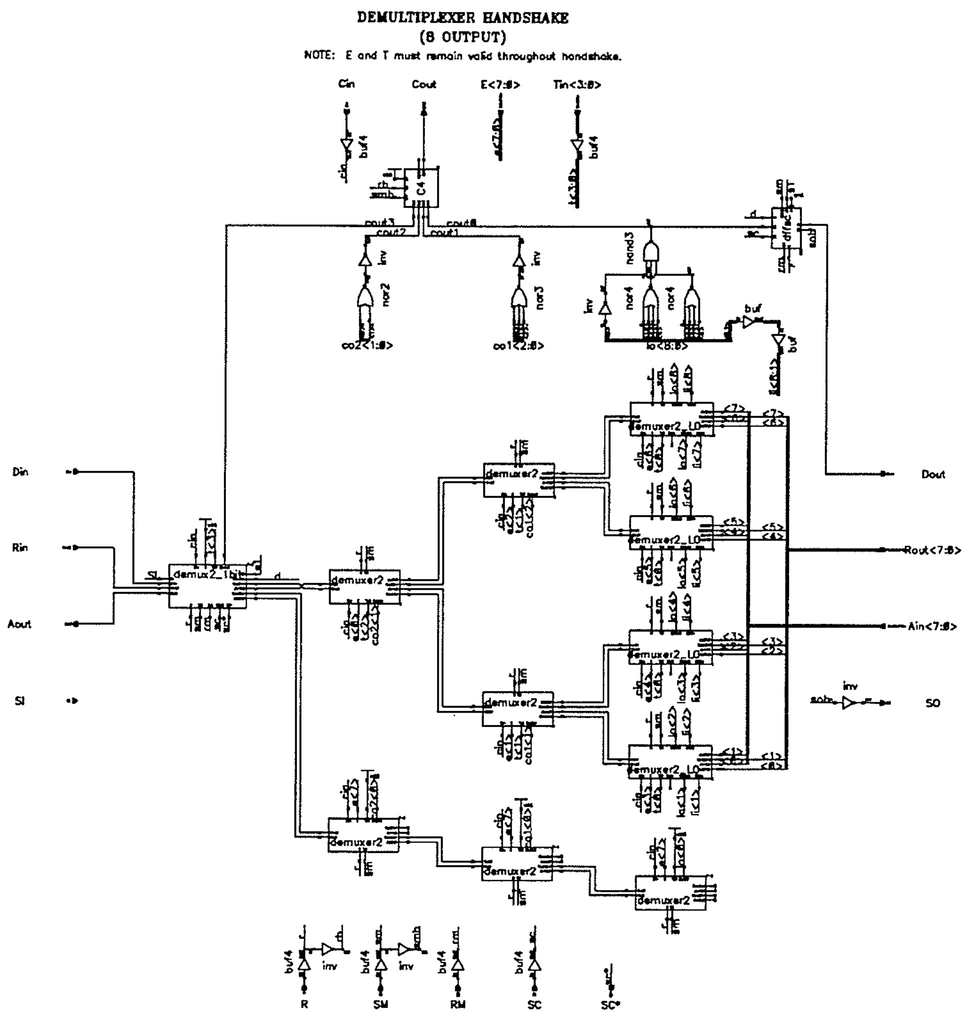


FIGURE 40 The demultiplex handshake circuit.

a one bit of the select nibble *T<3:0>*, both of which must stay valid throughout the handshake. The enable bit enables the controller handshake for the particular 2 input mux and the select bit selects either input. For each route code, the multiplexer handshake circuit requires a specific enable and select nibble to control each 2 input multiplexer appropriately, as in Table 4. The demultiplexer uses an identical pattern.

Data is latched into the multiplexers in the first stage before a controller handshake. Thus although there is no queue perse, there is a high degree of concurrency since incoming data from any port can be accepted before being serviced by the controller. This would be even better if the routers were operating with parallel rather than serial data. The multiplexer handshake circuit passes the select nibble *Tout<3:0>* and the data *Dout<7:0>* to the multiplexer logic. It is not necessary to pass the select code for the demultiplexer. The registers in the last stage for *Tout* and *Dout* are not required, since only when Ain^+ has occurred (meaning the next stage has latched the data) can the controller start another handshake, but were left in for test purposes.

Multiplex and Demultiplex Logic

The multiplex and demultiplex logic performs the actual mux and demux functions on the data. These logic blocks used delay rather than an asynchronous clock for synchronization; other than that they are standard gate implementations. The mux and demux logic could be integrated with the ALU to avoid using delay to provide the completion signal, but this would mean more registers. The mutliplex and demultiplex circuit can be found in the *Technical Reference Addendum* for this thesis.

Simulation

Simulations for the input and output router are shown in Figures 41 and 42. The ram was initialized to contain each of the 9 route codes, the first being the initiate or terminate route. An alternating pattern was placed on the input. Examining the simulations, you can see 16 handshakes for each route, showing the 16 bit serial operation. For the input router *Dout* alternates as each successive complemented input is selected, and for the output router *Dout* follows the input pattern and a request in placed on the selected output. Simulations were also performed loading the RAM manually via the scan chain.

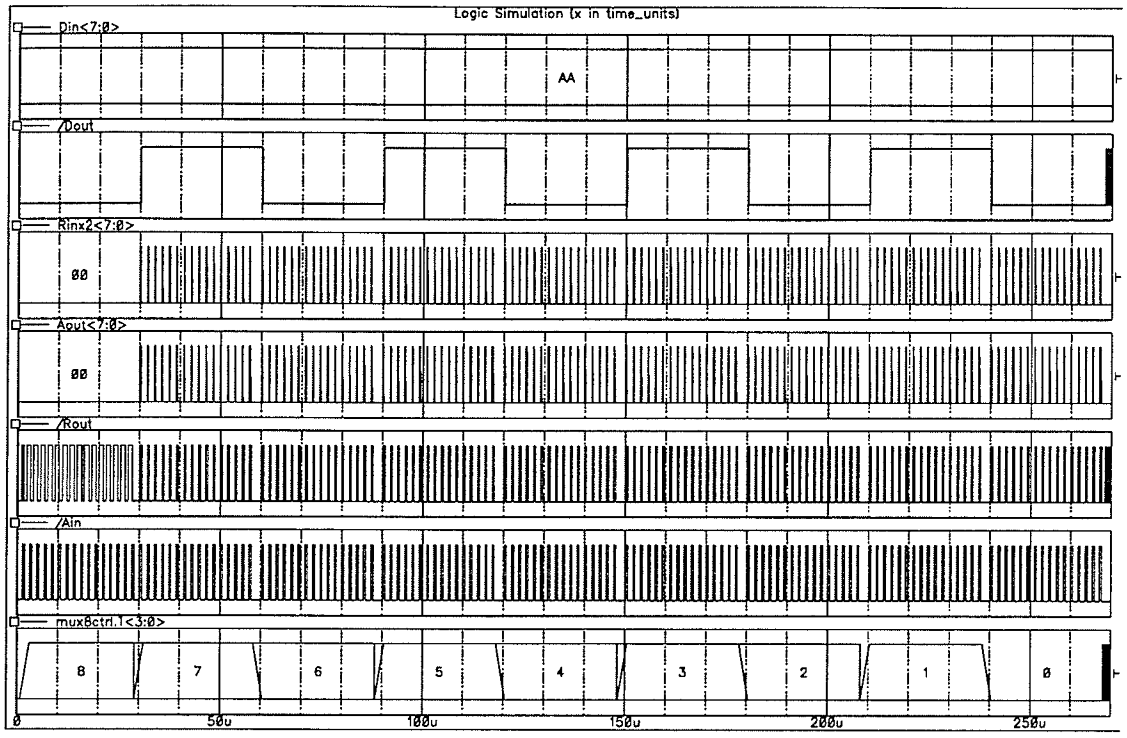


FIGURE 41 Input router simulation. There are 8 serial inputs and 1 serial output. The input data *Din<7:0>* is fixed at 10101010 or AA in hexadecimal, i.e. *Din<7>* = 1 for the duration of the simulation. Each of the 8 inputs is chosen according to the routing table and passed to *Dout*. The route codes from the table are shown as *mux8ctrl.T<3:0>*. There are 16 request/acknowledge cycles before a new route code is loaded from the table since the operation of the routers is 16 bit serial. Route code 8 is a is an initiate command and requires no request.

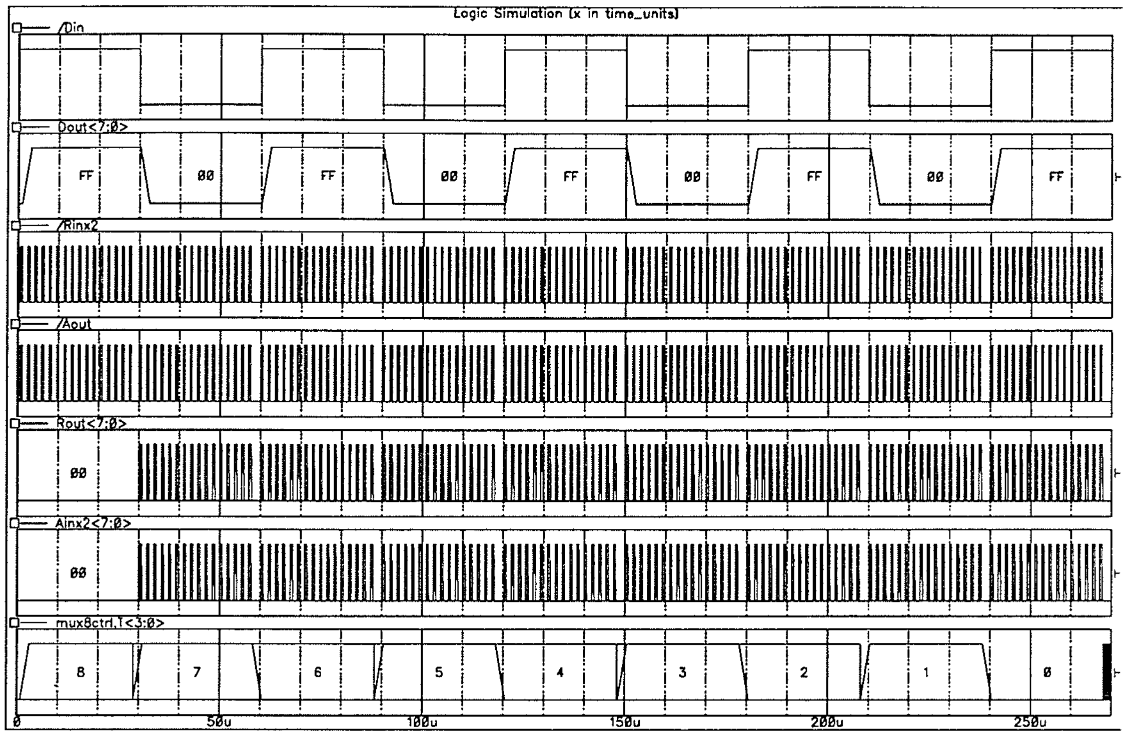
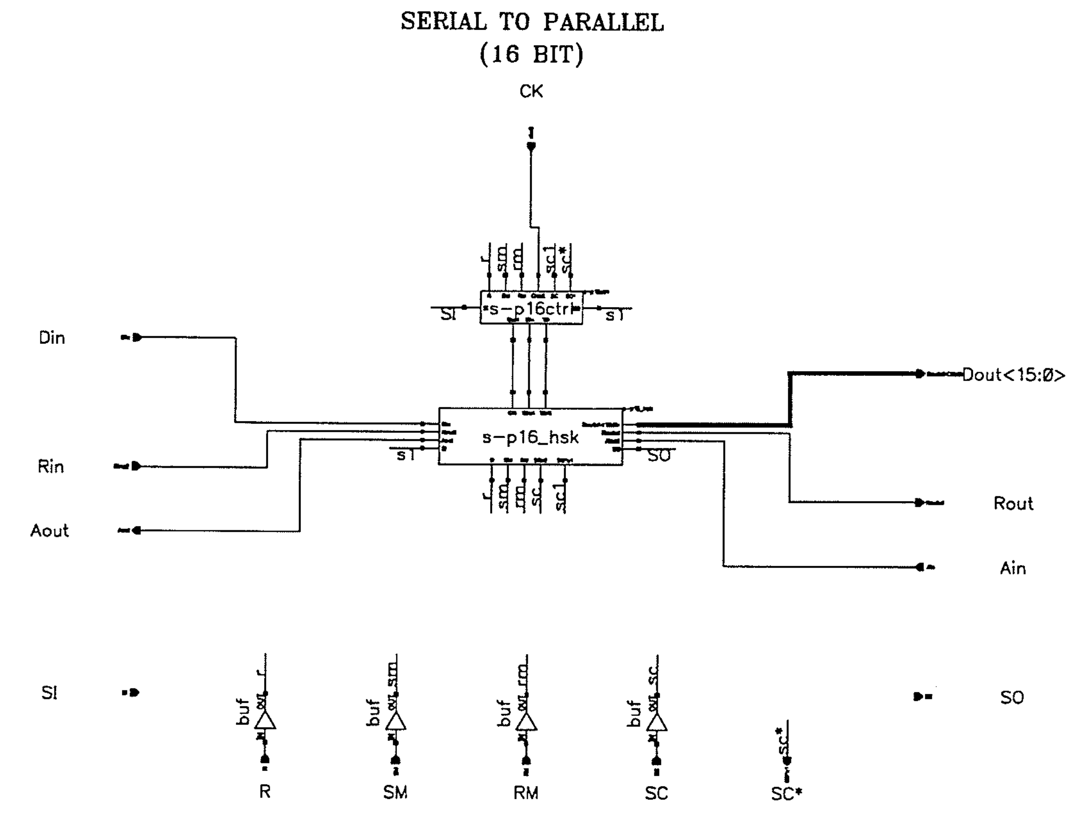


FIGURE 42 Output router simulation, dual of the input router. There is 1 serial input and 8 serial outputs. The input data *Din* is fixed at either 1 or 0 throughout a routing cycle. The input is passed to one of the outputs *Dout<7:0>* according to the routing table. Again the route codes are shown as *mux8ctrl.T<3:0>* and again there are 16 request/acknowledge cycles for each route code. For the output router, route code 8 is a terminate command and thus does not put out a request.

3.7 SERIAL TO PARALLEL/PARALLEL TO SERIAL

The parallel to serial/serial to parallel converters do not require much explanation, since they are just less complicated versions of the routers that contain shift registers instead of parallel registers. The serial to parallel circuit puts out one request for each 16 input requests and requires one 2 input demux in the handshake circuitry to do this. The parallel to serial circuit puts out 16 requests for each single input request and requires one 2 input multiplexer in the handshake circuitry. Data is shifted by the handshake signals of the mux or demux in a manner similar to scan mode operation. Detailed circuit schematics are in the *Technical Reference Document.* Parallel to serial conversion had to be done because of pin–out constraints.



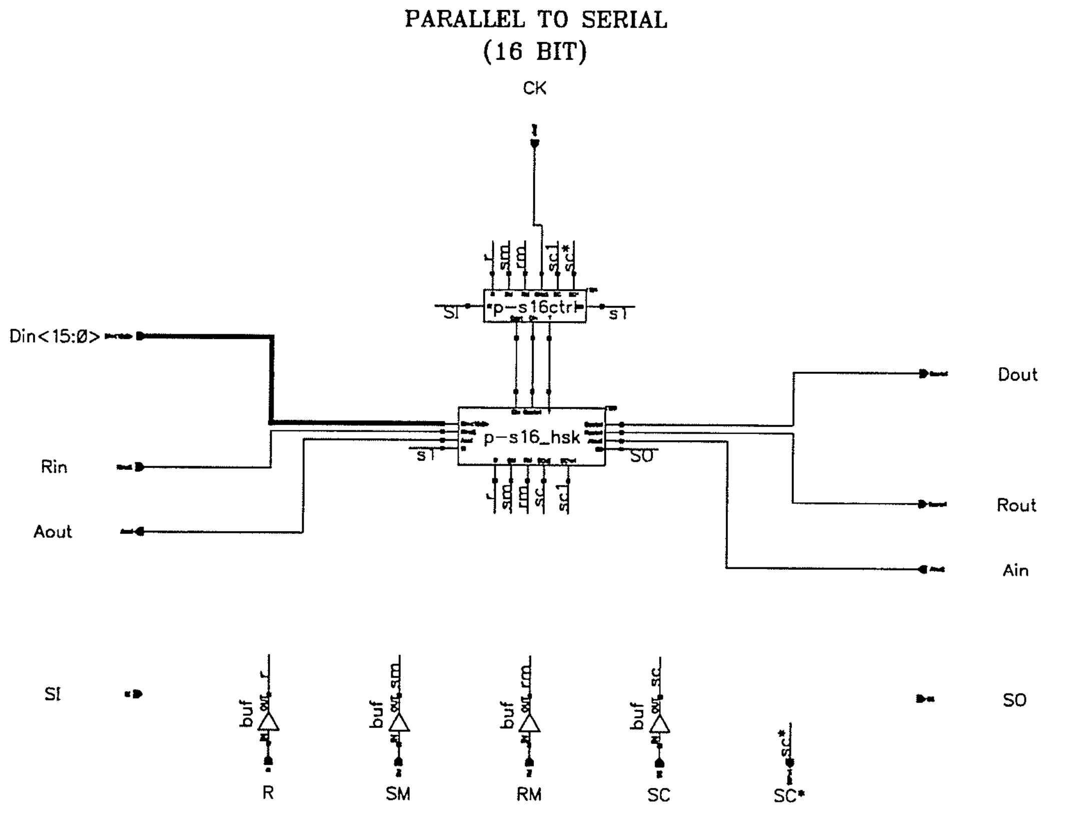


FIGURE 43 The serial to parallel and parallel to serial converter.

3.8 ALU

The ALU in Figure 44 was was designed with a 16 bit wide bus, large enough to pass ray sums for a 256x256 pixel black and white image with a 256 level gray scale. Enough commands were implemented to allow a PE array to calculate the Hough transform. Its major components are an 8 bit register to store the pixel, an adder, multiplexer logic, and the asynchronous clock element. Most of the signals are self explanatory, except for *OF* which indicates adder overflow. The decoded commands are the same as for the input and output router, excluding the *update* and *backproject* commands which were not implemented. It would be a simple matter to add these commands.

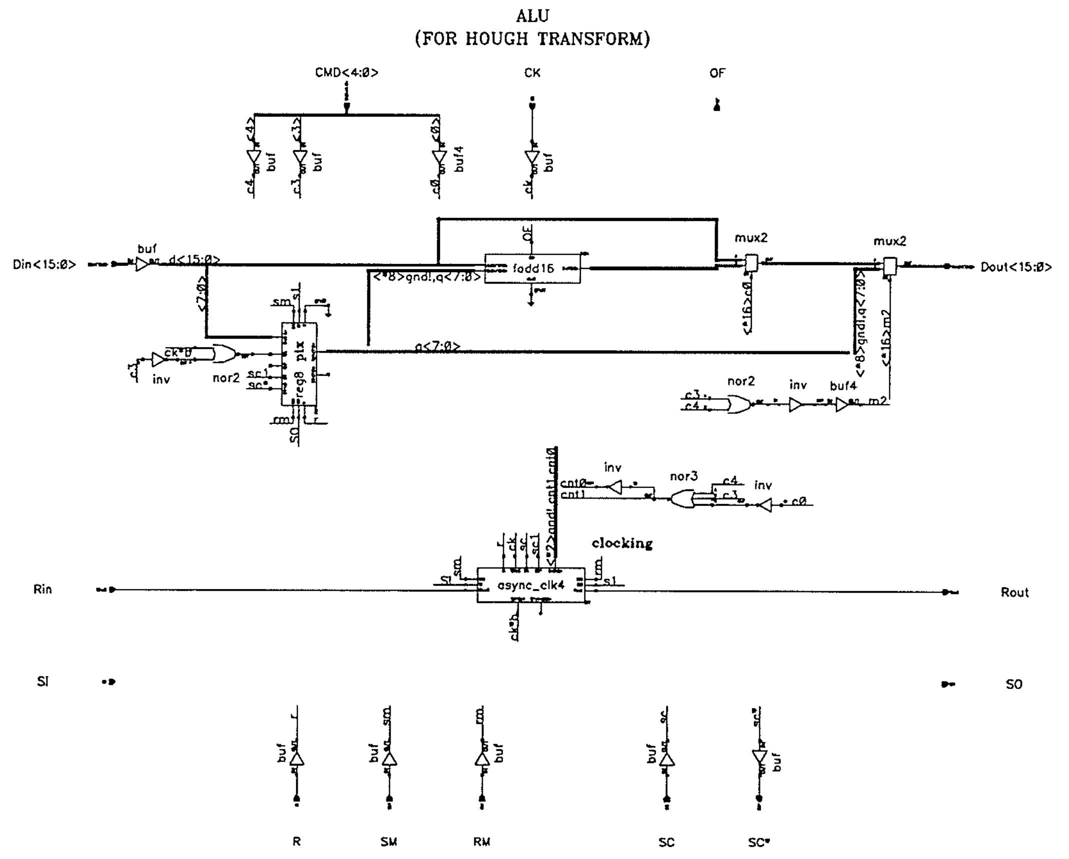


FIGURE 44 The ALU. It is a synchronous computation block with enough commands to allow an array of PE's to compute the Hough transform.

ALU COMMANDS
(Decoded, e.g. 00001 = *Project*).

CMD<4>	*Clear* command. Places the pixel on the data out bus..
CMD<3>	*Write/read.* The *Din* bus is loaded into the pixel register and placed on the *Dout* bus.
CMD<2>	*Update.* N/A. Not implemented, defaults to *clear.*
CMD<1>	*Backproject.* N/A. Not implemented, defaults to *clear.*
CMD<0>	*Project.* The pixel is added to the incoming data and the total is placed on the output bus, to calculate the ray sum.

REGISTERS

pix	The 8 bit pixel register.

SIGNALS

OF	Adder overflow.

Reading an image from the array must be done in conjunction with the *clear* command in a specific sequence to ensure the image data is not overwritten by previous pipeline stages. The *clear* command does not perform a computation, but is used to copy pixel data out of the ALU and into the next pipeline stage. The name of the commands is somewhat of a misnomer since the command actually has a purpose. Reading and writing image data to the array would be performed as follows; recall that in the *write/read* command mode, rows of PE's are arranged as linear arrays left to right by the input and output routers.

READ (Read an image from the array)

1. Set the command to *nothing* and fill the row pipelines in the array. This will move the pixel into the next pipeline register.
2. Switch to *write/read* mode and pull the pixel data off of the pipeline.

WRITE (Write an image to the array)

1. Start with the pipeline empty, and give no acknowledgements. Finish with the pipeline full.
2. Switch to another mode and continue.

This method of reading and writing image data is much faster that using a scan chain. It would be practical to process many different images in a short time after the routing tables were loaded.

The ALU we implemented was very simple; most of the computation power of the processing element is in the I/O routers. This particular implementation of the ALU showed use of a different number of instruction cycles for different commands, an important demonstration for the operation of the asynchronous clock unit. For fabrication, the serial to parallel converter, ALU, and parallel to serial converters were placed on one chip.

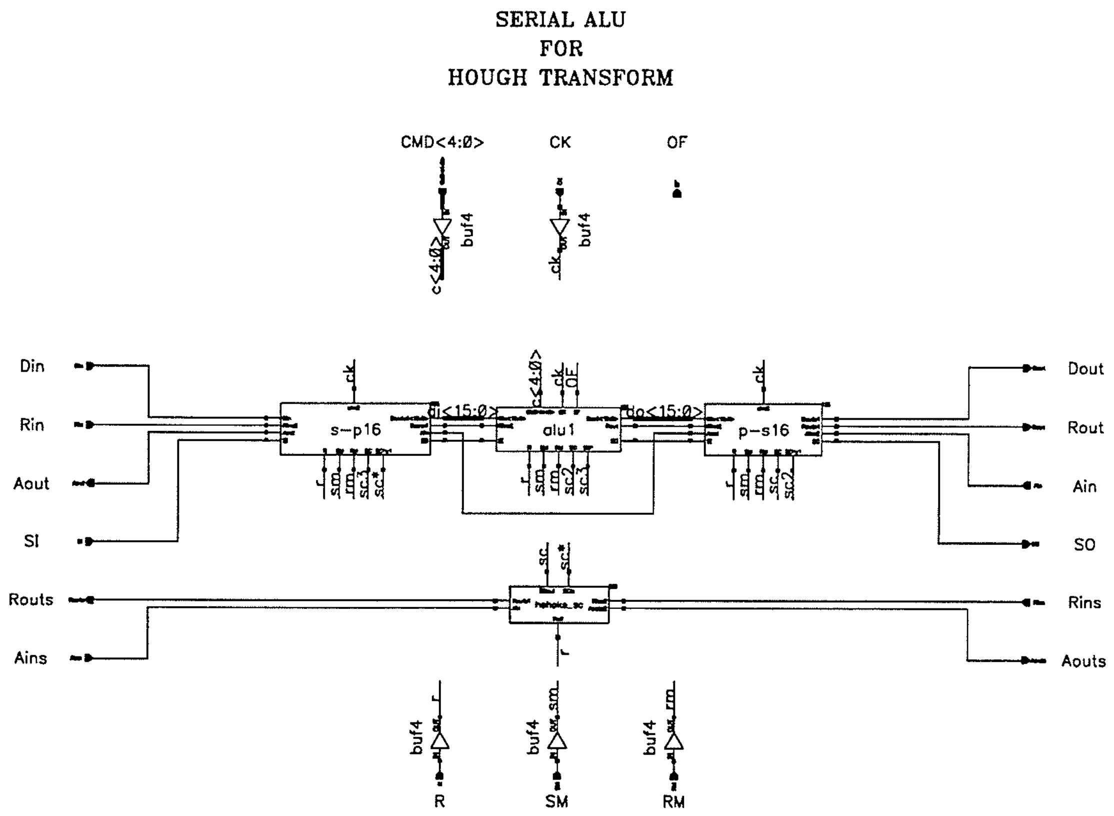


FIGURE 45 The ALU and serial to parallel converters as implemented on the test chip. A scan handshake element for the PE was also included on this chip.

3.9 PROCESSING ELEMENT

The processing element is a combinations of the input router, ALU, and output router operating bit–serially. As implemented, an array of the vertex 8 PE's would be capable of reading image data, performing (in parallel) some ray summing algorithm on it in such as the Hough transform, and writing image data. It can be programmed with up to 256 input and output selections from the 9 routes, which include the 8 I/0 directions and an initiate or terminate route code. The commands *nothing, write/read, update, backproject,* and *project* are all valid, but *update* and *backproject* were not implemented in the ALU at this time.

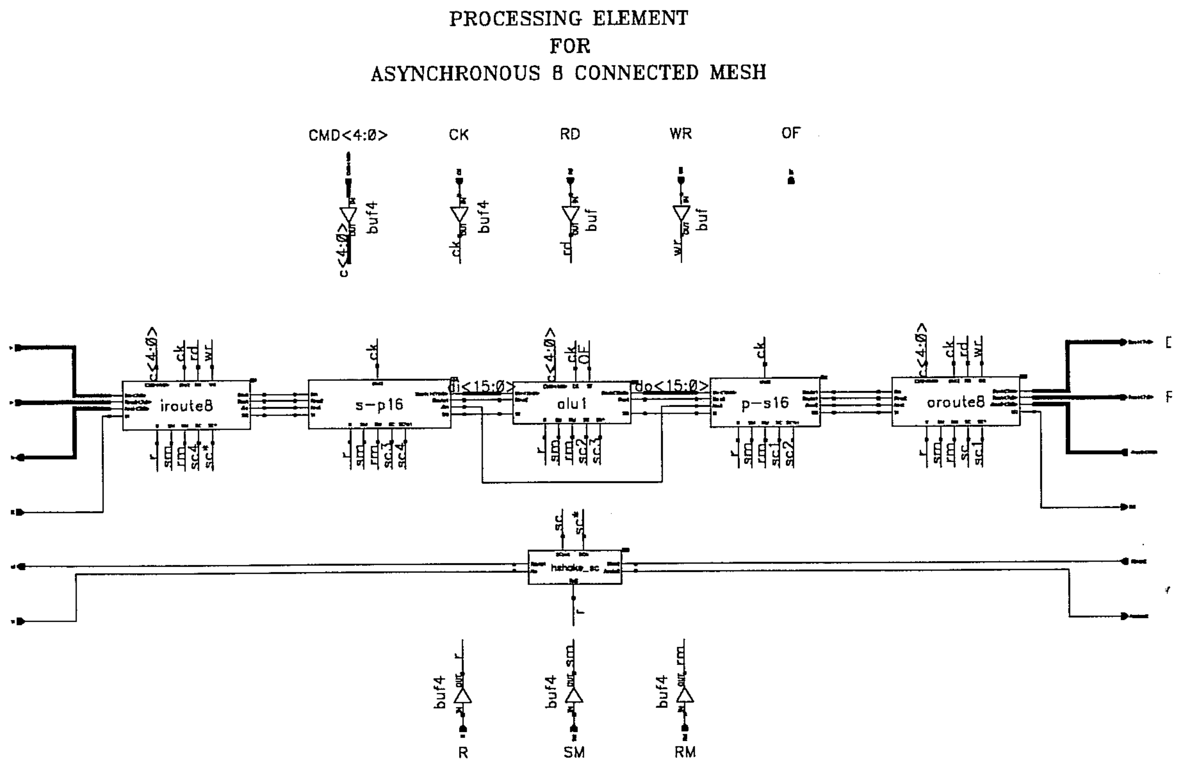


FIGURE 46 The processing element or PE.

A simulation testing the commands is shown in Figure 47. The simulations were run as fast as possible to the nearest 100us interval, so that a rough estimate of performance could be made. An explanation of what is occurring is given below:

> **0–50us, WRITE.** The PE is in *write/read* mode, as can be seen be the 08 command code. Data FF is read into the pixel register.
>
> **50–125us, PROJECT.** Two *project* operations are performed, input *initiate* to output 0 and input 7 to output 1. It is important to note the concurrent 'pipeline' operation from time 70us to 100us, where the data from the *initiate* input is being read out at the same time as data from input 7 is being read in.
>
> **125–200us, NOTHING.** The pixel value is being put into the outgoing pipeline registers, and the pipeline is being filled for a *read* pass.
>
> **200–250us, READ.** The pixel is read out.

The PE functions as desired; performance considerations must be addressed, to see if the PE can provide a reasonable degree of performance for reconstruction of images and if the area it take is small enough for a WSI implementation. This will be done in Section 3.12 after we describe the proposed I/O buffer and the micro–chips submitted for fabrication.

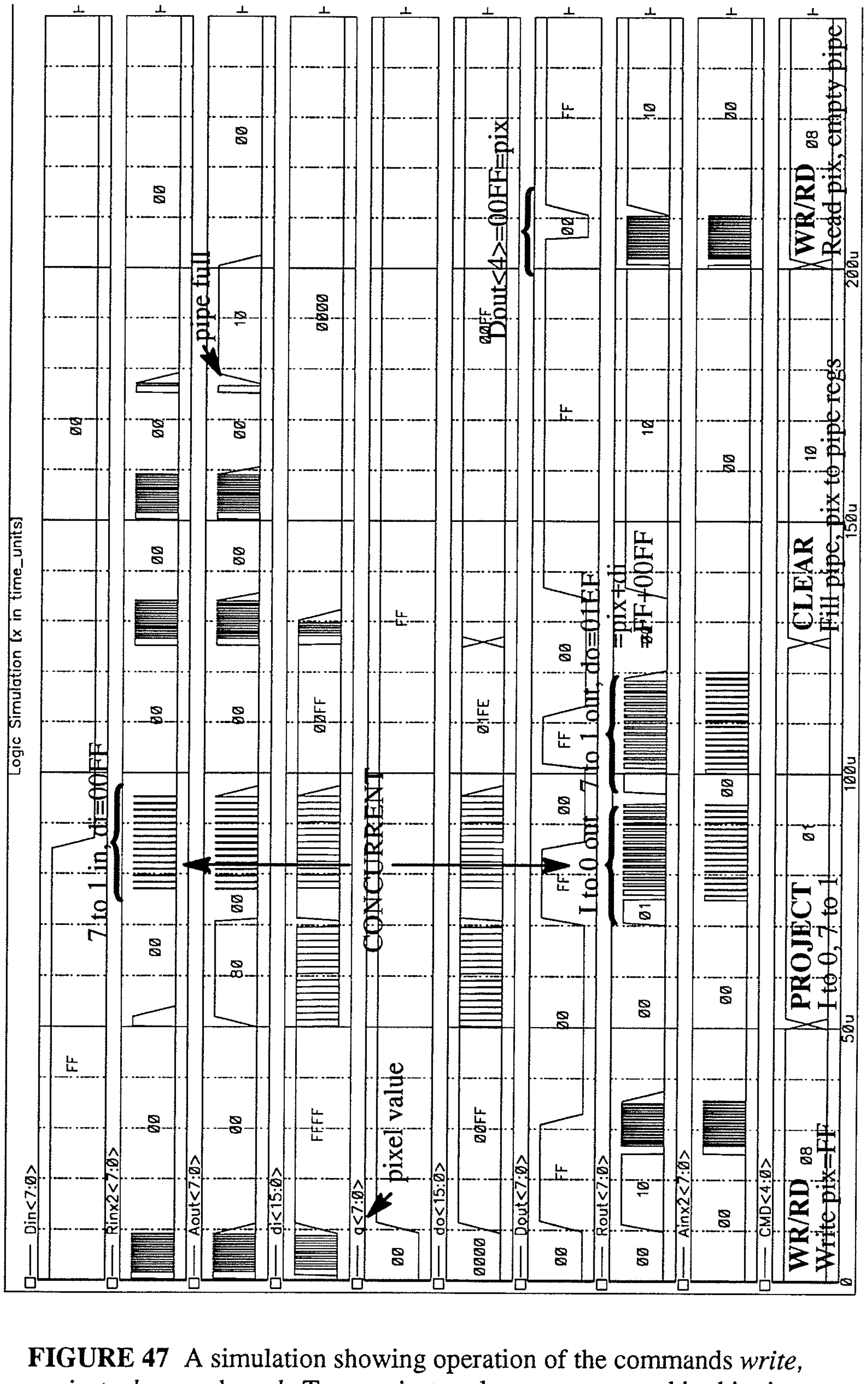


FIGURE 47 A simulation showing operation of the commands *write, project, clear,* and *read.* Two *project* cycles were executed in this simulation, from input *initiate* to output 0 and from input 7 to output 1. Refer to the text for more explanation.

3.10 I/O BUFFER

A part that we did not design that would be required for host data communications with the array is the I/O buffer, introduced in the Chapter 2. In hardware, the I/O buffer is a depth *K* bi–directional pipeline with no processing that rings the PE array and transmit or receive data from the outermost processors. The outer ring of the I/O buffer is special. The outer ring of the input buffer, for example, receives or transmits serial 16 bit wide data from the host until it is full and then initiates a parallel request to the next innermost ring of the pipeline. The outer ring or the output buffer would receive data in parallel from the next most inner ring and transmit it serially to the host. For our reconstruction algorithms, not every boundary PE will need a request/acknowledge pattern for a particular projection angle; the particular request/acknowledge pattern could be pre–calculated and loaded with the data as was done in the transputer simulations. An important consideration for the performance of the array would be the depth of the elastic I/O buffer, which could be calculated to try to optimize the usage of the host and PE array with an area trade–off.

3.11 CHIP SPECIFICATIONS

Four chips were submitted for fabrication, which will be returned in 3 months for testing. These include:

1, INPUT ROUTER	(schematic in Figure 36)
2. OUTPUT ROUTER	(schematic in Figure 36)
3. ALU (SER. VERSION)	(schematic in Figure 45)
4. MISC. TEST CHIP	(schematic in Figure 48)

The miscellaneous test chip schematic, shown in Figure 48, contains some critical parts to verify their operation for the current fabrication run parameters. It s These parts are a 3x1 bit pipeline, a delay element, a pipeline handshake element, and an asynchronous clock unit.

A fifth chip consolidating the components into a complete PE was not sent for fabrication because it was not reasonable to do so without first testing fabricated parts of its components. The PE consists of an input router, serial to parallel converter, ALU, parallel to serial converter, and an output router. An initial layout for the chip was slightly too large too be fabricated in the Northern Telecom CMOS3 process, but indications are that with some additional work the current design could be made to fit. A non–prototype with a more efficient RAM, new delay elements, and a standard cell C element would certainly fit on a single CMOS3 chip, and size would not be a concern in CMOS 4 technology, we will show in Section 3.12 that the PE size would reasonable in WSI implementation of our array. Table 5 gives specifications for each of the chips. Figures 49, 50, 51, 52, and 53 show the layouts for the processing element, input router, output router, ALU, and miscellaneous chip respectively.

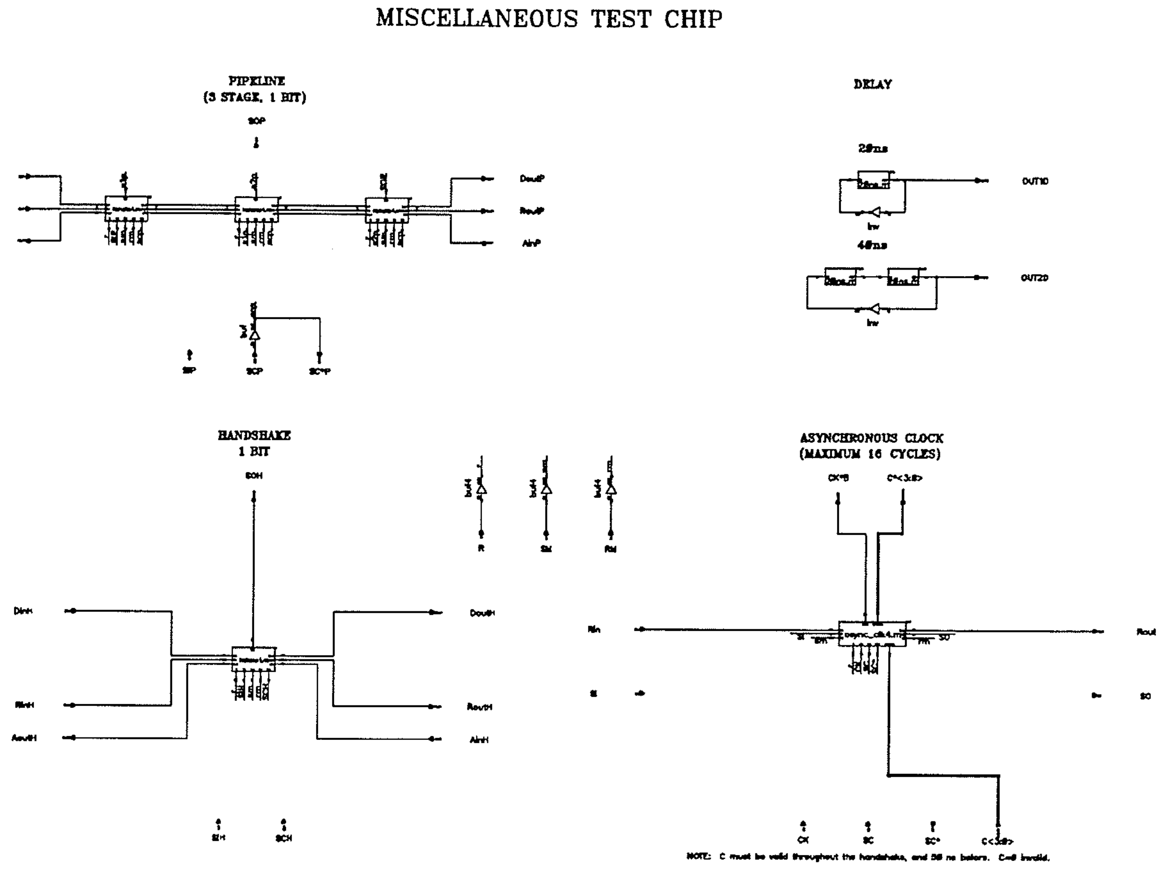


FIGURE 48 The miscellaneous test chip schematic.

CHIP SPECIFICATIONS

NAME	SIZE (um)		TRANSISTORS		I/O PINS		
	w	h	N	P	pwr	sig	total
INPUT ROUTER	5886	7274	8589	6405	10	35	45
OUTPUT ROUTER	5206	7261	7918	5287	10	39	49
ALU (SERIAL)	4704	4645	3313	3279	8	22	30
MISC.	3653	3860	887	1020	10	40	50
PROCESSING ELEMENT*	10265	7826	19810	15491	16	60	76

* All chips are 64 Pin Grid Array (PGA) packages, excluding the PE which would be an 84 PGA package. The maximum size that can be manufactured through the CMC is 7500x7500 microns.

TABLE 5 Chip specifications. The chips are being manufactured by Northern Telecom Electronics Ltd. in a 3 micron CMOS P–well process.

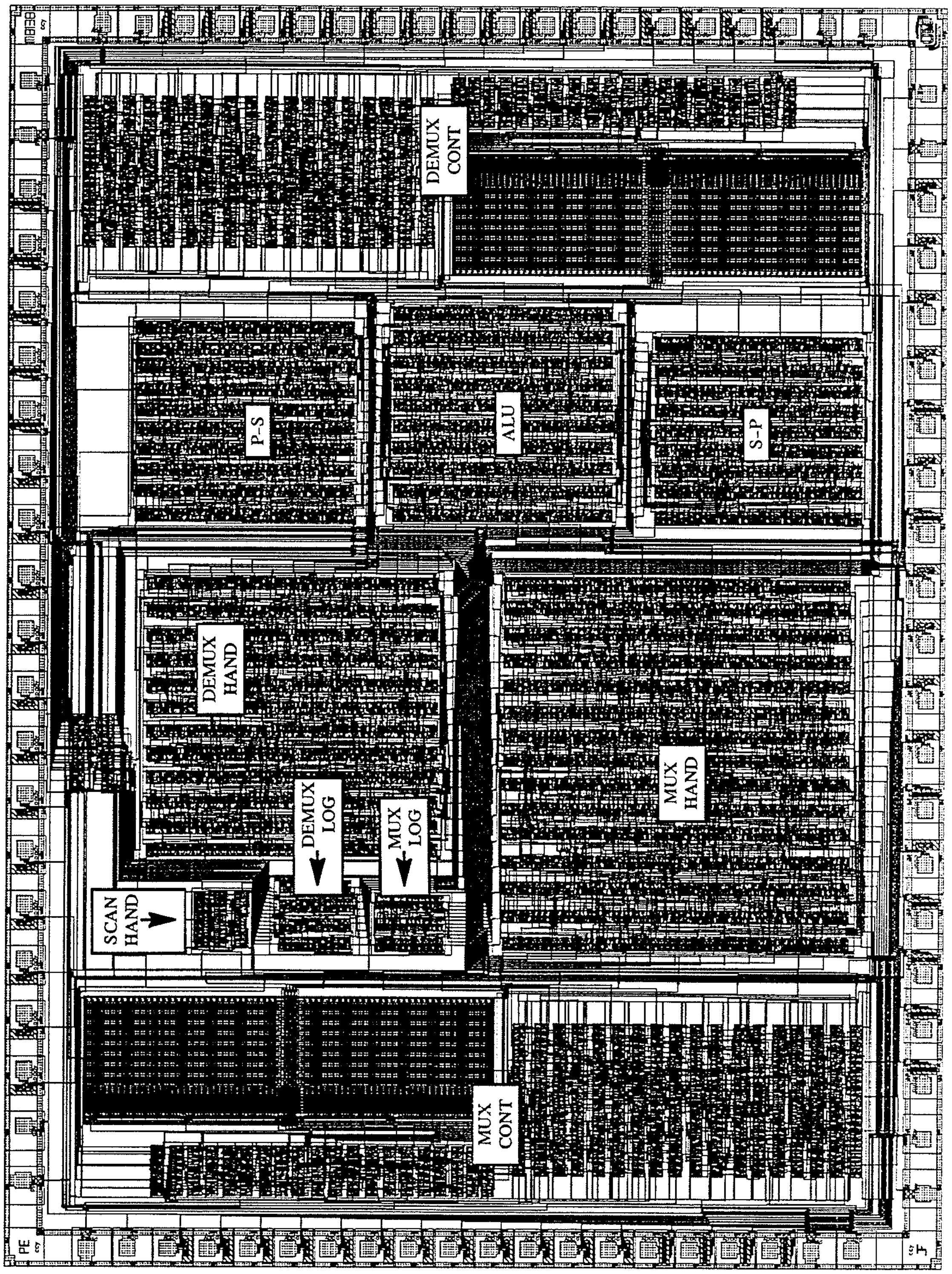


FIGURE 49 Processing element layout, showing the metal1 and metal2 layers.

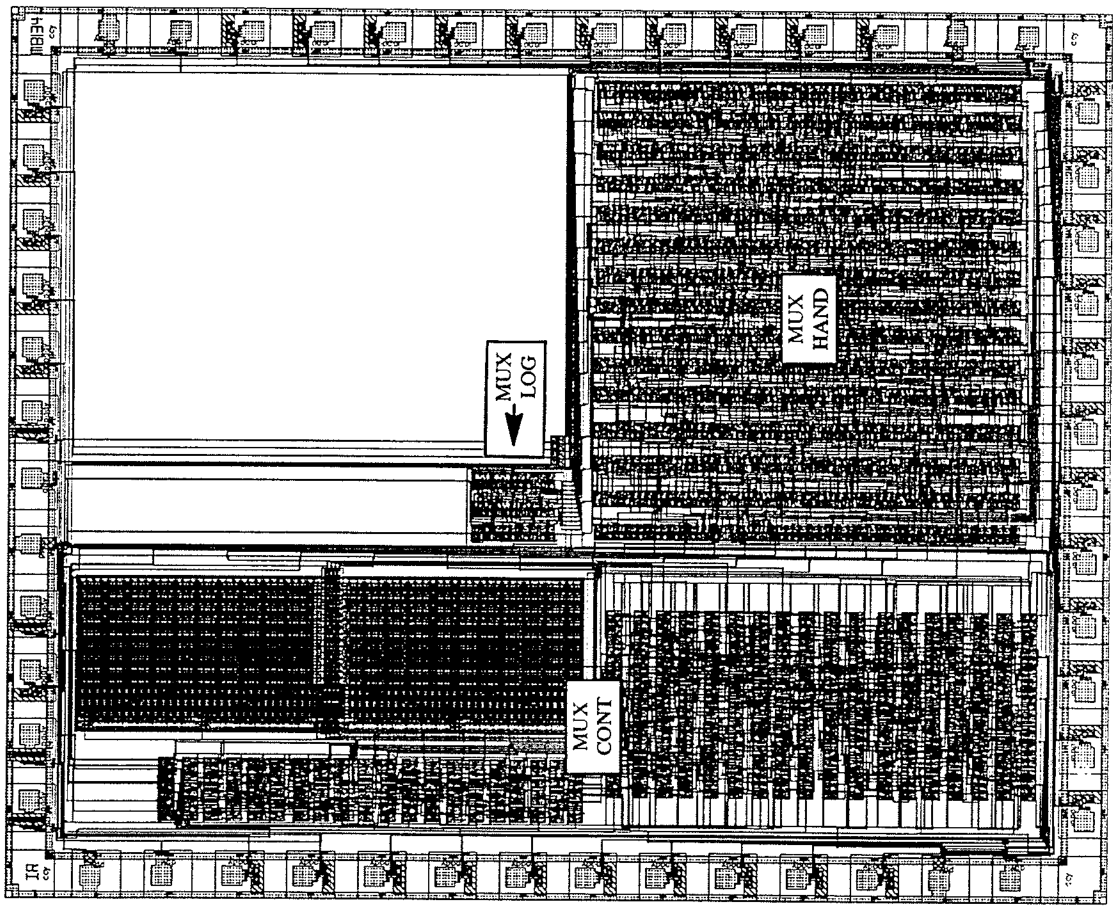


FIGURE 50 Input router layout.

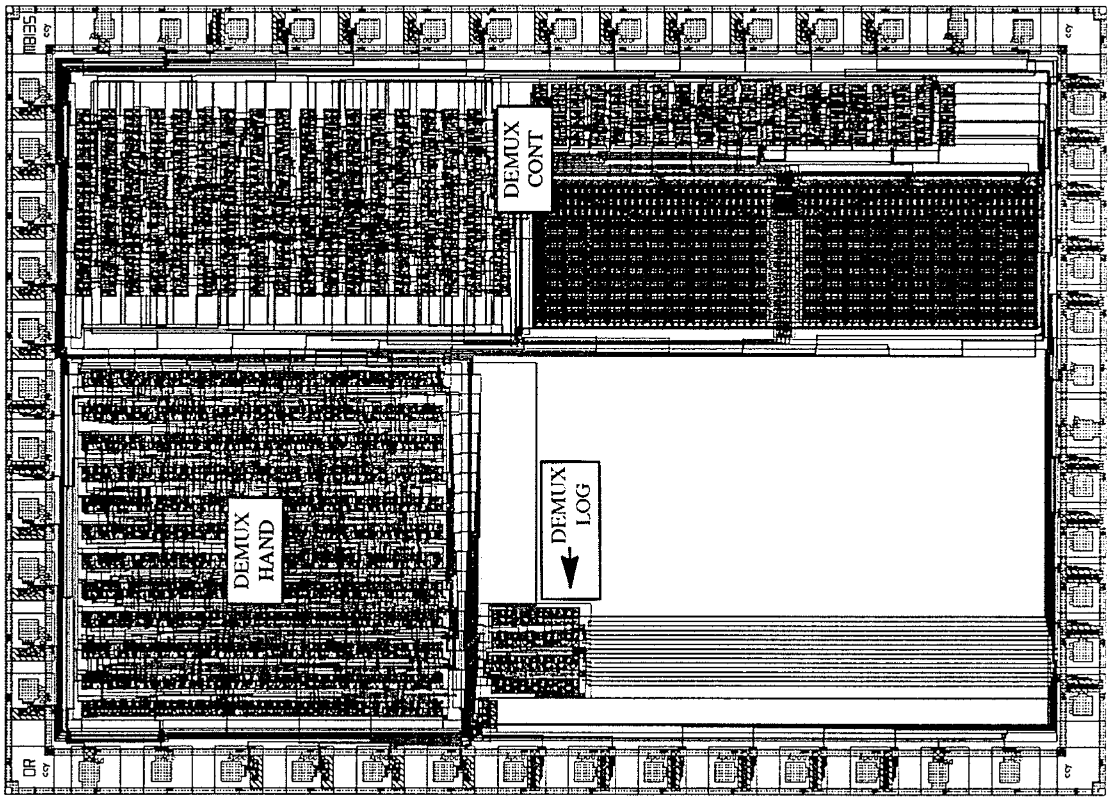


FIGURE 51 Output router layout.

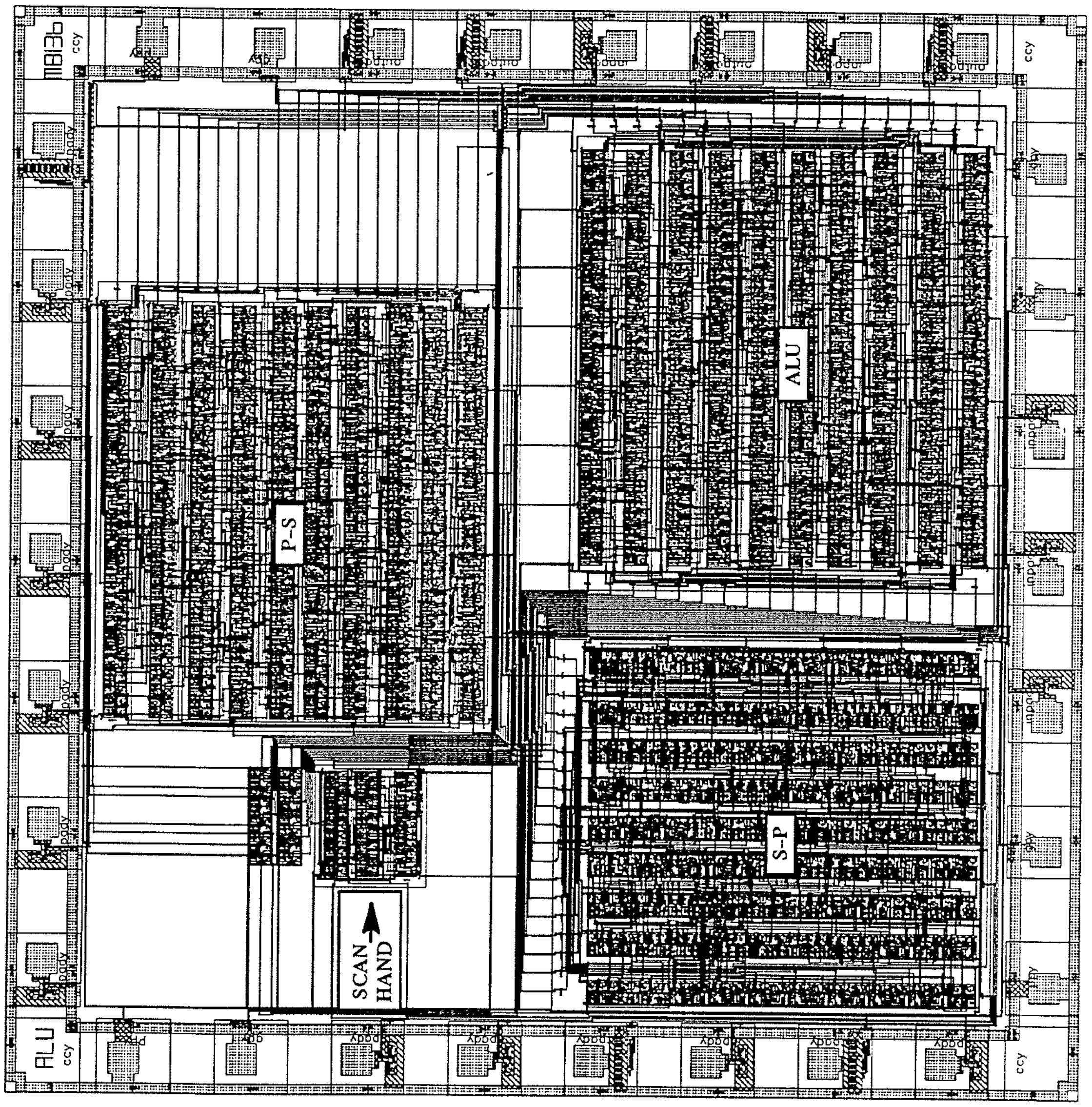


FIGURE 52 ALU layout, with serial to parallel/parallel to serial conversion.

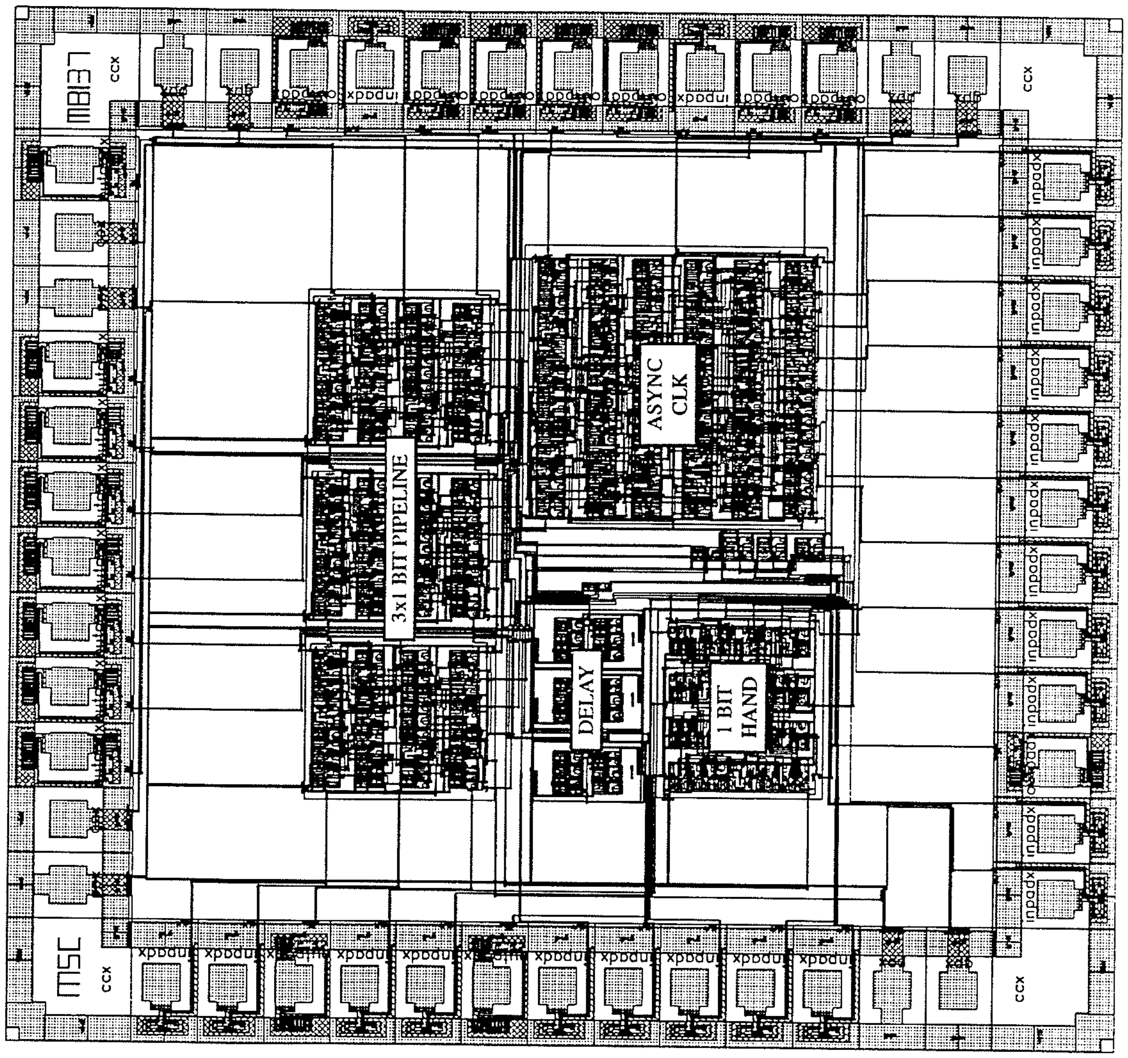


FIGURE 53 Miscellaneous chip layout. This chip includes a pipeline, handshake and asynchronous clock elements, and delay test circuitry.

3.12 AREA AND PERFORMANCE CONSIDERATIONS

The following analysis will show that our PE hardware provides sufficient speed for the image reconstruction application. Area usage is of greater concern in our prototype, but the area usage problem is not unreasonable within current technological constraints.

3.12.1 Speed

We will use the acquisition time for a 1000–element projection element for a typical CAT scan, 1 millisecond, as a benchmark for reconstruction of a 1000x1000 pixel image. This would require 1000 projections of projections data, or about 1 second of acquisition time in which to reconstruct the image to achieve near real–time operation. From the PE simulation for our bit serial PE shown in Figure 47, we estimate that it takes less that 20,000 ns (throughput) for the first bit of the 16 bit ray sum to travel from the input to the output (throughput) of the PE and that the next bit can follow in less that 1500 ns (latency). The time for one iteration, that is one pass through each projection angle, can be calculated as follows,

$$Tproj = time\ for\ first\ bit\ to\ get\ through\ array + time\ until\ last\ bit\ of\ last\ ray\ sum\ follows$$

$$= (\sqrt{N}\ x\ PE\ throughput) + (\ wordsize\ \ x\ K\ \ x\ PE\ latency) \qquad (24)$$

where K is the number of projection angles and N is the number of pixels in the image. For a 1000x1000 pixel image:

$$Tproj \approx 1000(20000) + 16(1000)(1500) \approx .044 seconds \qquad (25)$$

Equation (25) is approximately the time for 1 *project* phase of a SIRT , or the time it would take to perform the Hough transform for 1000 discrete angles. Doubling that to allow for the *backproject* phase and allowing a short time for the local *update* phase gives an iteration time of .1 seconds, or a reconstruction time of about 3 seconds for a 1000x1000 pixel image for the maximum iterations it took to achieve convergence in our simulations. This is quite good, considering that the PE's are serial and that the standard cells used are not the fastest available. Parallel communications would be used between PE's without the pin–out constraints of VLSI, and this would result in greater than 16 times speed–up. Lattard, Faure, and Mazare[19] achieved faster times; they found one transputer per pixel to be too fast for the projection acquisition equipment of a CAT scan.

3.12.2 Area

The P.E. we designed was approximately .75x1 centimeters in CMOS 3 micron technology. Given that a wafer is at least 10 in diameter, approximately a 10x10 array of the prototype PE could be fabricated in the same technology. This is too small, so many improvements would have to be made in another design iteration. Most of the area efficiency improvements have been discussed, and their implementation would result in great area savings. The prototype was designed as a "concept proof". A summary of the improvements is listed below.

1. An area efficient delay element or register latch completion detection mechanism. *Clock stealing* is an example that would result in very large area savings.
2. A more area efficient RAM.
3. A standard cell C element.
4. A more application specific route calculation method. We do not favor this since it will limit the functionality of the array. In our current architecture, a routing table of size 1k, sufficient for reconstruction of a 1000x1000 image, would mean RAM total of less that 1 gigabyte for the entire wafer. We do not consider this unreasonable for future technologies. The RAM is our design consumes the most space, and would be by far the largest consumer of area if the other improvements were done.
5. The synthesis process could be repeated for the large asynchronous parts. This would take considerable effort, since the difficulty in calculating the semi–modular STG with weakest conditions grows rapidly as more signal are introduced.[24] However, this would certainly be worthwhile for the WSI array.
6. The hierarchical schematic capture method used is the design creates unnecessary redundancy not easily seen by the designer. Much of the redundancy could be removed by minimization tools.
7. Elimination of the parallel to serial and serial to parallel circuitry the design, which would not be needed in WSI.

The locally synchronous globally synchronous design also had an area penalty. We estimate an area overhead of 40% for the self–timed asynchronous communications links above an entirely synchronous design. We believe that the area could be reduced by at least a factor of 4 by items 1, 2, and 3. Using the already available 1.2 micron CMOS twin–tub process would reduce it by a factor of 2 again. The array size is approaching the size required for high speed commercial applications without considering future fabrication process improvements.

CHAPTER 4

Conclusions

Reconstruction of images from their parallel beam projections is a difficult topic on which much research has been done, and only recently have attempts been made to parallelize the process. To date there is no commercial massively parallel implementation. This thesis studied a novel architecture suitable for WSI (Wafer Scale Integration) on which algorithms for image reconstruction from parallel beam projections or other cyclic image processing algorithms could be embedded, namely the mesh of vertex–8 processors with one processing element per pixel. The main result is that this architecture would be a practical one for the image reconstruction from parallel beam projections problem. To prove this point, the focus of the thesis was on the hardware design of a locally synchronous globally asynchronous vertex–8 processing element suitable for implementation in WSI. The process of the hardware design gave results and insights important to more general parallel processing WSI applications.

Before beginning the hardware implementation, simulations of the sequential and parallel algorithms were done that justified the need to parallelize the iterative reconstruction algorithms and verified that the algorithms could be mapped onto our proposed architecture. Iterative reconstruction techniques dubbed ART and a modification of ART called SIRT were tested for their suitability for a parallel embedding on the mesh, and the custom processing element for the mesh was designed according to the test results. The 8 connected mesh was shown to be advantageous for the image reconstruction algorithms we used. Sequential simulations were done on a PC and parallel simulations on a transputer array. The sequential simulations showed that the iterative algorithms could provide a more accurate reconstruction than commercially common algorithms but with a large time penalty, while the parallel simulations provided a functional hardware test that aided in the formal hardware specification. Improvements to our proposed hardware such as backprojecting fixed point update data were suggested by the simulations, and applied to our hardware design.

Asynchronous circuits are likely to play increasingly important role in large scale VLSI systems, and this was emphasized in our hardware design. The architecture is a fine–grained mesh of custom processors (vertex–8) that is locally synchronous and globally asynchronous, an architecture which we purport to be suitable for WSI. The mixed synchronous/asynchronous architecture has a cost in silicon area, estimated to be at least 40% above that of a corresponding synchronous design, but we conclude that the overhead cost is necessary and worthwhile to allow a parallel processing WSI implementation. The PE we designed has several other unique hardware features for WSI, including self–timed handshaking elements for inter–processor communications and a novel asynchronous scan test structure. The asynchronous scan test structure is an important contribution, since it allows testing of asynchronous and synchronous parts that would exist in a WSI environment. We chose to make the PE an escapement machine so that the parallel design is less complex and reliable; no queuing strategy is needed and there are no metastability problems. The design techniques and methodologies presented are also directly relevant to any excessively large fully synchronous design.

A performance analysis done on our prototype PE shows that in its present form an array of the PE's would perform at a sufficient speed to be competitive with the Fourier techniques. The assumptions made in the performance analysis were cautious and reasonable. Area usage of our prototype was inefficient, but this could be corrected in future designs and even 'as–is' the PE is a reasonable size for more recent smaller IC technologies. Performance of the PE will be verified when the test chips that have been sent for fabrication return. The detailed analysis we have done indicates that both speed and area parameters are reasonable for a WSI implementation.

4.1 FUTURE WORK

Much other work has been done on the algorithms themselves, and our purpose was not to develop or change the algorithms. However, more parallel simulations of the algorithms would lead to more insight on hardware requirements. Simulations with larger images (at least 32x32) from realistic projection data could be done. Simulations with the actual ray integral, with noisy data and with real data should be done before another hardware submission. We chose to focus on a particular set of algorithms; it should be clear that many other algorithms could be embedded onto our proposed array by loading in different routing patterns. These other algorithms, which include image transform and pattern recognition problems, can be investigated in more detail.

The hardware itself could be made much more area efficient by some of the suggestions made in the performance analysis. In particular, the delay element can be improved by using clock stealing. Research on quiescent current monitoring to detect latch completion in place of a delay element is another option. This is a difficulty associated with static CMOS design, which we chose as our implementation technology because of the tools available. It would be interesting to pursue other logic families, such as DVCSL, on which work on self–timed synthesized asynchronous circuit has been done. Besides the delay element, immediate work that could be done is to customize the Muller C element.

The prototype chips must be physically tested when they return from fabrication to verify our design. The next step would be to implement and test a small array of the PE's with bit parallel communications and an I/O buffer in 1.2 micron CMOS, leading finally to a WSI implementation. The WSI implementation would require much additional work since a fault tolerant scheme would have to be devised, consisting of testing the PE's and switching in spares.

4.2 CLOSING COMMENTS

It is the author's opinion that the rewards of asynchronous design will be worth the penalties in a massively parallel WSI designs of the future. We believe the major contribution of the thesis was in the design of the asynchronous scan test structure, which would allow scan testing on mixed synchronous/asynchronous circuits. Image processing problems such as image or field reconstruction will be some of the first algorithms to be customized in WSI. Much future work will be done in area of parallelization of image or field reconstruction algorithms, as it is a problem that demands parallelization.

* * *

References

[1] Bayford R. "The Bit–Serial Systolic Back–Projection Engine (BSSBPE)", *Proc. 1990 Int. Conf. on Application Specific Processors*, Sept. 1990, 43–54.

[2] Budinger T.F., Gullberg G.T. "Three–D. Reconstruction in Nuclear Medicine Emission Imaging", *Special Issue on Physical and Computational Aspects of 3–Dimensional Image Reconstruction, IEEE Trans. Nuclear Science,* V. 21 No. 3, June 1974, 2–20.

[3] Canadian Microelectronics Corporation, *The CMOS3 DLM Cell Library,* July, 1989.

[4] Chapiro, D. M. "Globally Asynchronous Locally Synchronous Circuits", Ph.D. dissertion, Stanford Univ., Oct. 1986.

[5] Cho Z.H. (Ed.). *Special Issue on Physical and Computational Aspects of 3–Dimensional Image Reconstruction, IEEE Trans. Nuclear Science,* V. 21 No. 3, June 1974.

[6] Cho, Z.H. "General View on 3–D Reconstruction and Computerized Transverse Axial Tomography", *Special Issue on Physical and Computational Aspects of 3–Dimensional Image Reconstruction, IEEE Trans. Nuclear Science,* V. 21 No. 3, June 1974, 44–71.

[7] Comer D.J. *Digital Logic and State Machine Design*, Saunders College Publishing, 2nd Edition, 1990.

[8] Cypher R.E., Sanz J.L.C., Snyder L. "The Hough Transform has O(N) Complexity on NxN Mesh Connected Computers", *SIAM J. Computing*, Vol. 19, No. 5, Oct. 1990, 805–820.

[9] Dijkstra E. "Guarded Commands, Non–determinacy and Formal Derivation of Programs", *Comm. Assc. Comput. Math.,* Vol. 18, No. 8, Aug. 1975, 453–457.

[10] Gagne D. "Module Generation in the Cadence Design Environment: Users Manual", UBC/CMC, 1992.

[11] Gilbert P. "Iterative methods for the reconstruction of three dimensional objects from their projections", *J. Theor. Biol.*, Vol. 36, 1972, 105–117.

[12] Gordon R. "A tutorial on ART (Algebraic Reconstruction Techniques)", *IEEE Trans. Nucl. Sci.,* NS–21, 1974, 78–93.

[13] Gordon R., Bender R., Herman G.T. "Algebraic reconstruction techniques (ART) for three–dimensional electron microscopy and x–ray photography", *J.Theor. Biol.,* Vol. 29, 1970, 471–481. Also in [5]

[14] Herman, G. T. (Ed.) *Image Reconstruction from Projections,* Academic Press, 1980.

[15] Jacobs G.M. "Self–timed Integrated Circuits for Digital Signal Processing", Ph.D. dissertion, Dept. of EECS, Univ. of California, Berkeley, 1989.

[16] Kak, A.C. "Image Reconstruction from projections", in Ekstrom M.P. (Ed.) *Digital Image Processing techniques Vol 2: Computational Techniques*, Academic Press, Orlando 1984.

[17] Kung S. Y. (Ed.) *Proc. 1990 Int. Conf. on Application Specific Array Processors*, Sept. 1990.

[18] Kung S. Y. *VLSI Array Processors,* Prentice Hall, 1988.

[19] Lattard D., Faure B., Mazare. G. " Massively Parallel Architecture: Application to Neural Net Emulation and Image Reconstruction", *Proc. Int. Conf. Application Specific Array Processors*, Sept. 1990, 215–225.

[20] Lattard D., Mazare G. "Image Reconstruction Using an Original Asynchronous Cellular Array", *Proc. 1989 Int. Symposium on Circuits and Systems,* Vol. 1, May 1989, 13–16.

[21] Lehn W. *Digital Image Processing lecture notes*, 1991.

[22] Leighton F. *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann, 1992.

[23] Mead C., Conway L. *Introduction to VLSI Systems*, Addison–Wesley, 1980.

[24] Meng T. H. –Y., Brodersen R. W., Messerschmitt D. G. "Synthesis of Asynch. Circuits from High–Level Specifications", *IEEE Trans. Comp.–Aided Design,* Vol. 8 No. 11, Nov. 1989, 1185–1205.

[25] Nadeau–Dostie B., Soufi M., Savaria Y. "Training Course Session of BNR DFT Tools", Ecole Polytechnique de Montréal, May 1992.

[26] Negrini M., Sami M.G., Stefanelli R. *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays,* MIT Press, 1989.

[27] Ohio State Supercomputer Center, *Trollius Users Manual,* 1992.

[28] Oppenhiem B.E. "More Accurate Algorithms for Iterative 3–D Reconstruction", *Special Issue on Physical and Computational Aspects of 3–Dimensional Image Reconstruction, IEEE Trans. Nuclear Science,* V. 21 No. 3, June 1974, 72–77.

[29] Radon J. "Ber. Saechs. Akad. Wiss. Leipzig, ", *Math. Phys. Kl.* Vol. 69, 1917, 262–277.

[30] Rosenfeld A., Kak A.C. *Digital Picture Processing*, Academic Press, 1982.

[31] Seitz C. "System Timing", in Mead C., Conway L. *Introduction to VLSI Systems*, Addison–Wesley, 1980. Chapter 7, 218–262.

[32] Shepp L.A., Logan B.F. "Fourier Reconstruction of a Head Section", *Special Issue on Physical and Computational Aspects of 3–Dimensional Image Reconstruction, IEEE Trans. Nuclear Science,* V. 21 No. 3, June 1974, 21–43.

[33] Sutherland I. "Micropipelines", *Communications of the ACM,* Vol. 32 No. 6, June 1989, 720–738.

[34] Tanabe K. "Projection method for solving a singular system", *Numer. Math.*, Vol. 17, 1971, 203–214.

[35] Volder J. "The CORDIC Trigonometric Computing Technique" *IRE(IEEE) Trans. Electronic Computers,* Vol. EC–8 No. 3, Sept. 1959, 330–334.

[36] Walther, J. A unified algorithm for elementary functions. *Proc. Spring Joint Computer Conference,* 1971, 379–385.

[37] Weste N., Eshraghian K. *Principles of CMOS VLSI Design, a Systems Perspective,* Addison–Wesley, 1988.

APPENDIX A

Acronyms and Abbreviations

ALU	Arithmetic Logic Unit
ART	Algebraic reconstruction technique
ASIC	Application specific integrated circuit
CMOS	Complimentary metal–oxide semiconductor
CMC	Canadian Micro–electronics Corporation
demux	Demultiplexer
fab	Fabrication
FBP	Filtered Back Projection
IC	Integrated circuit
MIMD	Multiple instruction multiple device
MSE	Mean squared error
mux	Multiplexer
NT	Northern Telecom
PE	Processing element
PGA	Pin grid array
RAM	Random access memory
SBP	Simple Back Projection

SIMD	Single instruction multiple data
SIRT	Simultaneous Iterative Reconstruction Technique
VLSI	Very large scale integration
WSI	Wafer scale integration

APPENDIX B

Technical Reference Addendum Contents

The *Technical Reference Addendum* contents are shown on the following pages. The *Technical Reference Addendum* can be obtained from:

Address:	J.W. Fitchett or Dr. R.D. McLeod Dept. of Computer & Electrical Eng. University of Manitoba 15 Gillson Street Winnipeg, MB. Canada R3T 5V6
Phone:	(204) 474–9603
Fax:	(204) 261–4639
Email:	fitch@ee.umanitoba.ca or mcleod@ee.umanitoba.ca

Technical Reference Addendum Table of Contents

More contents on following page...