HIGH SPEED SEQUENTIAL DEMODULATOR BASED ON SHIFT REGISTER SYSTOLIC PRIORITY QUEUE ARCHITECTURE

by

Hoo Man Ng

A Thesis

Presented to the Faculty of Graduate Studies

in Partial Fulfillment of the Requirements for the Degree

Master of Science

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba

ISBN 0-315-85931-8

Canada

Name _____

Dissertation Abstracts International is arrangea 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.

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

HIGH SPEED SEQUENTIAL DEMODULATOR

BASED ON SHIFT REGISTER

SYSTOLIC PRIORITY QUEUE ARCHITECTURE

BY

HOO MAN NG

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

The sequential algorithm has been applied to channels with intersymbol interference (ISI). It is superior to the Viterbi algorithm for channels with long or even infinite intersymbol interference, since the computation and storage complexity of the sequential algorithm does not grow exponentially with the length of the channel memory. However the time required for the complete stack reordering procedure in the conventional sequential stack algorithm is long and depends on the number of entries in the stack, hence places severe limitations on the decoding speed and thus the real–time implementation of the algorithm.

In this thesis, the Shift Register Systolic Priority Queue is used to substitute the stack of the conventional sequential stack algorithm to eliminate the complete stack reordering problem. With the systolic priority queue architecture, complete stack reordering is no longer required and the path with the largest path metric is guaranteed to appear at the top of the stack within a fixed and short interval of time regardless of the number of paths in the stack.

Hardware algorithms for storing the explored paths and evaluating the metrics are developed in this thesis. A sequential demodulator based on the Shift Register Systolic Priority Queue architecture and the developed algorithms was implemented. The design is capable of handling input blocks of 256 bits and a maximum of 9 interference terms. It has a maximum allowable stack size of 1024. An 8–bit representation is used for the sufficient statistic z_k and a 16–bit representation for the metric.

ACKNOWLEDGEMENTS

The author wishes to express his sincere thanks to Professor E. Shwedyk for his supervision and guidance throughout the course of the research.

I hereby declare that I am the sole author of this thesis. I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Hoo Man Ng

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in whole or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Hoo Man Ng

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

The increasing demand for higher rate digital data transmission makes digital communication an area of intensive research. Due to physical constraints, communication channels are bandlimited. One example is the voice–grade–telephone channel that has a bandwidth of approximately 4,000 Hz despite the fact that modems are transmitting data over this channel at a rate of approximately 24,000 bits/sec. The bandlimited characteristic results in intersymbol interference (ISI) [4], a phenomenon where each transmitted pulse stretches beyond the time interval allocated to that particular pulse and overlaps with pulses in other time intervals. The occurrence of ISI is caused by the time–dispersive characteristics of bandlimited channels. The number of pulses interfering with a particular pulse is called the memory of the channel (ν) or the length of the ISI. In high speed data transmission over bandlimited channels with high signal–to–noise (SNR) ratios, the existence of ISI becomes the major obstacle to reliable communication.

Various techniques for combating ISI have been studied. In 1928, Nyquist [19], being the first researcher to develop techniques for combating ISI, introduced baseband spectrum shaping for zero ISI. This eliminates ISI by using an equivalent filter at the receiver to suppress the ISI terms at any sampling instant. In 1963, Lender [15] introduced the duobinary signaling technique which allows the existence of one ISI term. Lender's technique allows ISI in a controlled manner so that it can be removed at the receiver. In 1966, Lender [16] and Kretzmer [12, 13] generalized the duobinary signaling technique to partial response or correlative coding which allows any number of ISI terms. During 1968–1969, Tomlinson [24] from the United Kingdom and Harashima [10, 11] from Japan invented the precoding technique [9]. With the assumption that the channel response is known at the transmitter, the input sequence is coded in a unique way before transmission through the channel. Equalization techniques were also introduced for combating ISI during the late 1960's. Channel equalization is performed by the receiver that usually consists of a matched

filter followed by a sampler and an equalizer. There are three main categories of equalization techniques [9, 22], namely linear equalization (LE), decision–feedback equalization (DFE), and maximum–likelihood sequence estimation (MLSE). In linear equalization [18], the present and past outputs of the matched filter are weighted by estimated gains and summed to produce the output. Decision–feedback equalization [2] is the simplest form of non–linear equalization. A decision–feedback equalizer improves upon a linear equalizer by passing the output of a linear equalizer through a second equalizer with feedback. MLSE, a sequence estimation technique, is considered the optimum equalization structure for communication channels with finite ISI and is found to have an error performance superior to the conventional symbol–by–symbol decision receivers mentioned above. In 1972, Forney [7] showed MLSE can be realized for channels with finite ISI. The maximum–likelihood sequence estimator consists of a whitened matched filter followed by a Viterbi processor. The detection task is modelled as a search for the best (maximum–likelihood) path through a regular structure called a trellis.

The Viterbi Algorithm [8, 17] was invented by Viterbi in 1967 as a method for decoding convolutional codes. Since the ISI channel characteristic is equivalent to that of the convolutional encoder, Forney [7] applied the VA to ISI channels. During the decoding process, the VA visits all the paths through the trellis in order to find the maximum–likelihood path. The fixed number of computations and the regular decoding procedure make the Viterbi processor very easy to implement. However, the computational complexity and the storage complexity of the VA grow exponentially with the length of the channel memory. For ISI channels with large or infinite memory, implementation of the maximum–likelihood sequence estimator using VA becomes impractical.

In order to reduce the computational complexity of the VA, a great deal of effort has been made to reduce the number of states in the trellis to which the VA is applied. The techniques used for reducing the number of trellis states are known as RSVA (Reduced–State Viterbi Algorithm) [1, 5, 6, 14, 20, 21, 23, 25]. These techniques reduce the number of trellis states

either by reducing the number of the most likely paths to be searched or the length of the channel memory. The RSVA techniques have the advantage of retaining the structure of MLSE–VA while reducing the computational complexity of MLSE–VA. However, for any specific technique, gains in computational complexity comes with a loss in detection performance.

In 1989, Xiong [26] developed the application of the sequential algorithm (SA) [17] to ISI channels. The sequential algorithm is another technique for decoding convolutional codes. Since the ISI channel characteristic is equivalent to that of the convolutional encoder, Xiong applied the SA to ISI channels. Unlike the VA, the SA visits only a small number of paths through the tree. The computational complexity of the SA is almost independent of the length of the channel memory and its detection performance is essentially maximum–likelihood. This property makes the SA applicable to ISI channels with large or even infinite memory. Compared to the VA for ISI channels, the sequential approach is able to handle more severe ISI channels (large or infinite memory) and therefore allows data to be transmitted at higher rates.

Although the sequential approach is superior to the Viterbi approach in its ability to handle a greater number of interference terms, it involves a very time consuming stack reordering. Not only is the time required for stack reordering long, but it also varies with the number of paths in the stack. It is undesirable for the operation time to be dependent upon the number of paths in the stack since this number increases after each decoding step. Since the only purpose of stack reordering is to have the best path placed at the top of the stack for the next decoding step. Chang and Yao [3] proposed the use of a Systolic Priority Queue. With the systolic priority queue architecture complete stack reordering is not necessary and the best path is always placed within a fixed and short period of time at the top of the stack for the next decoding step.

The objective of this thesis is to examine the VLSI implementation of a sequential demodulator based on the systolic priority queue architecture applicable to ISI channels. Chapter 2 provides a general background of the ISI channels, followed by a description of the Viterbi algorithm, sequential algorithm and Systolic Priority Queue. Chapter 3 describes the details in the development of the sequential demodulator. Chapter 4 presents the testing result and the discussion of the design. Chapter 5 gives the conclusions and suggestions for further study.

Chapter 2 Background and Theory

2.1 Channel Model and Receiver Structure for Intersymbol Interference (ISI)

Intersymbol interference (ISI) arises in all pulse–modulation systems whenever the impulse response of the channel is longer than one transmission time period (T). Pulse amplitude modulation (PAM) system is the simplest digital communication system that can be used to illustrate ISI. Figure 2.1 shows a simplified version of the baseband PAM communication system. The input data $x(t)$ is modeled as a train of impulses equally spaced at T–seconds intervals with specific weights x_k, where x_k are drawn from a discrete finite alphabet $\{0, 1, \ldots, m-1\}$.

$$x(t) = \sum_{k=0}^{k=K} x_k \delta(t - kT) \ , \tag{2.1}$$

K may be finite or infinite. $h(t)$ is the finite impulse response of the channel with a length of L symbol intervals, i.e. L is the smallest integer such that $h(t) = 0$ for $t \geq LT$. The impulse response $h(t)$ is assumed to be square–integrable,

$$\| h \|^2 \triangleq \int_{-\infty}^{\infty} h^2(t) dt < \infty \ . \tag{2.2}$$

If $h(t)$ has nonzero values at sampling instants $t = t_0 \pm kT$ for $k = 1, 2, \ldots, K$, then ISI occurs. The number of nonzero sampling values in $h(t)$ (except at $t = t_0$) is called the length of ISI or the channel memory (ν), where $\nu = L - 1$. The output of the channel impulse response, which is the convolution sum of $x(t)$ and $h(t)$, is denoted as $s(t)$.

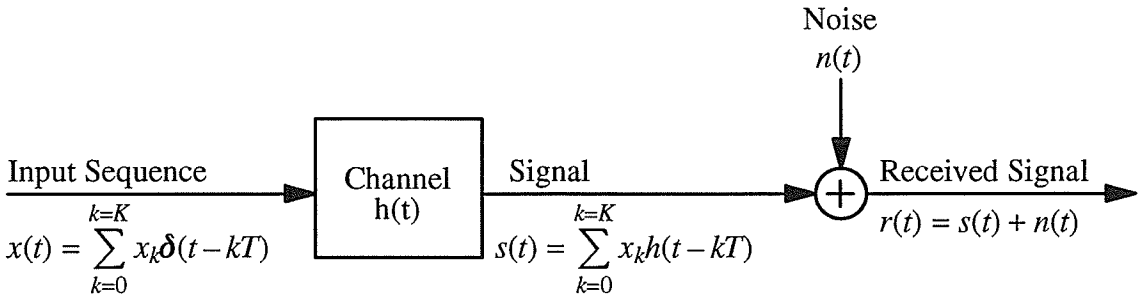


Figure 2.1: PAM communication system.

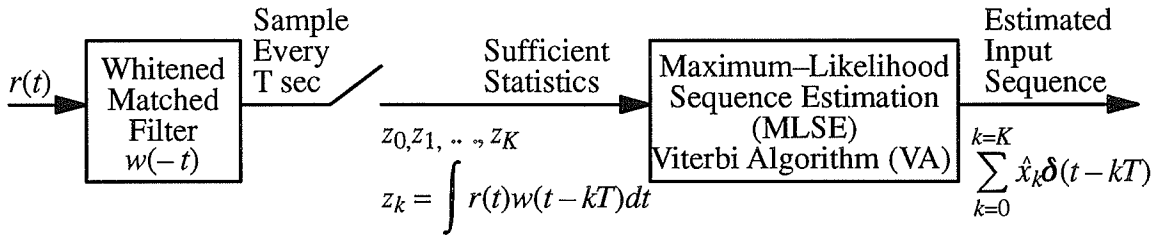


Figure 2.2: Maximum–likelihood sequence estimator.

$$s(t) = x(t) * h(t) \ ,$$

$$= \sum_{k=0}^{k=K} x_k h(t - kT) \ . \tag{2.3}$$

The output signal $s(t)$ is corrupted by additive white Gaussian noise $n(t)$ to yield the received signal $r(t)$.

$$r(t) = s(t) + n(t) \ ,$$

$$= \sum_{k=0}^{k=K} x_k h(t - kT) + n(t) \ . \tag{2.4}$$

$r(t)$ is the received signal corrupted by white Gaussian noise and intersymbol interference. If $r(t)$ is sampled at $t = jT$, where t_0 accounts for the channel delay and sampler phase, then

$$r(t_0 + jT) = x_j h(t_0) + \sum_{k \neq j} x_k h(t_0 + jT - kT) + n(t_0 + jT) \ . \tag{2.5}$$

Desired Output ISI Noise

The output is corrupted by ISI unless $h(t)$ is zero at all sampling instants except at $t = t_0$. The first term on the right side is the desired output produced by the input symbol at $t = jT$, the second term is the intersymbol interference and the last term is the additive white Gaussian noise. The presence of ISI is the primary impediment to reliable high–rate digital data transmission over high signal–to–noise ratio bandlimited channels such as voice–grade telephone circuits.

In 1972, Forney [7] developed a receiver structure for the maximum–likelihood estimation of digital sequences in the presence of ISI. The receiver consists of a whitened matched filter, a symbol–rate sampler and a Viterbi processor as shown in Figure 2.2. The sampled

outputs of the whitened matched filter form a set of sufficient statistics $\{ z_k \}$ for the estimation of the input sequence $\{ x_k \}$.

Equation (2.3) shows that $s(t)$ can be expressed as a linear combination of a set of square–integrable basis functions $h(t-kT)$. In the detection of signals that are a linear combinations of a set of square–integrable basis functions, the outputs of a bank of matched filters, each matched to a basis function, form a set of sufficient statistics for estimating the input sequence. Thus the $K+1$ quantities

$$a_k \triangleq \int_{-\infty}^{\infty} r(t)h(t-kT)dt \ , \qquad 0 \le k \le K \tag{2.6}$$

form a set of sufficient statistics for the estimation of the input sequence $\{ x_k \}$, $0 \le k \le K$, where K may be finite or infinite. But a_k are just the convolution integral of $r(t)$ and $h(-t)$, i.e. a_k are the sampled outputs of a filter $h(-t)$ matched to the channel impulse response $h(t)$.

By applying the D–transform to the matched–filter output sequence a_k, equation (2.6) can be defined as

$$a(D) \triangleq \sum_{k=0}^{k=K} a_k D^k \ . \tag{2.7}$$

Since

$$a_k \triangleq \int_{-\infty}^{\infty} r(t)h(t-kT)dt$$

$$= \int_{-\infty}^{\infty} \sum_{j=0}^{K} x_j h(t-jT)h(t-kT)dt + \int_{-\infty}^{\infty} n(t)h(t-kT)dt$$

$$= \sum_{j=0}^{K} x_j \int_{-\infty}^{\infty} h(t-jT)h(t-kT)dt + \int_{-\infty}^{\infty} n(t)h(t-kT)dt$$

$$= \sum_{j=0}^{K} x_j R_{k-j} + n_k' \ , \tag{2.8}$$

the D–transform of a_k can be expressed as

$$a(D) = x(D)R(D) + n'(D) \ . \tag{2.9}$$

In equation (2.8),

$$R_{k-j} \triangleq \begin{cases} \int_{-\infty}^{\infty} h(t-jT)h(t-kT)dt & |k-j| \leq \nu \\ 0 & |k-j| \geq \nu + 1 \end{cases} \tag{2.10}$$

are the pulse autocorrelation coefficients of $h(t)$. It follows that

$$R(D) \triangleq \sum_{k=-\nu}^{k=\nu} R_k D^k \tag{2.11}$$

is the D–transform of the pulse autocorrelation function or the spectral function of $h(t)$. Since $R(D)$ is finite with $2\nu + 1$ nonzero terms, it has 2ν complex roots. Furthermore, since $R(D) = R(D^{-1})$, the inverse β^{-1} of any root β is also a root of $R(D)$. Thus, the 2ν complex roots of $R(D)$ may be grouped into ν inverse pairs. If $f'(D)$ is any polynomial of degree ν whose roots consist of one root from each of the ν inverse pairs of $R(D)$, then $R(D)$ has the spectral factorization

$$R(D) = f'(D)f'(D^{-1}) \ . \tag{2.12}$$

By letting $f(D) = D^n f'(D)$ for any integer delay n, equation (2.12) can further be generalized to

$$R(D) = f(D)f(D^{-1}) \ . \tag{2.13}$$

$n'(D)$ in equation (2.9) is the zero–mean colored Gaussian noise with autocorrelation function $\sigma^2 R(D)$, since

$$E\{n_k' n_j'\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} E\{n(t)n(\tau)\}h(t-kT)h(\tau-jT)dtd\tau$$

$$= \sigma^2 R_{k-j} \tag{2.14}$$

where σ^2 is the spectral density of the white noise $n(t)$ and $E\{n(t)n(\tau)\} = \sigma^2\delta(t-\tau)$.

If $n(D)$ is the zero–mean white Gaussian noise with autocorrelation function σ^2, then the colored Gaussian noise $n'(D)$ can be expressed as

$$n'(D) = n(D)f(D^{-1}) \tag{2.15}$$

since $n'(D)$ has the autocorrelation function $\sigma^2 f(D^{-1})f(D) = \sigma^2 R(D)$. The autocorrelation function entirely specifies the zero–mean Gaussian noise.

By combining equations (2.9), (2.13) and (2.15), the D–transform of the output sequence of the matched–filter $h(-t)$ can be expressed as

$$a(D) = x(D)f(D)f(D^{-1}) + n(D)f(D^{-1}) \ . \tag{2.16}$$

If

$$z(D) \triangleq \frac{a(D)}{f(D^{-1})} \ , \tag{2.17}$$

then

$$z(D) = x(D)f(D) + n(D) \tag{2.18}$$

where $n(D)$ is the zero–mean white Gaussian noise and $z(D)$ is defined as the D–transform of the sampled output sequence $\{ z_k \}$ of the cascade of a matched filter $h(-t)$ with a transversal filter $1/f(D^{-1})$ as shown in figure 2.3. The cascade in figure 2.3 is called a whitened matched filter $w(-t)$ since the noise component $n(D)$ of the output sequence is whitened with a constant spectral density of σ^2.

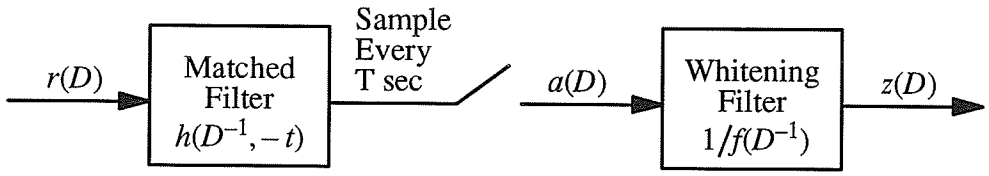


Figure 2.3: Whitened matched filter $w(-t)$.

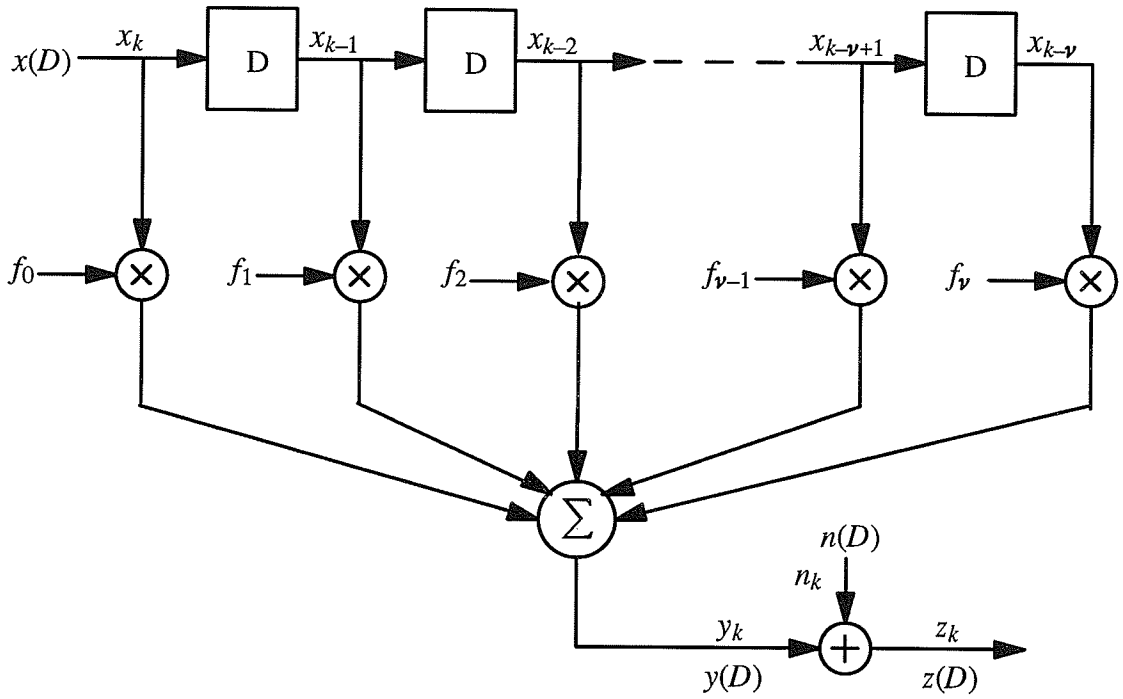


Figure 2.4: Finite–state machine model.

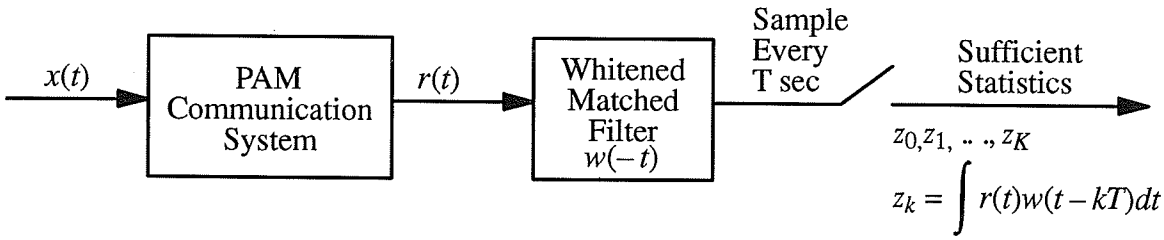


Figure 2.5: Equivalent channel model of the finite–state machine model.

More generally, for any spectral factorization of the form $R(D) = f(D)f(D^{-1})$, the filter $w(t)$ with a chip D–transform

$$w(D,t) \triangleq \frac{1}{f(D)} h(D,t) \tag{2.19}$$

is well defined and its time reversal $w(-t)$ can be used as a whitened matched filter. Hence the sampled outputs z_k of $w(-t)$

$$z_k \triangleq \int_{-\infty}^{\infty} r(t)w(t-kT)dt \tag{2.20}$$

satisfy equation (2.18) and thus form a set of sufficient statistics for the maximum–likelihood estimation of the input sequence $\{ x_k \}$ in the presence of ISI and zero mean white Gaussian noise .

However for an arbitrary spectral factorization of $R(D)$, the causality of the whitening filter $1/f(D^{-1})$ is not guaranteed. To actually realize $w(-t)$, the factorization of $R(D)$ has to be such that the whitening filter $1/f(D^{-1})$ is stable and causal, and preferably real.

In equation (2.18), the signal sequence

$$y(D) \triangleq x(D)f(D) \tag{2.21}$$

is the convolution of the input sequence $x(D)$ with the finite channel impulse response $f(D)$, and the received sequence $z(D)$ is the sum of the signal sequence $y(D)$ and a white Gaussian noise sequence $n(D)$. Thus, the relationship in equation (2.18) can be modeled as a finite–state machine (FSM) as shown in figure 2.4. An equivalent communication channel model is depicted in figure 2.5. The finite–state machine may be imagined as having a shift register of ν memory elements that store the ν most recent inputs x_{k-i}, $1 \leq i \leq \nu$. The

signal y_k may be taken as the weighted sum of the ν most recent inputs and the weighted current input x_k.

In the time domain the outputs of the finite–state machine can be expressed as

$$z_k = y_k + n_k \ , \tag{2.22}$$

where n_k is a zero mean white Gaussian noise with spectral density σ^2, and

$$y_k \triangleq \sum_{i=0}^{i=\nu} f_i x_{k-i} \ , \tag{2.23}$$

where f_i , $i = 0, 1, \ldots, \nu$ are the coefficients of $f(D)$. Note that equation (2.23) reflects the effect of ISI on the output signal.

The n_k are statistically independent zero mean Gaussian random variable with variance σ^2. Consequently, for a given input sequence $\{x_k\}$, z_k is (conditionally) statistically independent Gaussian random variable with mean y_k and variance σ^2 since z_k is a linear combination of y_k and n_k.

Based on the FSM model and the statistical properties of x_k, y_k, and z_k, Forney [7] introduced the application of the Viterbi algorithm to produce the maximum–likelihood estimate of the input sequence $\{x_k\}$ in the presence of ISI.

2.2 Viterbi Algorithm (VA)

The Viterbi algorithm [8, 17] was first introduced as a decoding method for convolutional codes. It is equivalent to a dynamic programming solution to the problem of finding the shortest path through a weighted graph. The Viterbi algorithm searches through a structure called trellis for the code word that gives the largest value of a log–likelihood function called path metric. The output of the Viterbi–decoder is always the code word that gives the largest path metric, thus it is in fact a maximum–likelihood decoding algorithm. Since the channel memory in an ISI channel is analogous to the encoder memory in a convolutional code the Viterbi algorithm can be used to produce the maximum–likelihood estimate of the sequence transmitted over an ISI channel.

On the basis of the finite–state machine model as depicted in figure 2.4, the one–to–one mapping relationship between the input sequence $\{ x_k \}$ and the signal sequence $\{ y_k \}$ in equation 2.23 can be described by a trellis. A trellis contains information of all the possible state sequences. Each node corresponds to a distinct state at a given time, and each branch represents a state transition at the next instant of time. For any ISI channel, the channel memory ν determines the number of states that exist in the trellis. Each possible state sequence in the trellis corresponds to a possible input sequence transmitted over the ISI channel and is represented by a unique path through the trellis. In the general case of a finite ISI channel with ν interference terms and m input alphabets, there are m^{ν} states in the trellis and m branches entering and leaving each state/node. The trellis of an ISI channel with binary input and a channel memory of $\nu = 2$ is shown in figure 2.6 as an example.

Associated with each branch is a branch metric which determines the likelihood of the occurrence of the state transition represented by the branch. The accumulation of the branch metrics of a particular path forms the path metric which determines the likelihood of the occurrence of the input sequence represented by the path. At the terminal node of the trellis, the path with the largest path metric is the maximum–likelihood path. Thus, to find the

maximum–likelihood path through the trellis, the Viterbi algorithm has to visit all the possible paths in order to compute and compare the path metrics.

During the decoding process, the Viterbi algorithm computes and compares the metrics of the m paths entering each of the m^{ν} states at each decoding step. The path with the largest path metric at each state is called the survivor. The fixed number of computations and the regular decoding procedure make a Viterbi processor very easy to implement. However, the number of states is m^{ν} and thus the computational complexity grows exponentially with the length of the channel memory ν. For ISI channels with large or infinite memory, the implementation of the maximum–likelihood sequence estimator using the Viterbi approach becomes impractical.

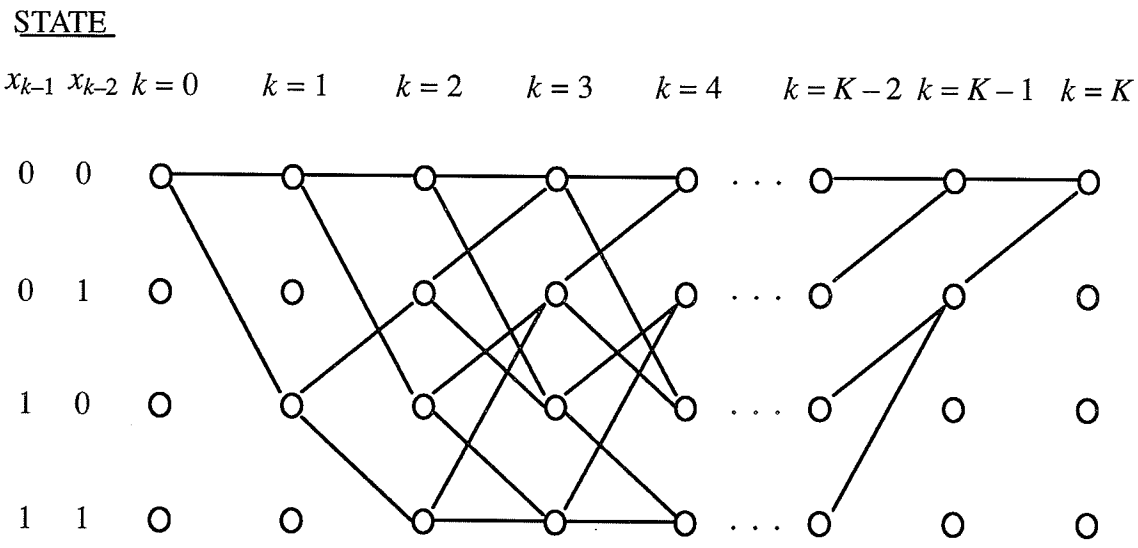


Figure 2.6: A four–state trellis.

2.3 Sequential Algorithm (SA)

The sequential algorithm [17] was introduced by Wozencraft as the first practical decoding method for convolutional codes. Fano introduced a new version of sequential decoding, subsequently referred to as the Fano algorithm [17]. Later, Zigangirov and Jelinek discovered independently another version of sequential decoding initially called ZJ algorithm but now commonly known as the stack algorithm [17]. In 1989, Xiong [26] developed the application of the sequential algorithm to the sequence estimation for ISI channels.

In the sequential decoding, the input and output relationship of the finite–state machine model is represented as paths through a code tree. Figure 2.7 shows an example of a code tree for an ISI channel with binary inputs and a channel memory of $\nu = 2$. Each node in the code tree represents a path through part of the tree, and each possible input sequence $\{ x_k \}$ transmitted over the ISI channel is represented by a unique path through the code tree. The purpose of a sequential decoding algorithm is to search through the nodes of the code tree in an efficient way so as to find the maximum–likelihood path. As in the Viterbi algorithm, whether or not a particular path is likely to be part of the maximum–likelihood path depends on the metric value associated with that path.

The most important difference between the Viterbi algorithm and the sequential decoding algorithms is that during the decoding process the Viterbi algorithm examines all the nodes in the trellis while the sequential decoding algorithms examine only a number of the nodes in the code tree. Thus, the computational complexity of the sequential algorithms does not grow exponentially with the channel memory ν as for the case of the Viterbi algorithm, but is essentially independent of the channel memory ν. This property makes sequential decoding algorithms applicable to ISI channels with large or even infinite memory. The number of nodes visited by the sequential decoding algorithms is determined by the noise level of the ISI channel. Moreover, for a given ISI channel the error probability of sequential

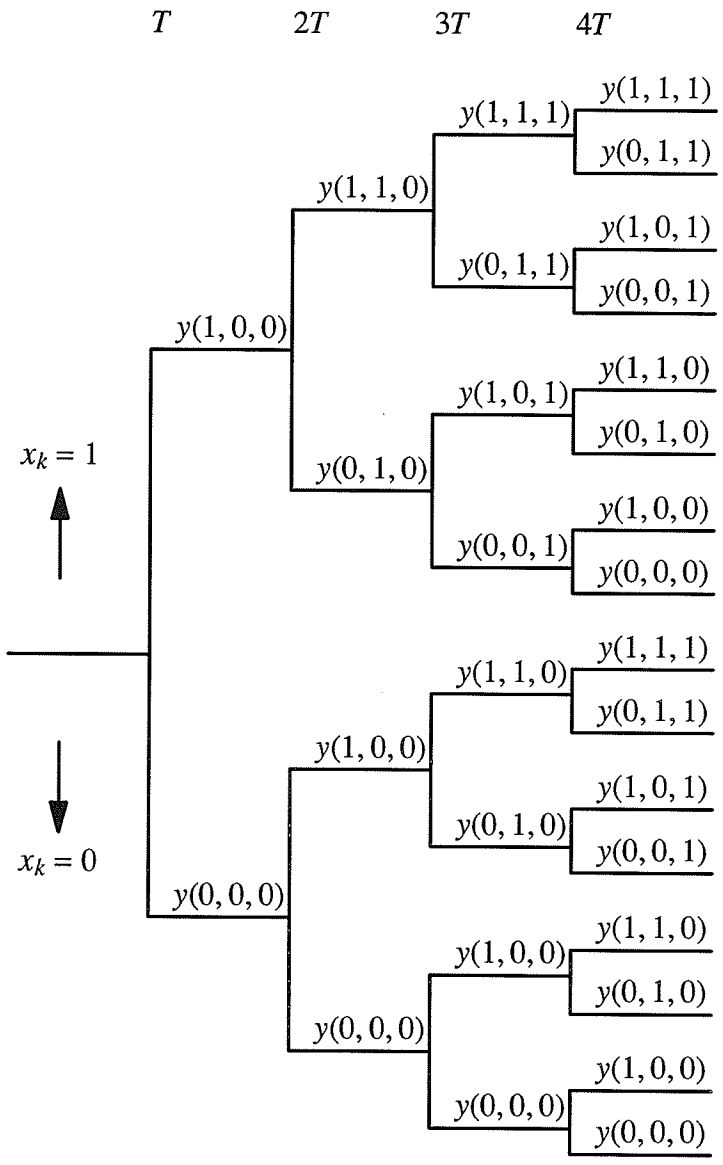


Figure 2.7: Code tree of a finite state machine model for an ISI channel with binary inputs and a channel memory of $\nu = 2$. Note: $y_k = y(x_k, x_{k-1}, x_{k-2})$.

decoding is essentially the same as for Viterbi decoding.

In this research, only the stack algorithm is considered. A very time consuming step known as the complete stack reordering in the conventional sequential stack algorithm is shown to be not fully needed. The stack reordering procedure of the stack algorithm is then modified so that the stack can be efficiently implemented by a very special type of systolic array called the systolic priority queue as proposed by Chang and Yao [3]. The details of systolic priority queue will be discussed in the next section. The Fano algorithm is not considered in this research since it has an irregular decoding structure which makes the Fano decoder unsuitable for the parallel and pipeline processing characteristics of the systolic array implementation.

2.3.1 Stack Algorithm

In the stack algorithm, a memory structure called the stack is required to store the previously examined paths of the code tree. Each stack entry holds a path along with its associated metric. The path with the largest metric is placed at the top of stack while the others are placed in a descending order of their associated metrics. The basic idea of the stack algorithm is to move forward along the path with the largest metric until the end of the code tree is reached. Each decoding step consists of extending the path at the top of the stack by computing the branch metrics of its m succeeding branches, and then adding these m branch metrics to the metric of the top path to form the m path metrics for the m successors of the top path. The top path is then deleted from the stack and the m succeeding paths are inserted into the stack. All of the paths in the stack are then rearranged in a descending order of their associated metric values so that the path with the largest metric is at the top of the list. The decoding steps are repeated until the top path reaches the end of the code tree. The top path is then taken as the decoded path and the algorithm terminates. The stack algorithm is summarized as follows.

The stack algorithm:

Step 1) Load the stack with the origin node in the code tree, whose metric is taken to be zero.

Step 2) Compute the metric of the m successors of the top path in the stack.

Step 3) Delete the top path from the stack.

Step 4) Insert the m new paths in the stack, and rearrange the stack in a order of descending metric values.

Step 5) If the top path in the stack reaches the end of the code tree, stop. Otherwise, return to step 2.

A complete flowchart for the stack algorithm is shown in figure 2.8.

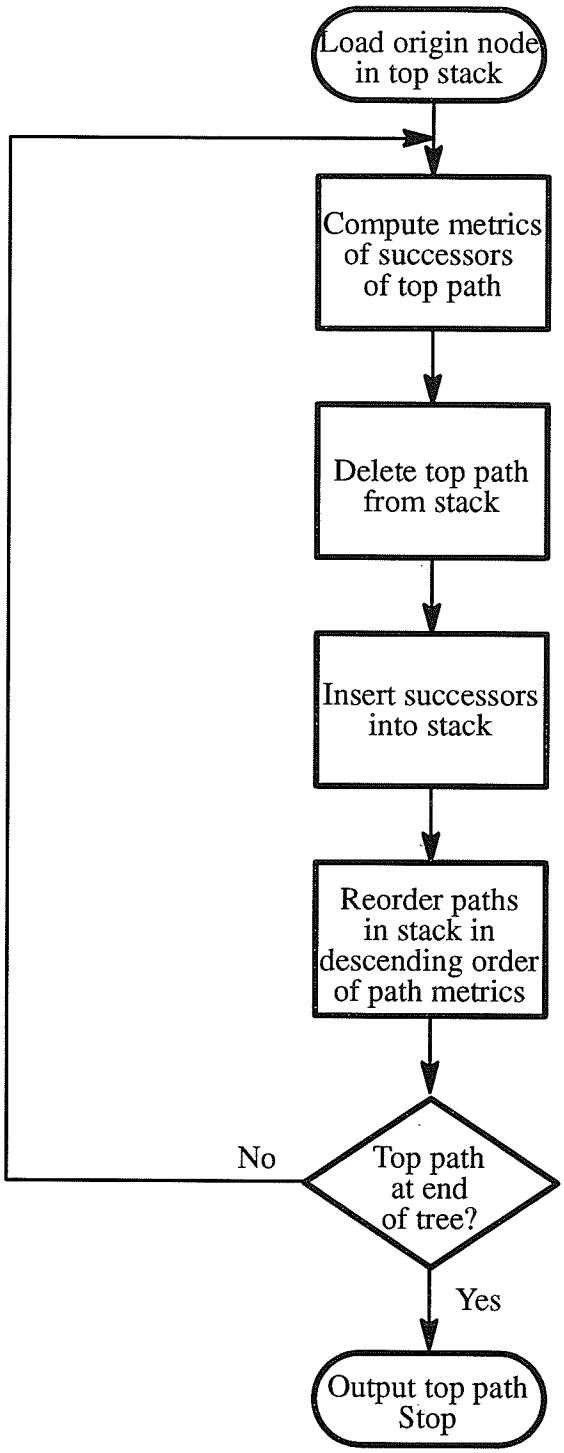


Figure 2.8: Flowchart of the conventional sequential stack algorithm.

2.3.2 Problems with the Practical Implementation of Stack Algorithm

There are three practical problems associated with the implementation of the stack algorithm. The first problem is input buffer overflow which results in a loss of data, or an erasure. A sequential stack decoder has to search for the maximum–likelihood path by tracing back and forth from node to node through the code tree, an input buffer must be present to store the incoming received data while they are waiting to be processed. In the case of a very noisy channel, the decoder may have to perform long searches so as to find the current best path without using any received data held in the input buffer. Under such a circumstance, the received data in the input buffer will accumulate which eventually leads to an overflow of the input buffer. When an input buffer overflows, incoming received data will force undecoded received data to be shifted out of the buffer. These bits are then lost which results in an erasure.

The second problem with the stack algorithm is stack overflow. In any practical implementation of the stack algorithm, the number of entries in the stack has to be finite. For a channel with m input alphabets, m paths are inserted into the stack while only one path is deleted from the stack in each decoding step. There is always some probability that the stack will fill up before decoding is completed, especially for the case of a noisy channel. The most common way of handling this problem is to allow the path at the bottom of the stack to be pushed out of the stack on the next decoding step. If the stack size is large enough, the probability that a path at the bottom of the stack would recover to reach the top of the stack and be extended is very small and the loss in performance due to stack overflow is negligible.

The third problem with the stack algorithm is the complete stack reordering of the paths in a descending order of their associated metric values after each decoding step. The complete reordering of the stack is not only time consuming but also dependent upon the number of paths exists in the stack. It is undesirable for the decoding time to be dependent upon the number of paths in the stack since this number increases by $m-1$ after each decoding step.

The complete stack reordering can become quite time consuming as the number of paths in the stack becomes large, and places severe limitations on the decoding speed that can be achieved with the basic algorithm.

This research concentrates only on solving the complete stack reordering problem of the stack algorithm. The complete stack reordering is not fully required in the stack algorithm. The only purpose of complete stack reordering is to have the current best path placed on the top of the stack so that it is ready to be extended in the next decoding step. To achieve the goal of having the current best path placed on the top entry of the stack, it is not necessary to have all the paths arranged in a descending order. On the basis of the operational characteristics of a special type of systolic array called the systolic priority queue, the stack reordering procedure of the conventional stack algorithm is modified in such a way that the best path is placed on the top entry of the stack without having the rest of the paths arranged in a descending order of their associated metric values. No matter how many paths are in the stack, the systolic array is always able to complete its task within a fixed and short interval of time.

2.4 Systolic Priority Queues

As described in the stack algorithm, there is an associated path metric stored with each path in the stack/array. Since the location of a particular path in the array depends on the magnitude of its associated path metric, the operations of the different systolic priority queues are explained in terms of the magnitudes of the path metric m_k. Thus in the examples illustrated in this section, each entry A_i in the array is denoted by only a number m_k representing the path metric of a particular path .

The requirement for the systolic array is that it be able to place the largest metric at the top of the array so that it can be extended in the next decoding step. Extending the largest/best metrics involves the deletion of the current best metric and the insertion of m new succeeding metrics. In particular, the time involved for the above three operations must be fixed, short and independent of the number of metric values in the array and the size of the array. This type of systolic array is known as the systolic priority queue. Two general types of linear systolic priority queues are discussed. They share the linear array structure shown in figure 2.9, where A_i is a sequence of registers used to store the path metrics m_k.

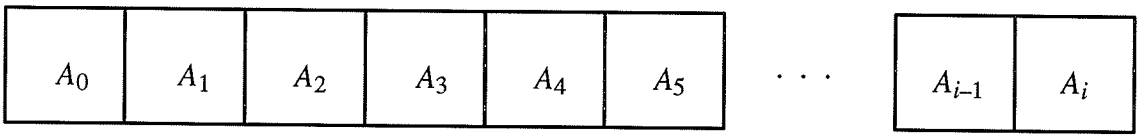


Figure 2.9: Basic structure of linear systolic priority queue.

The two types are called the Random Access Memory Systolic Priority Queue and the Shift Register Systolic Priority Queue. The primary difference between them is in the method for rearranging the order of the metrics. By implementing the stack in the stack algorithm with either one of the two linear systolic priority queues named above, the stack reordering can be carried out in parallel rather than sequentially as imposed by the conventional complete stack reordering procedure. Thus, the best path is always placed at the top of the array within a fixed and short interval of time. The Shift Register Systolic Priority Queue is used in this research for the final VLSI implementation of a sequential demodulator.

2.4.1 Random Access Memory Systolic Priority Queue (RAM–SPQ)

Figure 2.9 shows the basic structure of the RAM–SPQ, where A_i, $i = 0, 1, 2, \ldots$, is a sequence of registers for storing the metric values. Figure 2.10 shows the building unit A_i of the RAM–SPQ. Each register A_i can exchange data with its two adjacent neighbors inside the array and an external device as shown in figure 2.10.

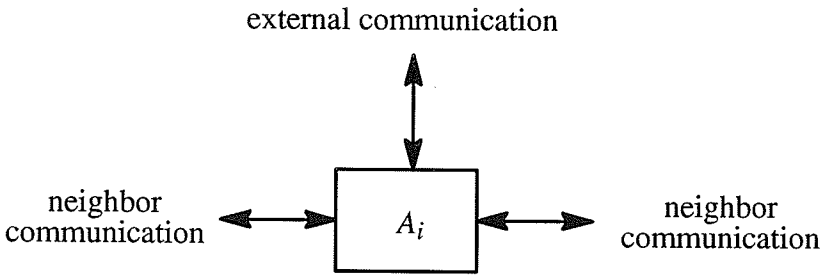


Figure 2.10: Building unit of the Random Access Memory Systolic Priority Queue.

In the random access memory scheme, each register is initialized to a path metric value of positive infinity (in practice a large positive number). A global control signal PT which serves as a pointer to locate a particular register in the array is initialized to zero (PT is pointing at register A_0).

Insertion and deletion of path metric values in this type of systolic priority queue are very similar to the push and pop operations of a stack. After each insertion or deletion, the path metric values in the queue are rearranged in a unique pairwise manner so as to ensure that the best metric is located at register A_{PT-1} (analogous to the top of the stack), where A_{PT} is the next available register. Since the metrics are reordered in pairs of two, reordering of the entire array proceeds in a parallel manner. The insertion operation takes place according to the procedure described below.

Insertion of a succeeding path metric m_k in RAM–SPQ:

1) Insert the succeeding path metric m_k into register A_{PT}, $A_{PT} \leftarrow m_k$.
2) Increment PT by one after insertion of m_k, $PT \leftarrow PT + 1$.
3) If i satisfies $(i - PT) \bmod 2 = 0$, for $i \geq 0$, then rearrange the metric values in A_i and A_{i+1} such that the metric value in A_{i+1} is greater than or equal to the metric value in A_i, $A_{i+1} \geq A_i$.

A flowchart for the insertion operation is shown in figure 2.11. An example of the insertion operation in the Random Access Memory Systolic Priority Queue is shown in figure 2.12.

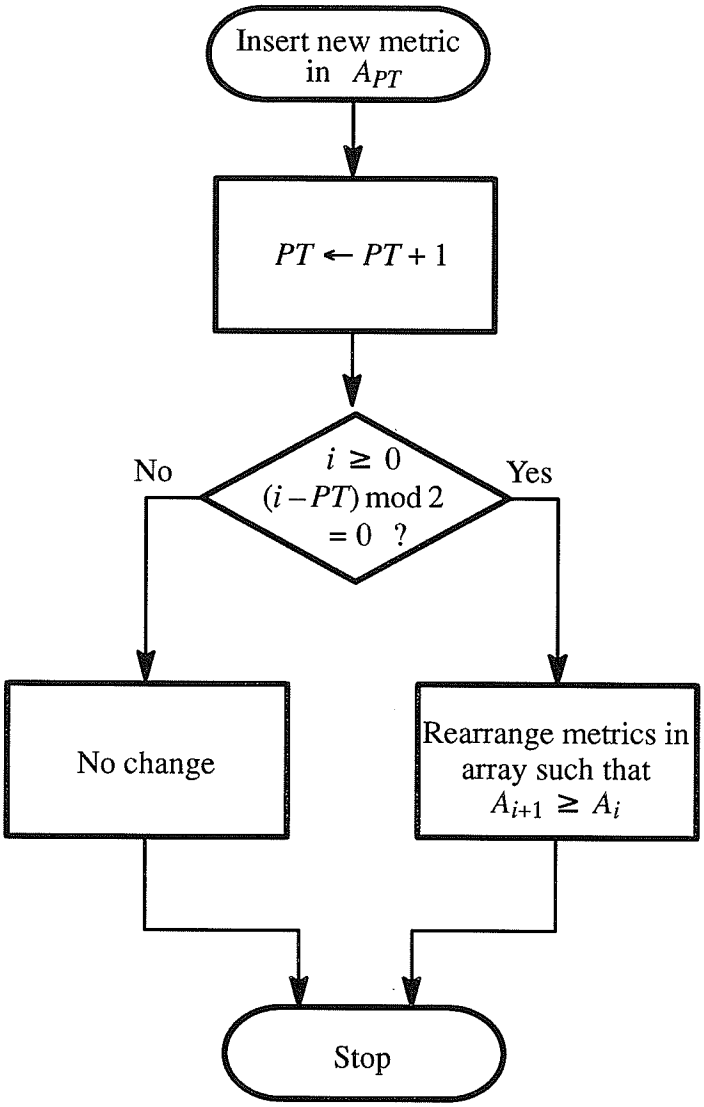


Figure 2.11: Flowchart of the insertion of a succeeding path metric in Random Access Memory Systolic Priority Queue.

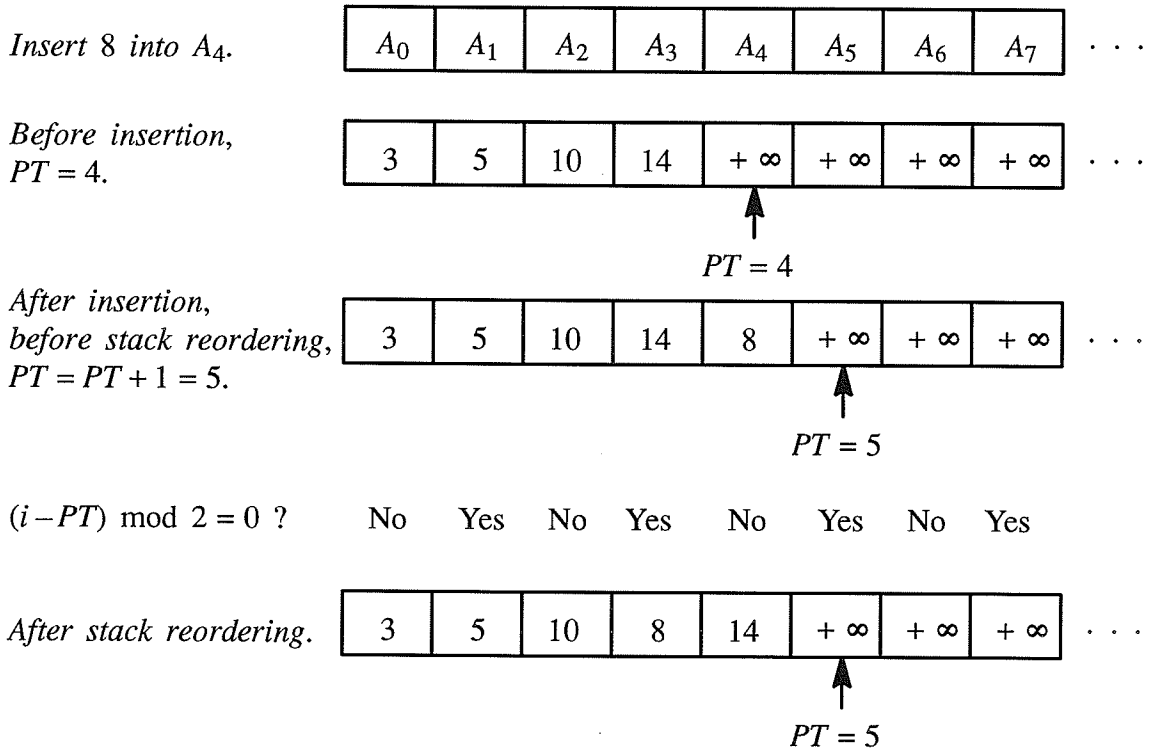


Figure 2.12: Example of the insertion operation in the Random Access Memory Systolic Priority Queue.

The deletion operation takes place according to the procedure described below.

Deletion of the largest path metric m_k in RAM–SPQ:

1) Decrement PT by one, $PT \leftarrow PT - 1$.

2) Empty A_{PT}, which contains the largest path metric m_k.

3) If i satisfies $(i - PT) \bmod 2 = 0$, for $i \geq 0$, then rearrange the metric values in A_i and A_{i+1} such that the metric value in A_{i+1} is greater than or equal to the metric value in A_i, $A_{i+1} \geq A_i$.

Figure 2.13 shows a flowchart for the deletion operation. Figure 2.14 shows an example of the deletion operation in the Random Access Memory Systolic Priority Queue.

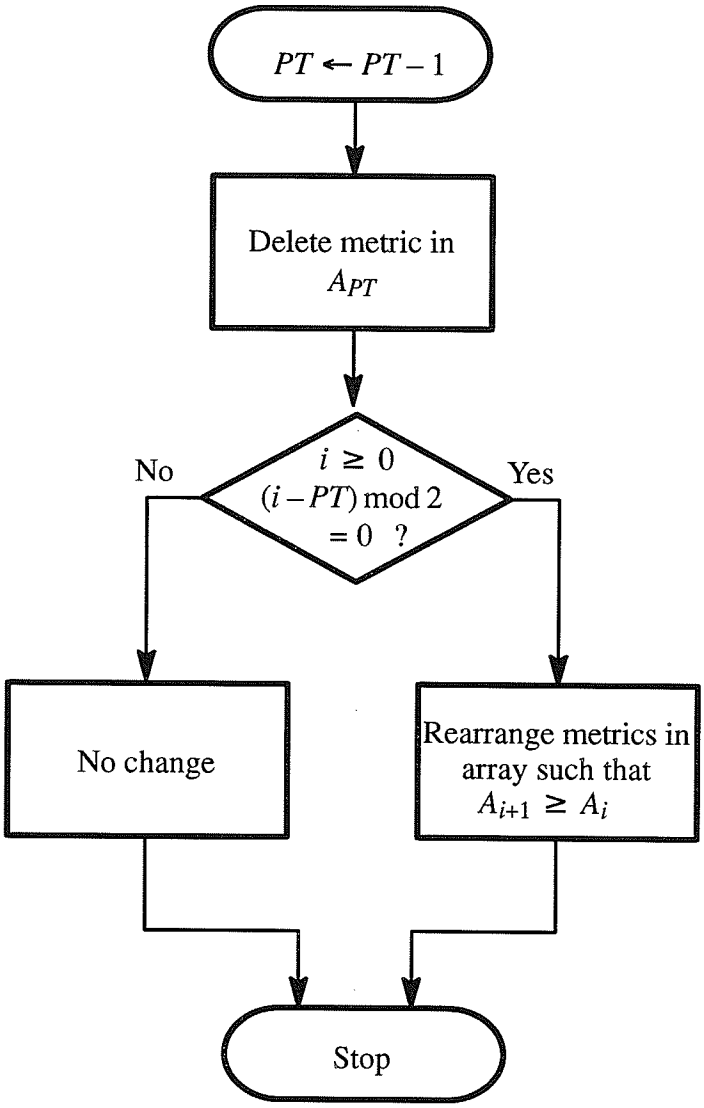


Figure 2.13: Flowchart of the deletion of the largest path metric in Random Access Memory Systolic Priority Queue.

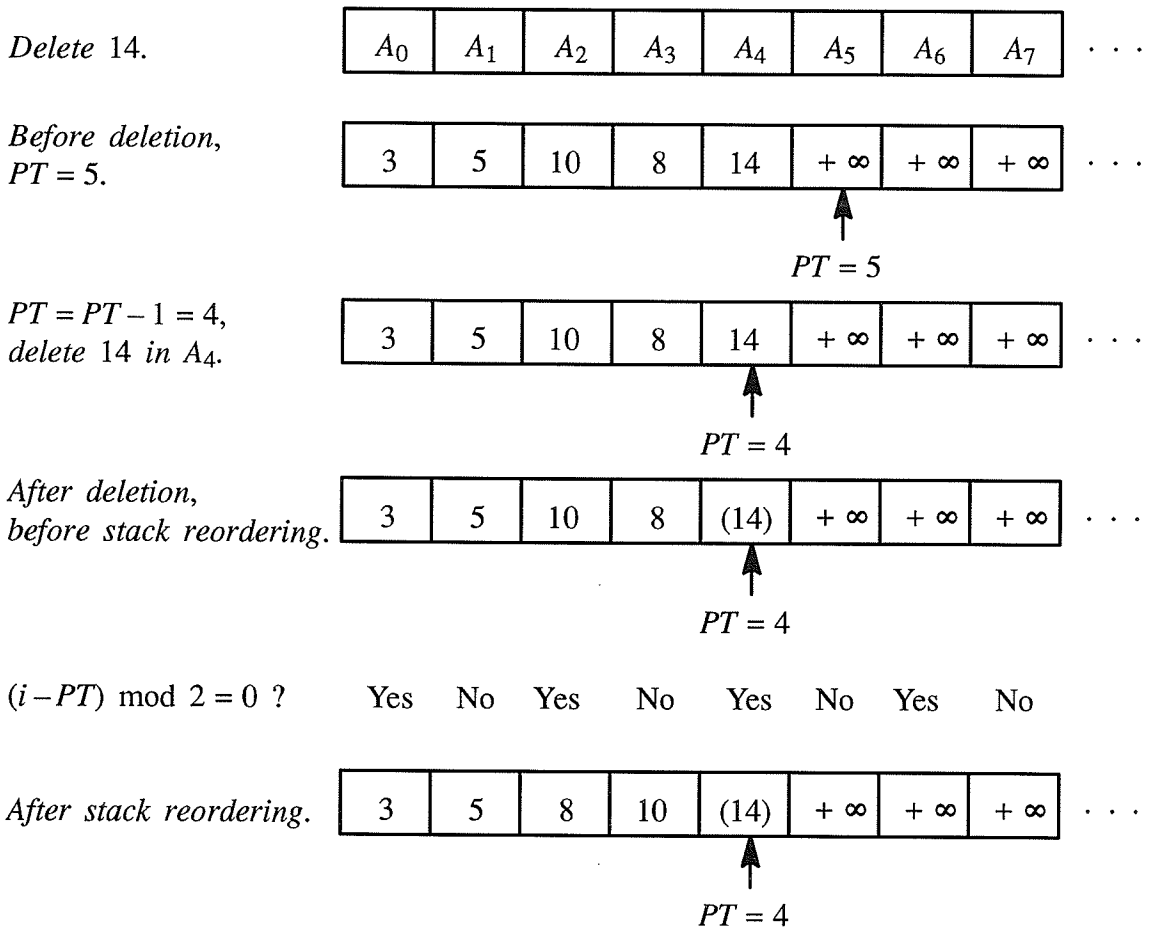


Figure 2.14: Example of the deletion operation in the Random Access Memory Systolic Priority Queue. Note: The path metric inside the parenthesis () is overwritten by the path metric in the next insertion step.

The examples shown in figure 2.12 and figure 2.14 illustrate the following properties of the Random Access Memory Systolic Priority Queue.

1). Only one control signal PT is required. PT is either incremented by one ($PT + 1$) to locate the next available register for insertion of a new metric or decremented by one ($PT - 1$) to locate the register which holds the largest metric value for deletion.

2). After each insertion or deletion, the order of metric values in the array are reordered in pairs of two adjacent registers and the stack reordering can be done in a parallel or pipeline manner. Thus, the array reordering procedure can be done in a fixed and short interval of time and the time required is independent of the number of metric values in the array.

3). After each array reordering, although the path metric values are only in a partly descending order, A_{PT-1} will always contain the largest path metric value ready for the next deletion and A_{PT} is always the next available register ready for insertion of a new metric.

With a basic understanding in the insertion, deletion and array reordering procedure of the RAM–SPQ, the application of this particular type of systolic priority queue in the stack algorithm is demonstrated by an example. Figure 2.15 shows the flowchart of the RAM scheme stack algorithm. Figure 2.16 shows an ISI tree to which the RAM scheme stack algorithm is applied. The number labeled at each node is the path metric of the path represented by that particular node. Several steps in decoding the ISI code tree shown in figure 2.16 is considered. The contents of the RAM–SPQ at each decoding step is shown in figure 2.17.

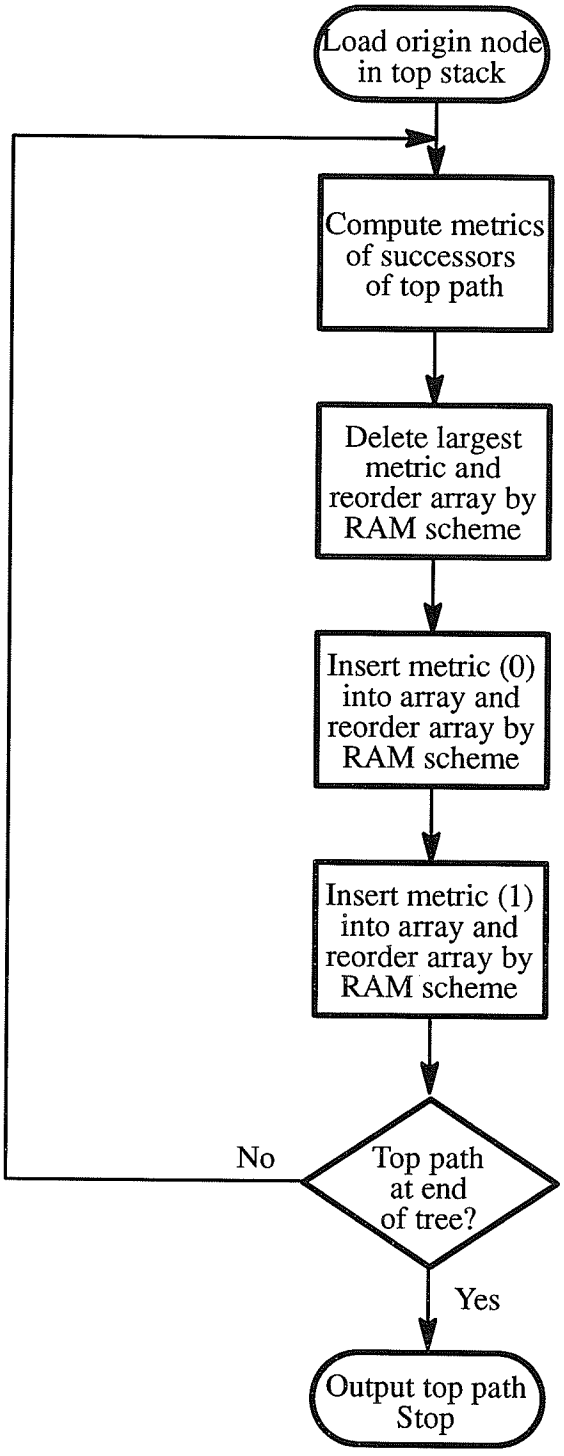


Figure 2.15: Flowchart of the RAM scheme stack algorithm for a channel with binary inputs.

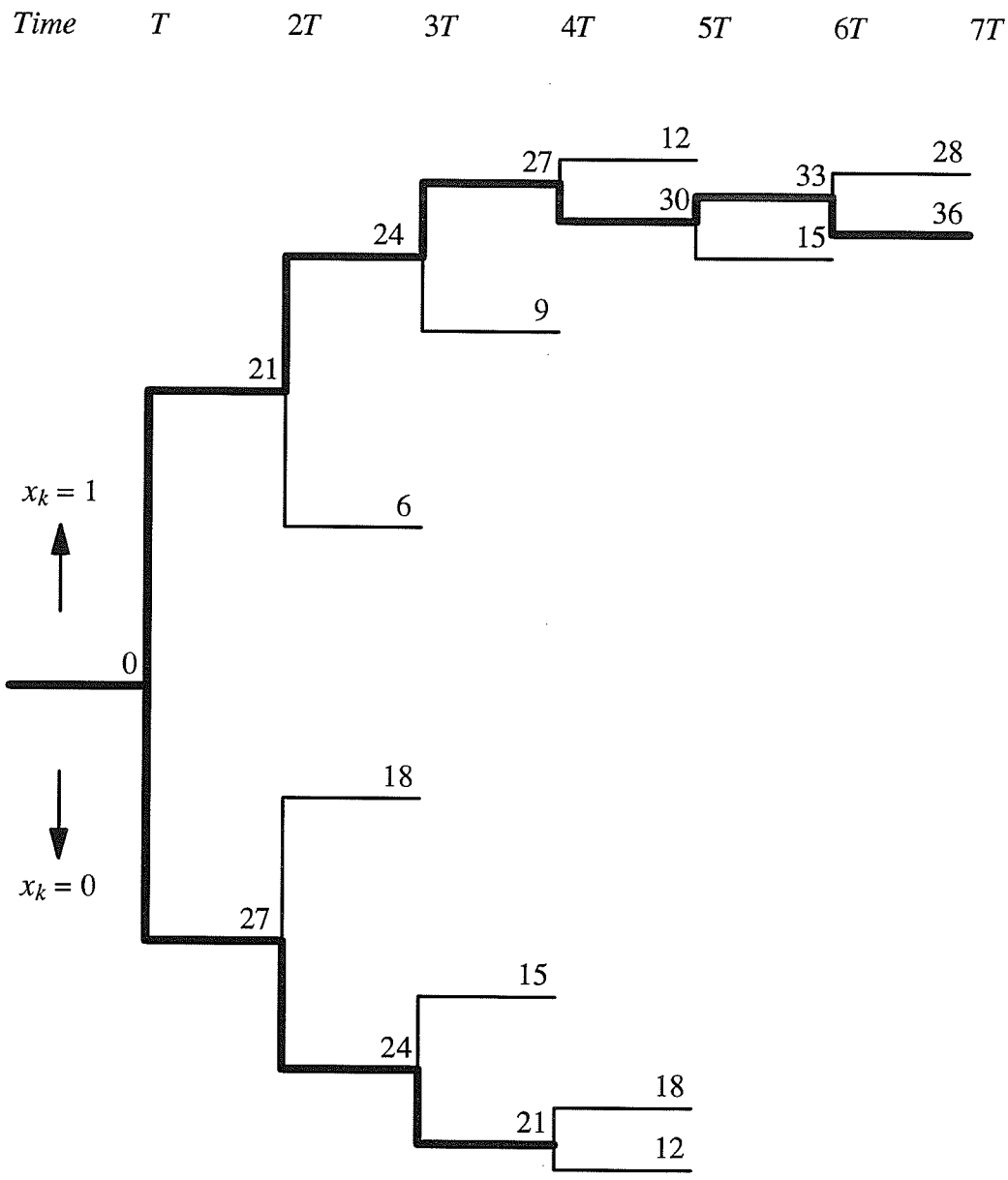


Figure 2.16: Example of a code tree for an ISI channel with binary inputs.

Step	*Operation*	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	· · ·
		+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
1	*insert* 27	27	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
	insert 21	27	21	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
2	*delete* 27	21	(27)	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
	insert 24	21	24	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
	insert 18	21	24	18	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
3	*delete* 24	21	18	(24)	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
	insert 21	18	21	21	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
	insert 15	18	21	21	15	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
4	*delete* 21	18	21	15	(21)	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
	insert 12	18	15	21	12	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
	insert 18	15	18	12	21	18	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
5	*delete* 21	15	12	18	18	(21)	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
	insert 6	12	15	18	18	6	+ ∞	+ ∞	+ ∞	+ ∞	· · ·
	insert 24	12	15	18	6	18	24	+ ∞	+ ∞	+ ∞	· · ·
6	*delete* 24	12	15	6	18	18	(24)	+ ∞	+ ∞	+ ∞	· · ·
	insert 9	12	6	15	18	18	9	+ ∞	+ ∞	+ ∞	· · ·
	insert 27	6	12	15	18	9	18	27	+ ∞	+ ∞	· · ·
7	*delete* 27	6	12	15	9	18	18	(27)	+ ∞	+ ∞	· · ·
	insert 30	6	12	9	15	18	18	30	+ ∞	+ ∞	· · ·
	insert 12	6	9	12	15	18	18	30	12	+ ∞	· · ·
8	*delete* 30	6	9	12	15	18	18	12	(30)	+ ∞	· · ·
.	.					.					
.	.					.					
.	.					.					

Figure 2.17: Array contents of the stack decoding of the ISI tree shown in figure 2.16 with Random Access Memory Systolic Priority Queue. Note: The path metric inside the parenthesis () is overwritten by the path metric in the next insertion step.

Refer to figure 2.17, the number of metric values in the array increases by one after each decoding step and the largest metric is always at the $PT-1$ position after each insertion or deletion. Most important of all, the structure of the systolic priority queue allows the parallel processing of data, and hence the array reordering procedure can be completed in a fixed and short interval of time independent of the number of metric values in the array. This feature greatly enhances the decoding speed and applicability of the stack algorithm.

2.4.2 Shift Register Systolic Priority Queue (SR–SPQ)

Figure 2.9 shows the basic structure of the SR–SPQ, where A_0 serves as an input/output port and A_i, $i = 1, 2, 3, \ldots$, is a sequence of registers for storing the metric values. Figure 2.18 shows the building units A_0 and A_i of the SR–SPQ. A_0 can exchange data with an external device and its adjacent neighbor, and each register A_i can exchange data with its two neighbors inside the array as shown in figure 2.18.

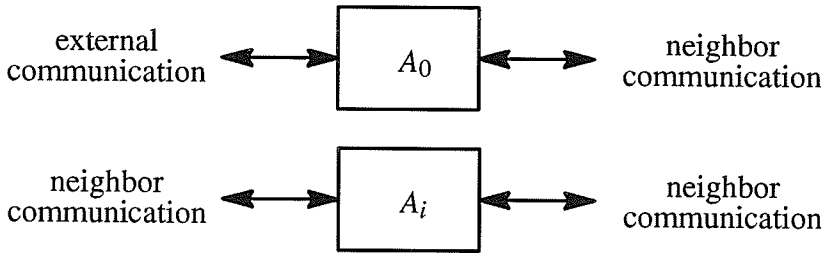


Figure 2.18: Building units of the Shift Register Systolic Priority Queue.

In the shift register scheme, each register A_i (for $i = 1, 2, 3, \ldots$) is initialized to a path metric value of negative infinity (in practice a large negative number). Two global control signals are required to control the shift right and shift left operations of the path metric values in the array. At the end of this sub–section, a modified version of the insertion and deletion operations shows that only one global control signal is required to control shift right operation of the path metric values in the array. Insertion and deletion operations can only take place at A_0 since register A_0 serves as an input/output device.

After each insertion or deletion, the path metric values in the array are rearranged in a unique pairwise manner so as to ensure that the largest path metric is located at register A_1 (analogous to the top of the stack). As in the Random Access Memory Systolic Priority Queue, the pairwise rearrangement of metrics allows reordering of the entire array to proceed in a parallel manner. The procedure for the insertion operation in the Shift Register Systolic Priority Queue is described below.

Insertion of a succeeding path metric m_k in SR–SPQ:

1) Insert the succeeding path metric m_k into register A_0, $A_0 \leftarrow m_k$.
2) Shift all the path metric values m_k in the array one position to the right, $A_{i+1} \leftarrow A_i$.
3) Rearrange the metric values in A_{2i+1} and A_{2i+2}, for $i \geq 0$, such that the metric value in A_{2i+1} is greater than or equal to the metric value in A_{2i+2}, $A_{2i+1} \geq A_{2i+2}$.

A flowchart for the insertion operation is illustrated in figure 2.19. An example of the insertion operation in the Shift Register Systolic Priority Queue is shown in figure 2.20.

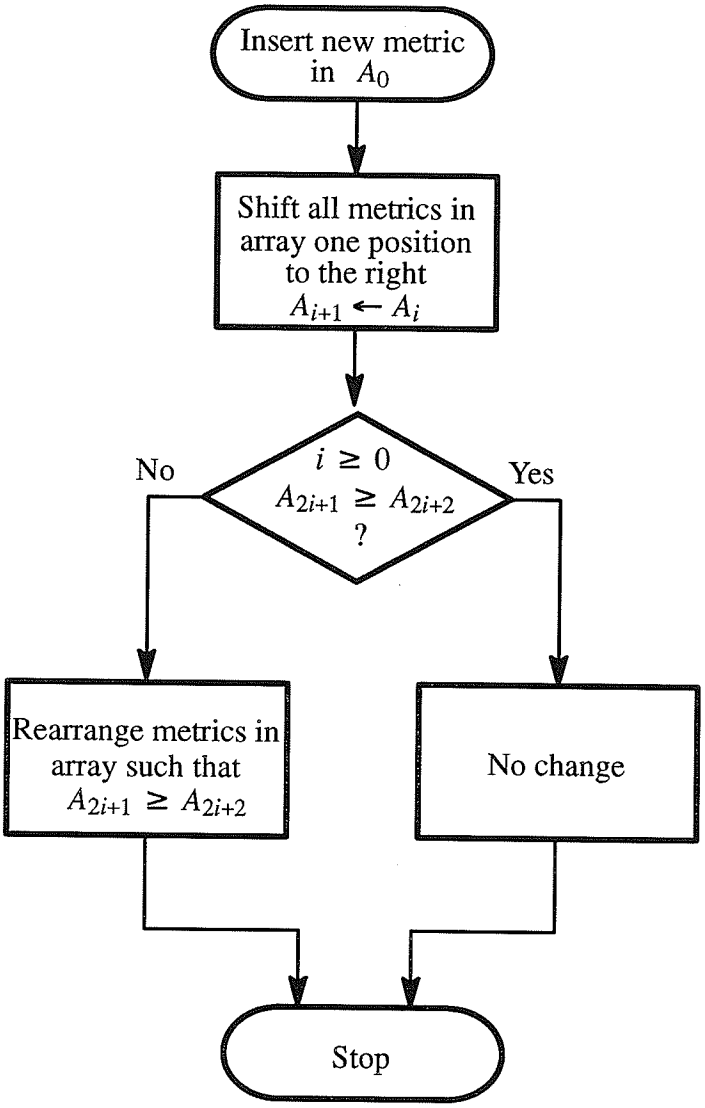


Figure 2.19: Flowchart of the insertion of a succeeding path metric in Shift Register Systolic Priority Queue.

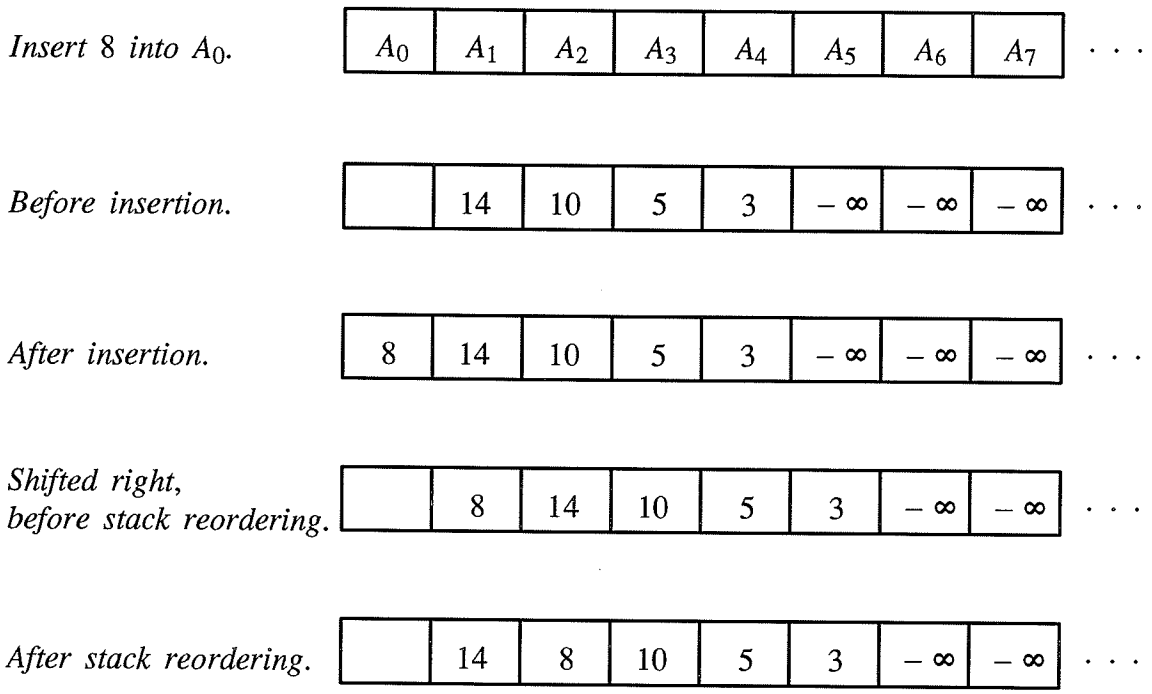


Figure 2.20: Example of the insertion operation in the Shift Register Systolic Priority Queue.

The procedure for the deletion operation in the Shift Register Systolic Priority Queue is described below.

Deletion of the largest path metric m_k in SR–SPQ:

1) Shift all the path metric values m_k in the array one position to the left, $A_i \leftarrow A_{i+1}$.
2) Empty A_0, which contains the largest path metric.
3) Rearrange the metric values in A_{2i+1} and A_{2i+2}, for $i \geq 0$, such that the metric value in A_{2i+1} is greater than or equal to the metric value in A_{2i+2}, $A_{2i+1} \geq A_{2i+2}$.

A flowchart for the deletion operation is illustrated in figure 2.21. An example of the deletion operation in the Shift Register Systolic Priority Queue is shown in figure 2.22.

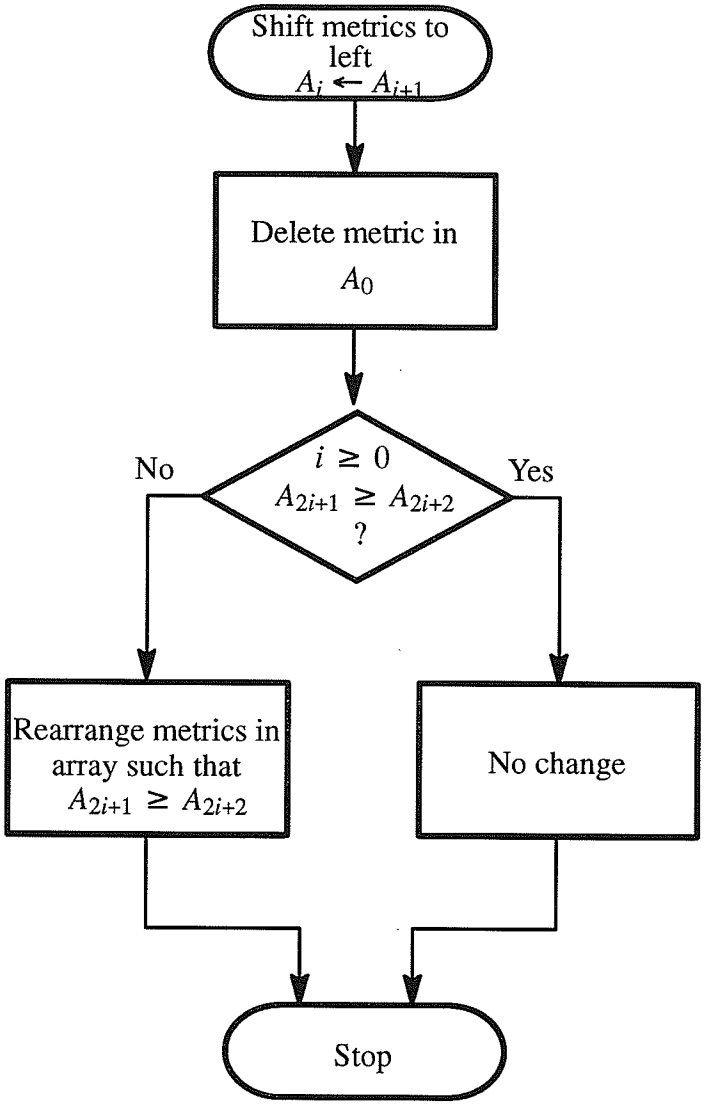


Figure 2.21: Flowchart of the deletion of the largest path metric in Shift Register Systolic Priority Queue.

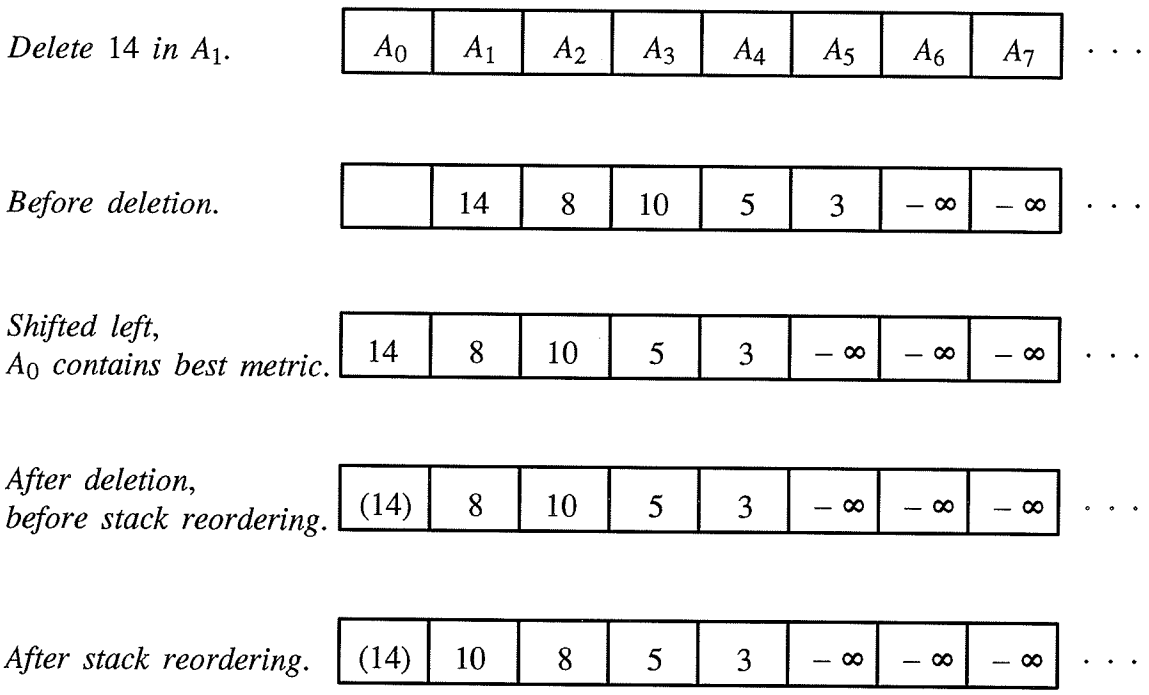


Figure 2.22: Example of the deletion operation in the Shift Register Systolic Priority Queue. Note: The path metric inside the parenthesis () is overwritten by the path metric in the next insertion step.

The examples shown in figure 2.20 and figure 2.22 illustrate the following properties of the Shift Register Systolic Priority Queue.

1). A right shift global signal is required to shift the contents of the array one position to the right after each insertion. A left shift global signal is required to shift the contents of the array one position to the left before each deletion so as to place the largest metric in register A_0 for deletion.

2). The insertion and deletion operations are followed by pairwise reorderings of the path metric values in the array. The pairwise reorderings are to shuffle the path metric values in A_{2i+1} and A_{2i+2} so that the values in A_{2i+1} are greater than or equal to the values in A_{2i+2}, for $i = 0, 1, 2, \ldots$. Since the reordering procedure occurs in pairs of two adjacent registers in the $2i + 1$ and $2i + 2$ positions, the reordering of the metric values in the entire array can take place in a parallel manner (in the same time interval).

3). After each array reordering, although the path metric values are only in a partly descending order, the largest path metric value is always located at register A_1.

The application of the SR–SPQ in the stack algorithm is demonstrated through the same example, i.e., the ISI code tree of figure 2.16. Figure 2.23 shows the flowchart of the SR scheme stack algorithm. Figure 2.24 shows the contents of the SR–SPQ at each decoding steps.

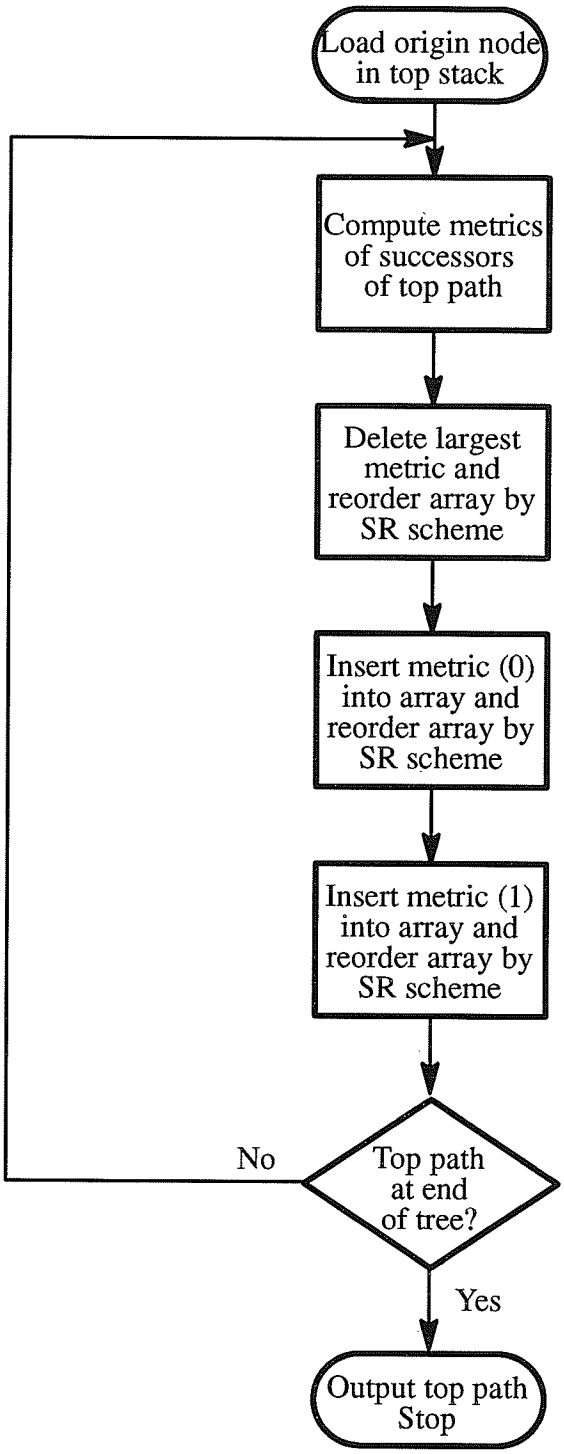


Figure 2.23: Flowchart of the SR scheme stack algorithm for a channel with binary inputs.

Step	*Operation*	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	...
			$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
1	*insert* 27	27	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
	insert 21	21	27	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
2	*delete* 27	(27)	21	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
	insert 24	24	21	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
	insert 18	18	24	21	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
3	*delete* 24	(24)	18	21	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
	insert 21	21	21	18	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
	insert 15	15	21	21	18	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
4	*delete* 21	(21)	15	21	18	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
	insert 12	12	21	15	18	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
	insert 18	18	21	12	18	15	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
5	*delete* 21	(21)	18	18	12	15	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
	insert 6	6	18	18	15	12	$-\infty$	$-\infty$	$-\infty$	$-\infty$	...
	insert 24	24	18	6	18	15	12	$-\infty$	$-\infty$	$-\infty$	...
6	*delete* 24	(24)	18	18	6	15	12	$-\infty$	$-\infty$	$-\infty$	...
	insert 9	9	18	18	15	6	12	$-\infty$	$-\infty$	$-\infty$	...
	insert 27	27	18	9	18	15	12	6	$-\infty$	$-\infty$	...
7	*delete* 27	(27)	18	18	9	15	12	6	$-\infty$	$-\infty$	...
	insert 30	30	18	18	15	9	12	6	$-\infty$	$-\infty$	...
	insert 12	12	30	18	18	15	12	9	6	$-\infty$	...
8	*delete* 30	(30)	12	18	18	15	12	9	6	$-\infty$	...
.	.					.					
.	.					.					
.	.					.					

Figure 2.24: Array contents of the stack decoding of the ISI tree shown in figure 2.16 with Shift Register Systolic Priority Queue. Note: The path metric inside the parenthesis () is overwritten by the path metric in the next insertion step.

Refer to figure 2.24, the largest metric is always placed at register A_1 after each insertion or deletion and the number of metric values in the array increases by one after each decoding step. As in the Random Access Memory Systolic Priority Queue, the array reordering procedure only takes a fixed and short interval of time regardless of the number of values in the array since the structure of the systolic priority queue allows the parallel processing of data. Compared to the conventional stack algorithm, the time saved in the array/stack reordering procedure by using the SR Systolic Priority Queue greatly enhances the decoding speed and applicability of the stack algorithm. The decoding speed of the SR scheme stack algorithm can be further improved as outlined by the following discussion.

In the stack algorithm, the deletion of the largest metric always proceeds after the insertion of the last succeeding path metric in the previous decoding step. A faster operational speed is achieved by combining the two operations mentioned above. The structure of the SR Systolic Priority Queue allows the insertion and deletion operations to take place simultaneously. The procedure for performing insertion and deletion simultaneously is described as follows.

Insertion of a succeeding path metric m_j and deletion of the largest path metric m_k in the SR Systolic Priority Queue:

1) Insert the succeeding path metric m_j into register A_0, $A_0 \leftarrow m_j$.
2) Rearrange the metric values in A_{2i} and A_{2i+1}, for $i \geq 0$, such that the metric value in A_{2i} is greater than or equal to the metric value in A_{2i+1}, $A_{2i} \geq A_{2i+1}$.
3) Empty A_0, which contains the largest path metric.
4) Rearrange the metric values in A_{2i+1} and A_{2i+2}, for $i \geq 0$, such that the metric value in A_{2i+1} is greater than or equal to the metric value in A_{2i+2}, $A_{2i+1} \geq A_{2i+2}$.

Figure 2.25 shows the flowchart where insertion and deletion operations are performed simultaneously. Figure 2.26 shows an example for the above combined operations.

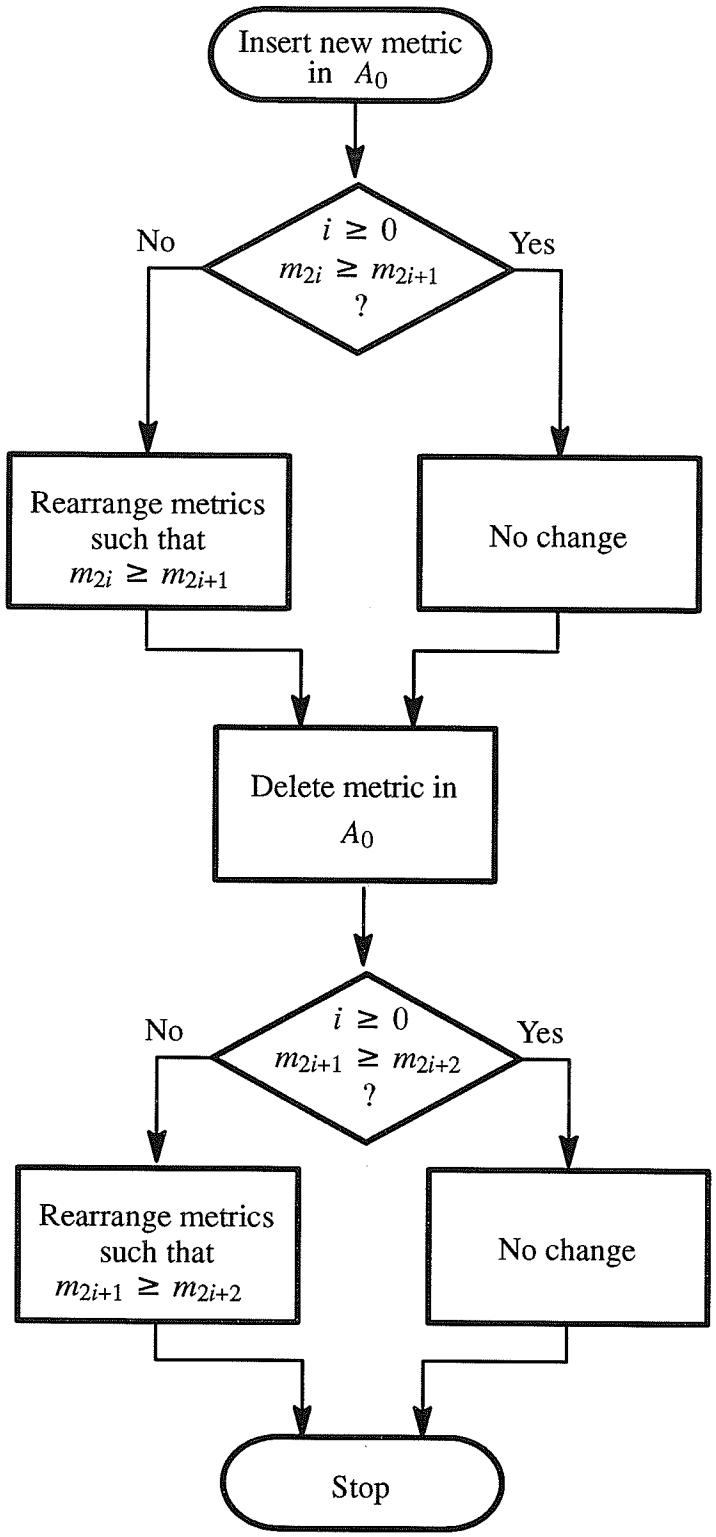


Figure 2.25: Flowchart of simultaneous insertion of a succeeding path metric and deletion of the largest path metric in SR Systolic Priority Queue.

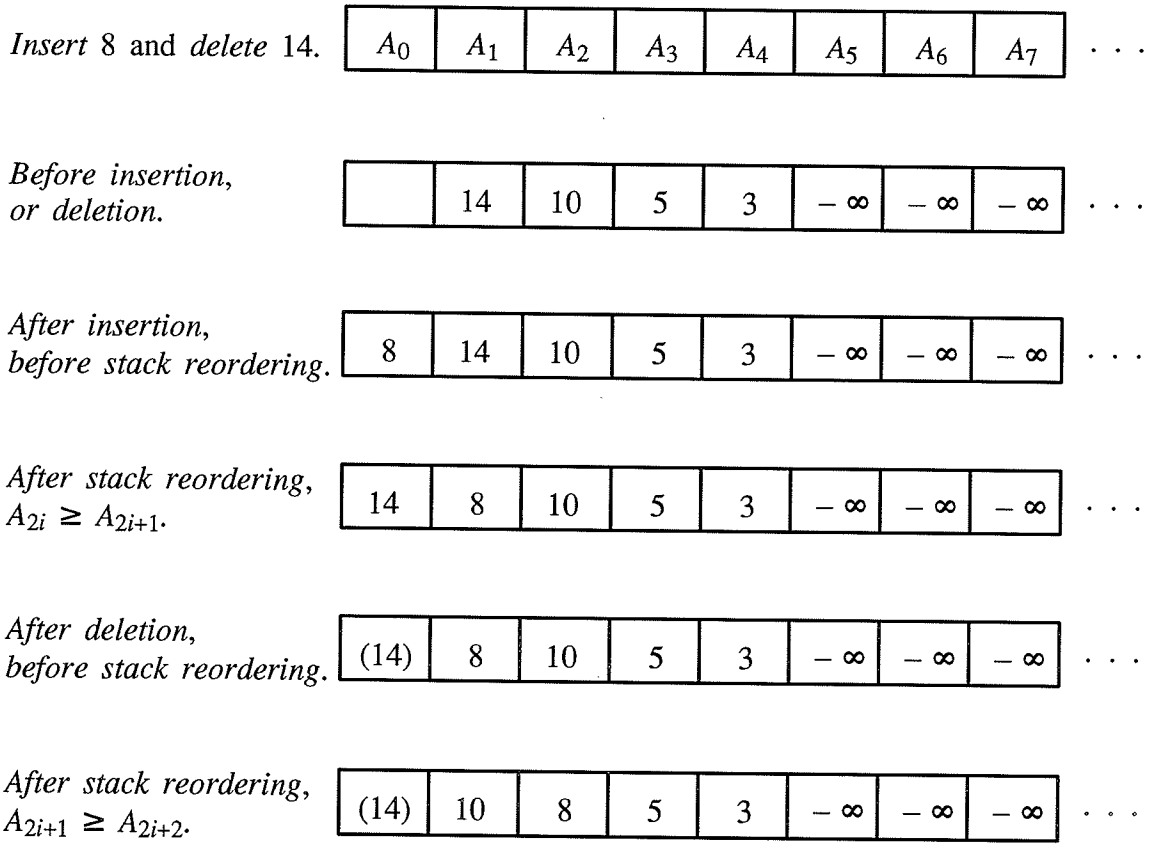


Figure 2.26: Example of the simultaneous insertion and deletion operations in Shift Register Systolic Priority Queue. Note: The path metric inside the parenthesis () is overwritten by the path metric in the next insertion step.

In the practical implementation of the stack algorithm, the actual removal of the largest path metric from the top of the stack is not required. In all the discussions concerning the deletion operation of the two types of systolic priority queues mentioned before, a step for emptying the register holding the largest metric is only included for the easier understanding of the deletion operation. When the insertion of the first succeeding path metric takes place in the next decoding step, the content of the top stack will simply be written over by the newly inserted values. Thus, the only important fact is to have the current largest path metric placed on top of the stack at the end of each decoding step.

Based on the fact mentioned above, the simultaneous insertion and deletion operation of the SR Systolic Priority Queue can be used in the stack algorithm for inserting the last succeeding path metric in each decoding step so that the largest path metric is ready for deletion in the next decoding step. For a channel with binary inputs, one deletion operation and two insertion operations are required in each decoding step. By using the simultaneous insertion and deletion operation described above, only one more insertion operation is required for a channel with binary inputs. Since the time unit required for the simultaneous operation is the same as the time unit required for the insertion operation, thus the time units required for the deletion operation is completely eliminated.

Finally, since the left shift control signal is only required in the deletion operation, elimination of the deletion operation means the elimination of the left shift control signal. Thus, only one control signal is required for the Shift Register Systolic Priority Queue. Figure 2.27 shows the flowchart of the modified version of the Shift Register stack algorithm. This modified version is used in the design and development of the sequential demodulator described in Chapter 3.

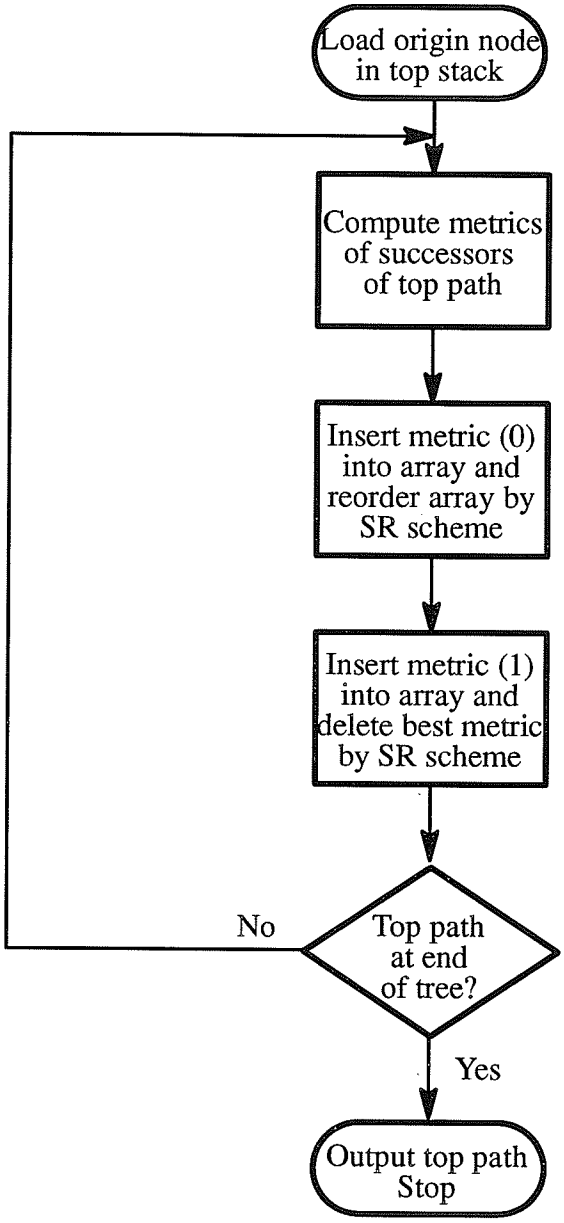


Figure 2.27: Flowchart of the modified version of the SR scheme stack algorithm for a channel with binary inputs.

2.4.3 Concluding Remarks on Systolic Priority Queues

The implementation of the stack algorithm by systolic priority queues alleviates the stack reordering problem in the conventional stack algorithm. By using the systolic priority queue, the stack reordering procedure can be completed in a fixed and short time independent of the number of path metric values in the stack. Both of the RAM and SR Systolic Priority Queues have a very satisfactory speed in performing the required operations.

In the RAM Systolic Priority Queue, each register needs an external input/output connection. While in the SR Systolic Priority Queue, only the register A_0 at the leftmost end needs an external input/output connection. From the hardware implementation point of view, a lot more circuitry is required in the RAM Systolic Priority Queue to locate the desired register to do the input/output operation at each decoding step. In the SR Systolic Priority Queue, the input/output operation only takes place at register A_0 and thus no extra circuitry is required to determine the location of the desire register. Moreover, with the simultaneous operation in the SR scheme, the number of global control signals in the SR scheme is reduced to one which is the same as in the RAM scheme.

On the basis of the above, the SR Systolic Priority Queue is superior to the RAM Systolic Priority Queue for the hardware implementation of a sequential demodulator.

Chapter 3 Design of the Sequential Demodulator

3.1 General Description of the Sequential Demodulator

The receiver structure for the maximum–likelihood estimation of digital sequences in the presence of ISI consists of a whitened matched filter, a symbol–rate sampler, an analog–to–digital (A/D) convertor and a sequential demodulator. To generate the sufficient statistics required for maximum–likelihood estimation, the output from an ISI channel is first filtered by the whitened matched filter and then sampled by the symbol–rate sampler. The sample is quantized by an A/D convertor to a level suitable for input to the sequential demodulator.

This research concentrates only on the design of the sequential demodulator. The circuitry of the whitened matched filter, the symbol–rate sampler and the analog–to–digital (A/D) convertor is assumed to exist. In this chapter, the storage configuration, operation and design of a sequential demodulator applicable to ISI channels is discussed in some detail. The design of the sequential demodulator is based on a Shift Register Systolic Priority Queue architecture. The overall block diagram of the sequential demodulator is shown in figure 3.1. It consists of seven custom–made ASIC VLSI chips and eight standard memory ICs. Figure 3.1 depicts the connections between blocks of the demodulator and direction of signal flow.

General design parameters for the demodulator are that it can handle a block length of 256 bits and a maximum of 9 interference terms. The maximum allowable stack size of the design is 1024. An 8–bit representation is used for the sufficient statistic z_k and a 16–bit representation for the metric.

Details of the sequential demodulator are discussed in the subsequent sections. Section 3.2 describes the input and output storage configuration of the design. Section 3.3 demonstrates

the operation of the design through a specific example. Section 3.4 discusses the details of the hardware design.

3.2 Storage Configuration

In this section, the storage format of the sequential demodulator's input and output are discussed. The input consists of a sufficient statistic look–up table and a branch metric look–up table. The output consists of path metric values, associated addresses, 9–previous information bits, path lengths and explored paths. Figure 3.1 shows the storage locations of the corresponding input and output. The storage format of the input look–up tables is first described. Then the output storage format is illustrated through an example.

In the design, the sufficient statistic z_k is assumed to be available as a form of data stored in memory. The output of the whitened matched filter is sampled every T seconds and the sampled output is then quantized to one of 256 different levels using an 8–bit analog–to–digital (A/D) converter. The quantized values are stored in chronological order as 8–bit numbers in RAM7. RAM7 thus acts as a sufficient statistic look–up table. In general, address add_i of RAM7 holds sufficient statistic z_{i+1} received at time $(i+1)T$. For example, if the number of symbols in the explored path (011) is three, then, to extend the path, the sufficient statistic z_4 received at time $4T$ is required. The above shows that the path length or the number of symbols in the explored path can be used as an address to retrieve the desired sufficient statistic from RAM7.

The second input to the sequential demodulator is the branch metric look–up table. Since the sufficient statistic z_k is represented by an 8–bit number then for any ISI channel with ν interference terms, there are only 2^{L+8}, where $L = \nu + 1$, possible combinations of the L–previous information bits and the sufficient statistic z_k. To each combination of the $L+8$ bits, there is a particular value for the branch metric. All of the 2^{L+8} possible branch metric values of a particular ISI channel are precalculated and the values are stored in a

EPROM. The corresponding values of the bit patterns (2^{L+8} possible combinations) are used as addresses for the branch metric values in the EPROM. The EPROM thus acts as a look–up table for the branch metric calculation as shown in figure 3.1.

The output storage format of the sequential demodulator is illustrated through the specific example shown in figure 3.2. Figure 3.2 shows the relationship between a sample code tree and the contents of SPQ–MU, SPQ–AU, RAM1, RAM2 and RAM3–6. To each path metric stored in the SPQ–MU, there is a associated address stored in the corresponding location in the SPQ–AU. The associated address provides the locations of the 9–previous information bits, the path lengths and the explored paths in corresponding RAMs. For example, m_5 is the current largest path metric and add_2 is its associated address since they are located in the register A_0 of the SPQ–MU and SPQ–AU respectively.

As can be seen from the code tree shown in figure 3.2, the explored path which corresponds to the node labeled m_5 is (010). With the assumption that all the previous inputs are '0', the 9–previous information bits of the explored path (010) is (000000010). Moreover, since there are three symbols in the path (010), the path length is equal to three or (00000011) in an 8–bit binary representation. Figure 3.2 shows that the 9–previous information bits (000000010), the path length (00000011) and the explored path (010) are stored in add_2 of RAM1, RAM2 and RAM3–6 respectively. Finally, note that the 9–previous information bits and the path length of an explored path are stored as one word in address add_i of RAM1 and RAM2 respectively while the explored path is stored as eight consecutive words in address block add_i of RAM3–6.

Based on this basic understanding of the general features and storage structure of the sequential demodulator, the details of the design are discussed in the next section.

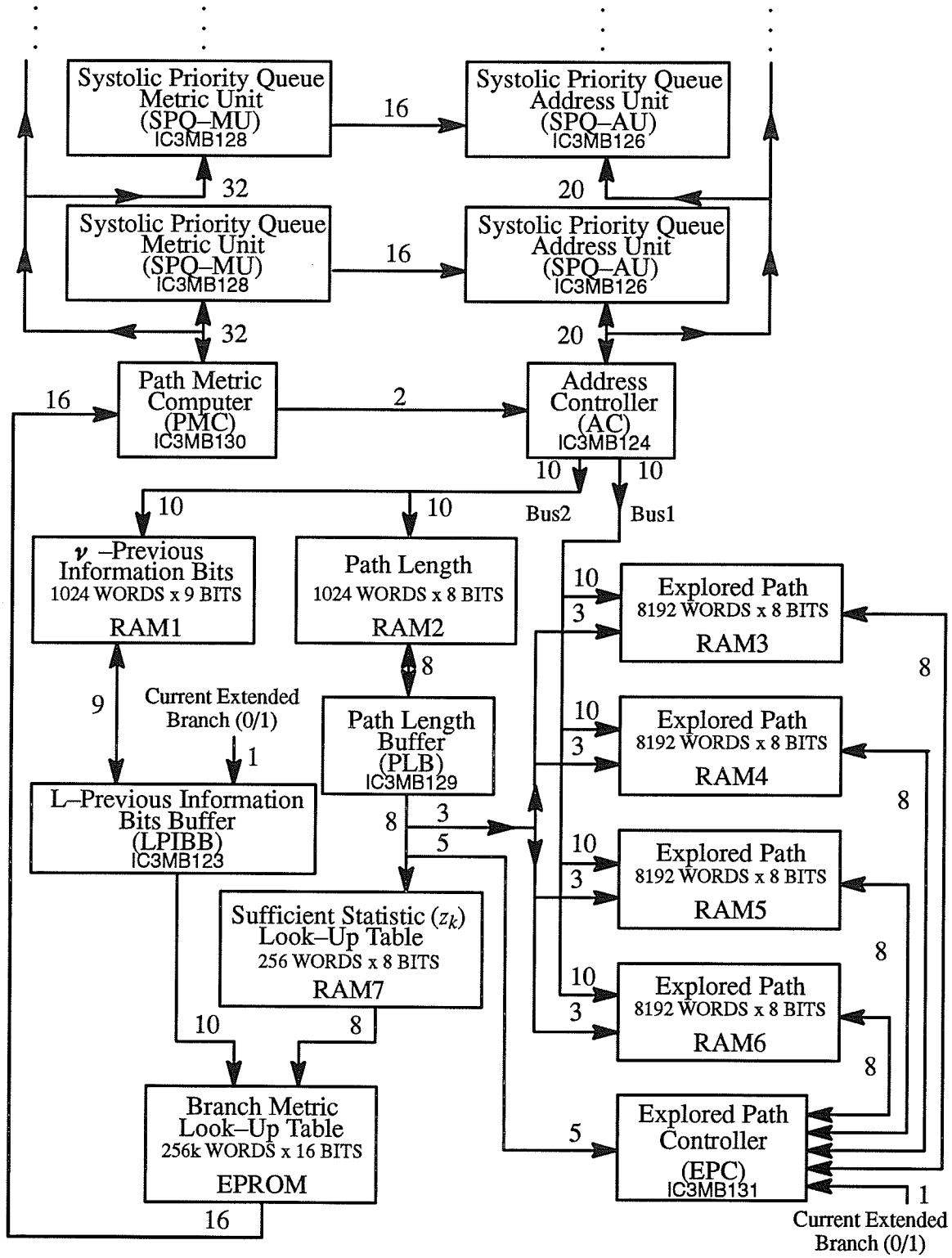
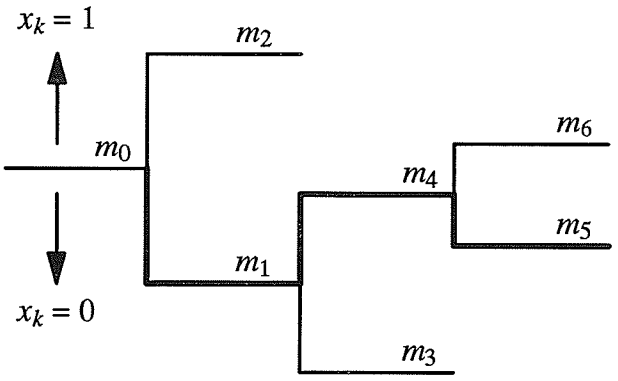


Figure 3.1: Block diagram of the sequential demodulator based on a systolic priority queue architecture. Note: The number on the side of each arrow indicates the number of data lines represented by that particular arrow.



SPQ–MU	m_5	m_6	m_3	m_2	$-\infty$	$-\infty$	$-\infty$	$-\infty$	. . .
SPQ–AU	add_2	add_3	add_0	add_1	0	0	0	0	. . .

ν –Previous Information Bits
RAM1

add_0	– – – – – – – – – 0
add_1	– – – – – – – – – 1
add_2	– – – – – – – 0 1 0
add_3	– – – – – – – 0 1 1
add_4	next available address

Path Length
RAM2

add_0	0 0 0 0 0 0 0 1 0
add_1	0 0 0 0 0 0 0 0 1
add_2	0 0 0 0 0 0 0 1 1
add_3	0 0 0 0 0 0 0 1 1
add_4	next available address

Explored Path

	RAM3	RAM4	RAM5	RAM6
add_0	0 0			
add_1	1			
add_2	0 1 0			
add_3	0 1 1			
add_4		next available address		

Figure 3.2: The relation of a sample code tree with the systolic priority queue contents and the stack contents. Note: Each add_i in RAM3–6 is an address block of 8 words and each word consists of 32 bits.

3.3 Operation of the Sequential Demodulator

This section explains the extension of an explored path by the sequential demodulator implementation. After outlining the general steps, the details of each step are illustrated through a specific example, namely, that of extending the explored path (010) in the code tree of figure 3.2.

There are seven steps involved in extending an explored path in the binary code tree. The first step sets up the RAMs. Steps 2–4 extend the path associated with branch (0) while steps 5–7 extend the path associated with branch (1). The steps are:

1). Duplicate data in the associated address (of the current largest path metric) to the next available address of RAM1, RAM2 and RAM3–6.

2). Calculate the metric of the extended path associated with branch (0).

3). Insert the path metric and the address of the extended path associated with branch (0) into the systolic priority queues.

4). Store the extended path, its 9–previous information bits and its path length in RAM3–6, RAM1 and RAM2 respectively.

5) to 7). Repeat 2) to 4) for the extended path associated with branch (1).

To explain these steps in detail, consider figure 3.2. Metric m_5, corresponding to explored path (010), is the current largest path metric with associated address add_2. The next available address in the RAMs is add_4. The first step in extending an explored path is that of duplication of data from address add_l (i.e. associated address of the largest path metric) to address add_t (i.e. the next available address) of RAM1, RAM2 and RAM3–6.

3.3.1 Duplication of data in RAM1, RAM2 and RAM3–6

Before considering the details of the duplication process, the memory arrangement of RAM3–6 and the address control of RAM1, RAM2 and RAM3–6 are described.

RAM3–6 can be viewed as one RAM with a 32–bit data bus and a 13–bit address bus. Since a maximum path length of 256 bits is chosen in the design, RAM3–6 are partitioned into blocks of eight words where each word contains 32 bits. The maximum available stack size is thus 1024. Figure 3.3 shows the storage arrangement of an explored path within a block in RAM3–6.

10 MSB	One Block RAM3	RAM4	RAM5	RAM6	
.....000	bit [1,8]	bit [9,16]	bit [17,24]	bit [25,32]	32 bit word ←
.....001	bit [33,40]	bit [41,48]	bit [49,56]	bit [57,64]	
.....010	bit [65,72]	bit [73,80]	bit [81,88]	bit [89,96]	
.....011	bit [97,104]	bit [105,112]	bit [113,120]	bit [121,128]	
.....100	bit [129,136]	bit [137,144]	bit [145,152]	bit [153,160]	
.....101	bit [161,168]	bit [169,176]	bit [177,184]	bit [185,192]	
.....110	bit [193,200]	bit [201,208]	bit [209,216]	bit [217,224]	
.....111	bit [225,232]	bit [233,240]	bit [241,248]	bit [249,256]	

Figure 3.3: Explored path storage configuration. Note: ..... represents the block address add_i and is controlled by Bus1 of the Address Controller. The 3 LSB are controlled by the Path Length Buffer.

The address of each 32–bit word is specified by a unique 13–bit pattern. The ten MSB (add_i) determine the block location within RAM3–6 and the three LSB (000 to 111) determine the word location within the block.

The addresses of RAM1, RAM2 and RAM3–6 are controlled by two ASIC's, the Address Controller and the Path Length Buffer. RAM3–6 addresses are controlled by Bus1 of the Address Controller and the three MSB from the Path Length Buffer. Within the Path Length Buffer, there is a 3–bit counter which controls its three most significant output lines during the duplication process. The output of the 3–bit counter (000 to 111) is held while Bus1 output changes from add_i to add_t allowing each of the eight consecutive 32–bit words of

an explored path to be duplicated. Once a word is duplicated, Bus1 goes back to add_l. This is repeated eight times. Similar to Bus1, RAM1 and RAM2 address lines are controlled by Bus2 of the Address Controller. Bus2 changes from add_l to add_t to duplicate the 9–previous information bits and the path length of an explored path. However, since only one word needs to be duplicated, Bus2 only switches once.

Having described how the addresses of the RAMs are controlled, the actual duplication process is illustrated through an example of extending the explored path (010) in the code tree of figure 3.2. The data in address add_2 of RAM1 and RAM2 are read and then written into address add_4 of RAM1 and RAM2 by the L–Previous Information Bits Buffer and Path Length Buffer respectively. At the same instant, each of the eight consecutive words in block add_2 is read and then written into block add_4 of RAM3–6 by the Explored Path Controller.

As soon as the duplication process in RAM1 and RAM2 is completed, the Address Controller switches Bus2 back to add_2. Then the sequential demodulator uses the data in add_2 of RAM1 and RAM2 to calculate the path metric of the extended path (0100).

3.3.2 Path extension

Continuing the example, the next step is to extend path (010) to (0100) and (0101). Refer to figure 3.4. Path metric m_7 for the extended path (0100) is calculated by retrieving the branch metric of branch (0) from the EPROM and adding it to path metric m_5.

To retrieve the branch metric, the 9–previous information bits (000000010) of the explored path (010) are obtained from add_2 of RAM1 and combined with the extended branch (0) to form the 10–bit sequence (0000000100). This 10–bit sequence is used to control the ten most significant address lines of the EPROM. The eight least significant address lines of the EPROM are determined by the appropriate sufficient statistic z_k which is determined by the depth in the tree or the path length. Thus the Path Length Buffer retrieves the path length (00000011) of the explored path from add_2 of RAM2 and uses it as an address to retrieve

z_k. The 16–bit branch metric from the EPROM is then passed to the Path Metric Computer for the calculation of m_7 for the extended path (0100). In the Path Metric Computer, the 16–bit adder adds the branch metric from the EPROM to the path metric m_5 from register A_0 of the metric systolic priority queue.

Having determined the path metric m_7, all data relevant to the extended path (0100) have to be stored. The data consist of m_7, the associated address add_2, the extended path, the 9–previous information bits and the path length of the extended path.

Path metric m_7 and its associated address add_2 of the extended path (0100) are inserted into register A_0 of the metric systolic priority queue and the address systolic priority queue respectively by the procedure *Insertion of a succeeding path metric in SR Systolic Priority Queue* (pp. 36–38). Since add_2 is now the associated address of m_7, the data in address add_2 of RAM1, RAM2 and RAM3–6 have to be updated so that they are in correspondence to m_7.

The extended path (0100) is stored in RAM3–6 prior to updating the path length of the explored path (010). To do this, the ten MSB of RAM3–6 address are set to add_2. The three LSB, since data duplication has been completed, are now addressed by the three MSB of the 8–bit path length of the explored path (010). This determines the address of the word within the block. The Explored Path Controller then uses the five LSB of the 8–bit path length of the explored path (010) to determine the bit location within the 32–bit word for inserting the extended branch (0) into RAM3–6. Thus the extended path is stored as (0100) in address add_2 of RAM3–6.

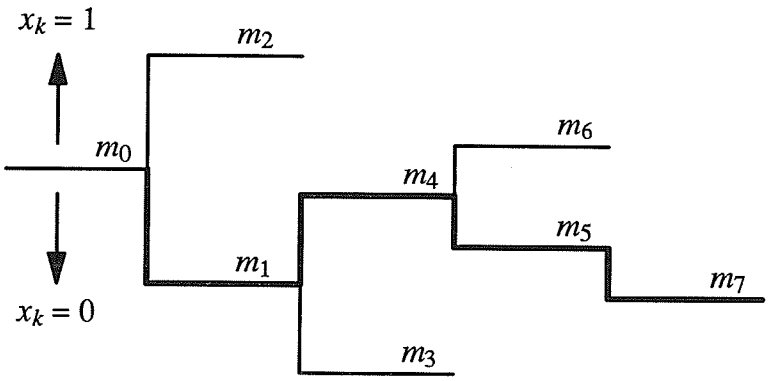
The new 9–previous information bits (000000100) is stored in address add_2 of RAM1. The L–Previous Information Bits Buffer shifts out the most previous bit of the sequence (**0**000000100) to form the new sequence (000000100) which is the 9–previous information bits corresponding to the extended path (0010).

Finally the length of the extended path is updated by having the Path Length Buffer increment the path length (00000011) of the explored path (010) by one. The new path length (00000100) is stored in address add_2 of RAM2.

Extension of branch (0) of the explored path (010) is now complete. Figure 3.4 shows the contents of the code tree, SPQ–MU, SPQ–AU, RAM1, RAM2 and RAM3–6 after the above operations with the assumption that $m_7 < m_6$. The Address Controller switches Bus1 and Bus2 to add_4 in order to extend branch (1). RAM1, RAM2, and RAM3–6 are now addressed by add_4 until the beginning of the next extension step.

The extension procedures for branch (1) of the explored path (010) are similar to those for branch (0). The differences are: 1) data in address add_4 of RAM1, RAM2 and RAM3–6 are used and updated, 2) path metric m_8 and address add_4 are inserted into register A_0 of the SPQ–MU and the SPQ–AU respectively, note that, by a different procedure. It is described on (pp. 45–47) under the heading *Insertion of a succeeding path metric and deletion of the largest path metric in the SR Systolic Priority Queue.*

Figure 3.5 shows the contents of the code tree, SPQ–MU, SPQ–AU, RAM1, RAM2 and RAM3–6 after a complete step of extending the explored path (010). The description of the 7 steps involved in extending an explored path is complete. For example, if the extension of the code tree is to be continued, with the assumption that $m_7 < m_6$ and $m_8 < m_6$, then the explored path (011) is extended. The seven described steps are repeated. In the next section, a discussion of the details in the hardware design of the sequential demodulator is given.



SPQ–MU		m_6	m_7	m_3	m_2	$-\infty$	$-\infty$	$-\infty$
SPQ–AU		add_3	add_2	add_0	add_1	0	0	0

ν –Previous Information Bits
RAM1

add_0	– – – – – – – – 0 0
add_1	– – – – – – – – – 1
add_2	– – – – – 0 1 0 0
add_3	– – – – – – – 0 1 1
add_4	– – – – – – – 0 1 0
add_5	next available address

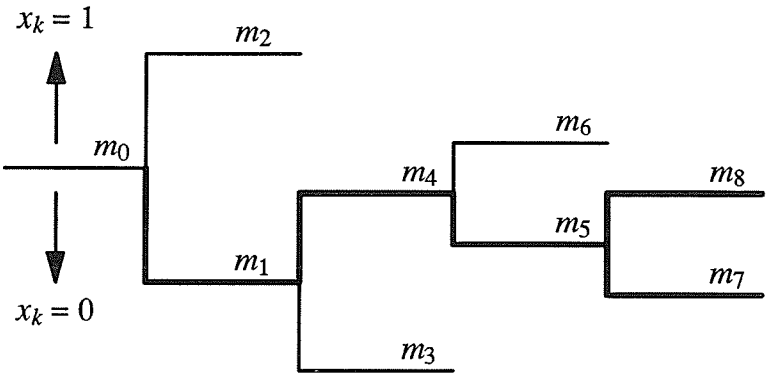
Path Length
RAM2

add_0	0 0 0 0 0 0 1 0
add_1	0 0 0 0 0 0 0 1
add_2	0 0 0 0 0 1 0 0
add_3	0 0 0 0 0 0 1 1
add_4	0 0 0 0 0 0 1 1
add_5	next available address

Explored Path
RAM3 RAM4 RAM5 RAM6

add_0	0 0			
add_1	1			
add_2	0 1 0 0			
add_3	0 1 1			
add_4	0 1 0			
add_5		next available address		

Figure 3.4: The relation of a sample code tree with the systolic priority queue contents and the stack contents after extending branch (0) of the explored path (010). Note: Each add_i in RAM3–6 is an address block of 8 words and each word consists of 32 bits.



SPQ–MU	m_6	m_8	m_7	m_3	m_2	$-\infty$	$-\infty$	$-\infty$	. . .
SPQ–AU	add_3	add_4	add_2	add_0	add_1	0	0	0	. . .

ν –Previous Information Bits
RAM1

add_0	– – – – – – – – 0 0
add_1	– – – – – – – – – 1
add_2	– – – – – 0 1 0 0
add_3	– – – – – – – 0 1 1
add_4	– – – – – 0 1 0 1
add_5	next available address

Path Length
RAM2

add_0	0 0 0 0 0 0 1 0
add_1	0 0 0 0 0 0 0 1
add_2	0 0 0 0 0 1 0 0
add_3	0 0 0 0 0 0 1 1
add_4	0 0 0 0 0 1 0 0
add_5	next available address

Explored Path

	RAM3	RAM4	RAM5	RAM6
add_0	0 0			
add_1	1			
add_2	0 1 0 0			
add_3	0 1 1			
add_4	0 1 0 1			
add_5		next available address		

Figure 3.5: The relation of a sample code tree with the systolic priority queue contents and the stack contents after extending branch (0) and branch (1) of the explored path (010). Note: Each add_i in RAM3–6 is a address block of 8 words and each word consists of 32 bits.

3.4 Hardware Design

In this section, the hardware design of the seven ASIC's shown in figure 3.1 is described. Details of the design of the Systolic Priority Queue–Metric Unit and the Systolic Priority Queue–Address Unit are given first followed by a brief description of the other 5 ASIC's.

3.4.1 Systolic Priority Queue–Metric Unit (SPQ–MU) and Systolic Priority Queue–Address Unit (SPQ–AU)

The hardware configuration of the SR Systolic Priority Queue architecture for the stack algorithm is based on the steps given below.

1) Check for overflow of path metric values in the stack. If overflow occurs, then adjust all the path metric values, else the contents of the registers are held. (adjust or hold).
2) Insert the path metric corresponding to branch (0), (insert data into register A_0).
3) Shift all the path metric values in the queue one position to the right, (shift right).
4) Rearrange the path metric values in A_{2i+1} and A_{2i+2}, so that $A_{2i+1} \geq A_{2i+2}$, (if $A_{2i+1} < A_{2i+2}$, then exchange the data in A_{2i+1} and A_{2i+2}).
5) The contents of the registers are held, (hold).
6) Insert the path metric corresponding to branch (1), (insert data into register A_0).
7) Rearrange the path metric values in A_{2i} and A_{2i+1}, so that $A_{2i} \geq A_{2i+1}$, (if $A_{2i} < A_{2i+1}$, then exchange the data in A_{2i} and A_{2i+1}).
8) Repeat 4).

The above shows that only 5 distinct operations are needed to be performed by the queue, namely, that of adjust, hold, insert data, exchange data in A_{2i+1} and A_{2i+2} and exchange data in A_{2i} and A_{2i+1}. Figure 3.6 shows the circuitry of the hardware to perform the 5 required operations. Table 3.1 shows the logic table for the register A_0 and register A_i, for $i = 1, 2, 3, 4, \ldots$, of figure 3.6.

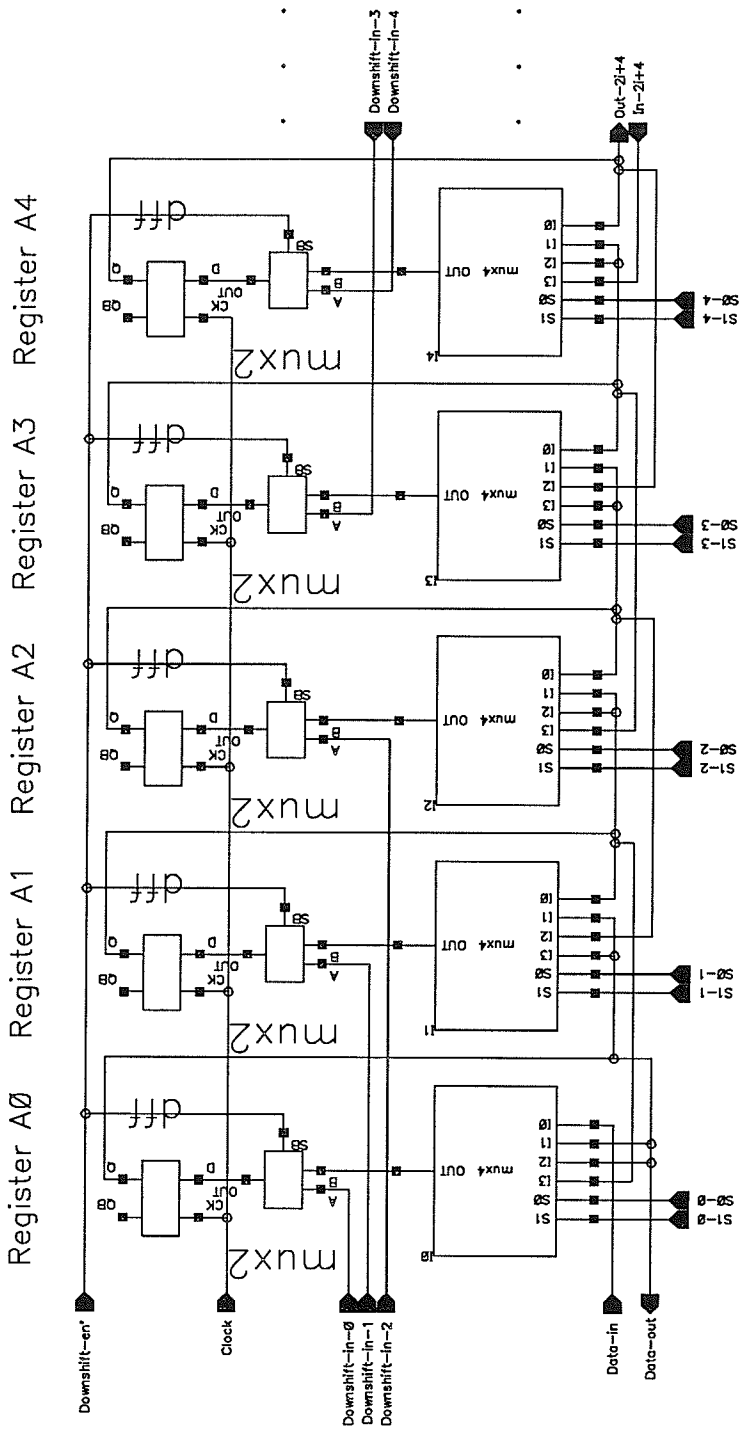


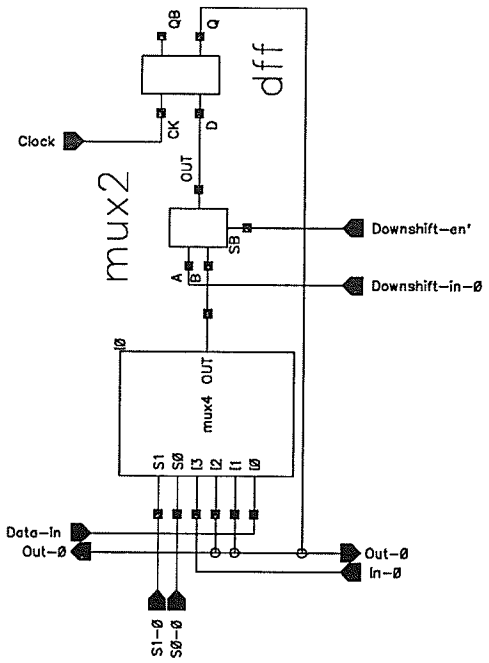
Figure 3.6: Basic hardware structure of the SR Systolic Priority Queue for the stack algorithm. Note: Register A_0 just serves as an input/output port. It is not part of either the SPQ–MU ASIC or the SPQ–AU ASIC.

Downshift–en'	S1	S0	Register A_0	Register A_i
0	0	0	Adjust	Adjust
0	0	1	Adjust	Adjust
0	1	0	Adjust	Adjust
0	1	1	Adjust	Adjust
1	0	0	Insert data	Hold
1	0	1	Hold	Shift all data right
1	1	0	Hold	Exchange data in A_{2i+1} and A_{2i+2}
1	1	1	Exchange data in A_0 and A_1	Exchange data in A_{2i} and A_{2i+1}

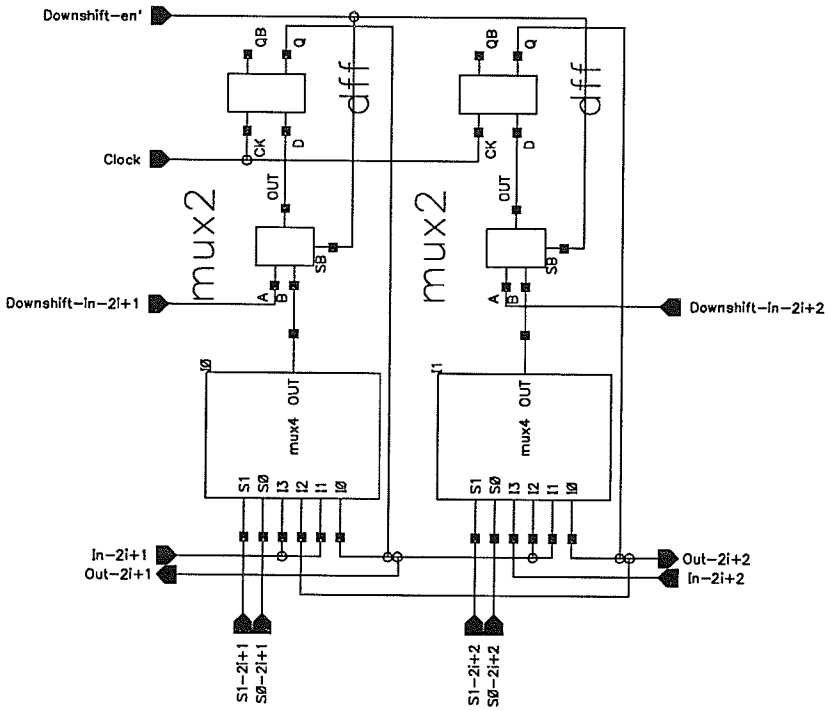
Table 3.1: Logic table for the register A_0 and register A_i in figure 3.6.

Except for register A_0, the remaining registers should be considered in pairs of A_{2i+1} and A_{2i+2}, i.e. (A_1, A_2) is one pair, (A_3, A_4) is another pair identical to (A_1, A_2), etc. Thus register A_0 and the register pair (A_{2i+1}, A_{2i+2}) are the 2 building units of the SR Systolic Priority Queue as shown in figure 3.7. The circuits in figure 3.6 and figure 3.7 really only deal with 1 bit of an M–bit path metric. The building blocks of an M–bit SR Systolic Priority Queue are constructed simply by connecting M of the units in parallel as shown in figure 3.8.

To complete the above building blocks, comparators to compare adjacent path metrics and selectors to control hold, shift and exchange of the shift register contents, need to be added. This is shown in figure 3.9. Having illustrated the basic hardware structure of the SR Systolic Priority Queue for the stack algorithm, the hardware design of the Systolic Priority Queue–Metric Unit ASIC and the Systolic Priority Queue–Address Unit ASIC is described below.

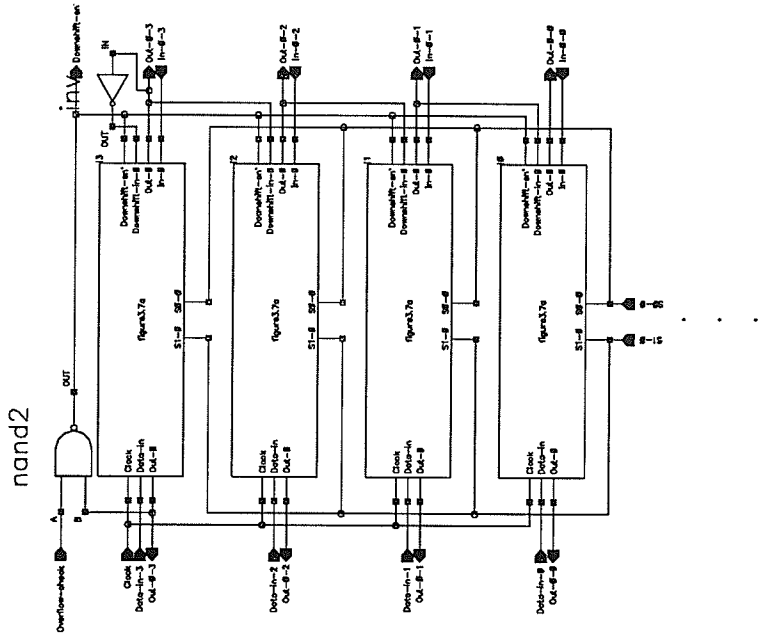


a). Register A_0. Note: It is not part of either the SPQ–MU ASIC or the SPQ–AU ASIC.

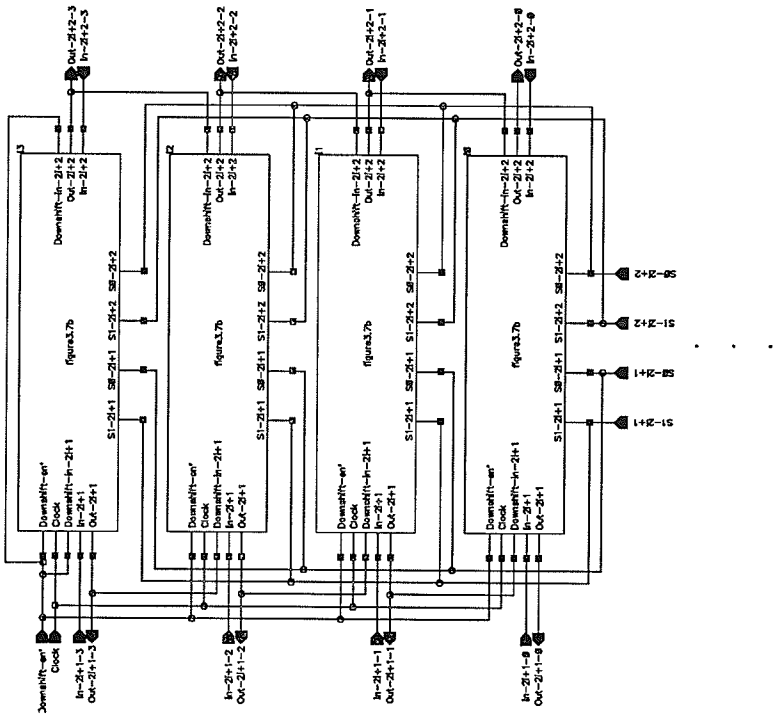


b). Register pair (A_{2i+1} , A_{2i+2}).

Figure 3.7: Basic building units of the SR Systolic Priority Queue for the stack algorithm.



a). Register A_0. Note: It is not part of either the SPQ–MU ASIC or the SPQ–AU ASIC.



b). Register pair (A_{2i+1}, A_{2i+2}).

Figure 3.8: Paralleling the building units of the SR Systolic Priority Queue for the stack algorithm. Note: M–bit building blocks.

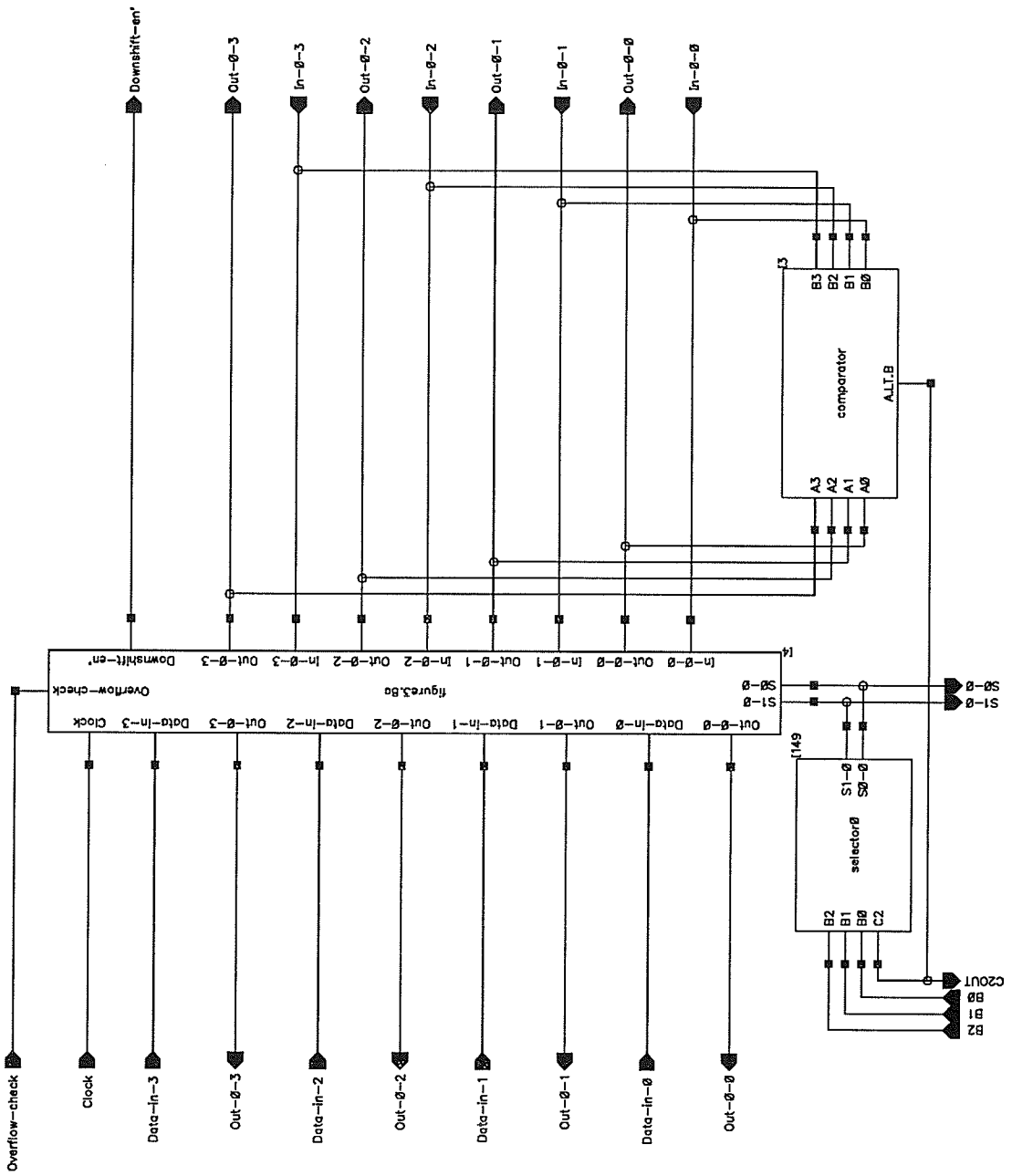


Figure 3.9a: M–bit register A_0 of the SR Systolic Priority Queue for the stack algorithm with comparator and selector added. Note: It is not part of either the SPQ–MU ASIC or the SPQ–AU ASIC.

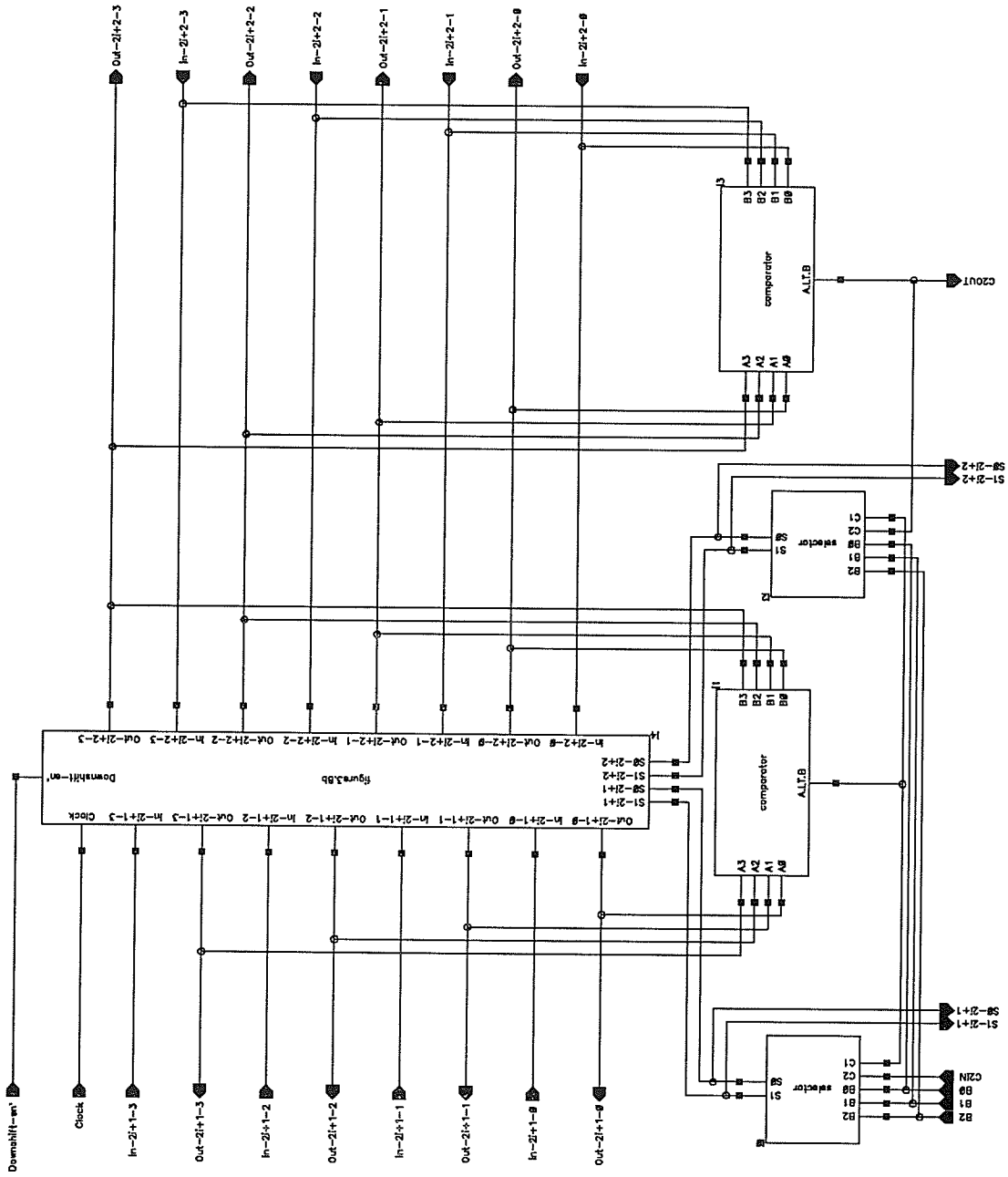
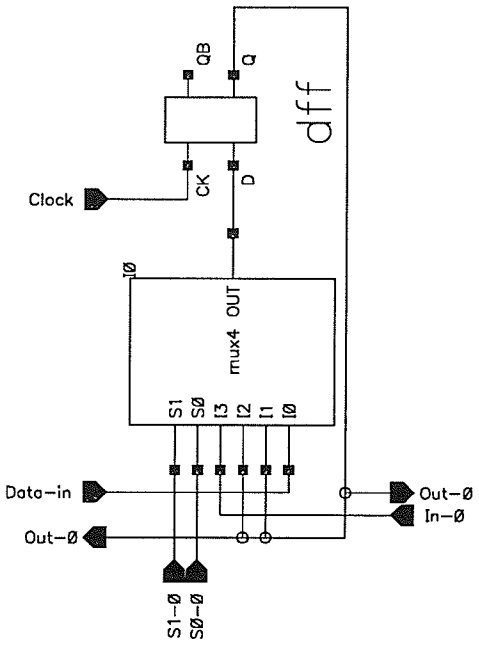


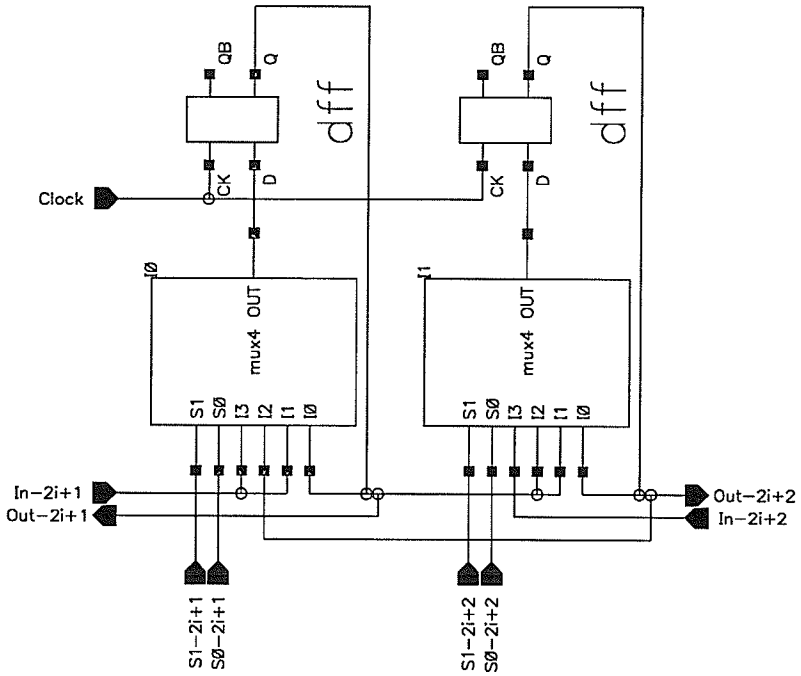
Figure 3.9b: M–bit register pair (A_{2i+1} , A_{2i+2}) of the SR Systolic Priority Queue for the stack algorithm with comparator and selector added. Note: M–bit building block of the SPQ–MU ASIC.

The Systolic Priority Queue–Metric Unit ASIC is a 16–bit version with figure 3.9b used as the building block. It consists of a cascade of four blocks. The Systolic Priority Queue–Address Unit ASIC is different from the SPQ–MU ASIC in the following ways. It is simply used to store the associated addresses of the path metric values in the SPQ–MU ASIC, thus neither overflow–checking nor magnitude comparing capability is required. Moreover, no extra control circuitry is required since the location of the associated addresses changes together with the their corresponding path metrics. Therefore, the 2–to–1 line multiplexers, the comparators and the selectors in the circuits shown in figure 3.7, figure 3.8 and figure 3.9 are not required. The circuits in the figures are simplified to obtain the building units of the SR Systolic Priority Queue for storing the associated addresses. The simplified circuits are shown in figure 3.10 and figure 3.11. Since a stack size of 1024 is chosen in the final design of the sequential demodulator, a 10–bit version of figure 3.11b is used as the building block of the SPQ–AU ASIC. The final SPQ–AU ASIC is constructed by cascading 4 of the 10–bit building block.

This completes the design description of the SPQ–MU and the SPQ–AU. In the following 5 sub–sections, the hardware designs of the other 5 ASIC's are briefly described. For each of the 5 ASIC's, a top level block diagram and the function of each component block are given.

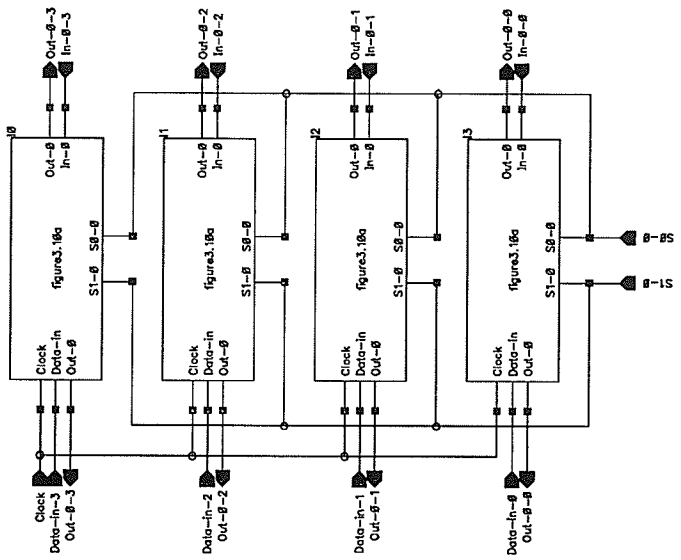


a). Register A_0. Note: It is not part of either the SPQ–MU ASIC or the SPQ–AU ASIC.

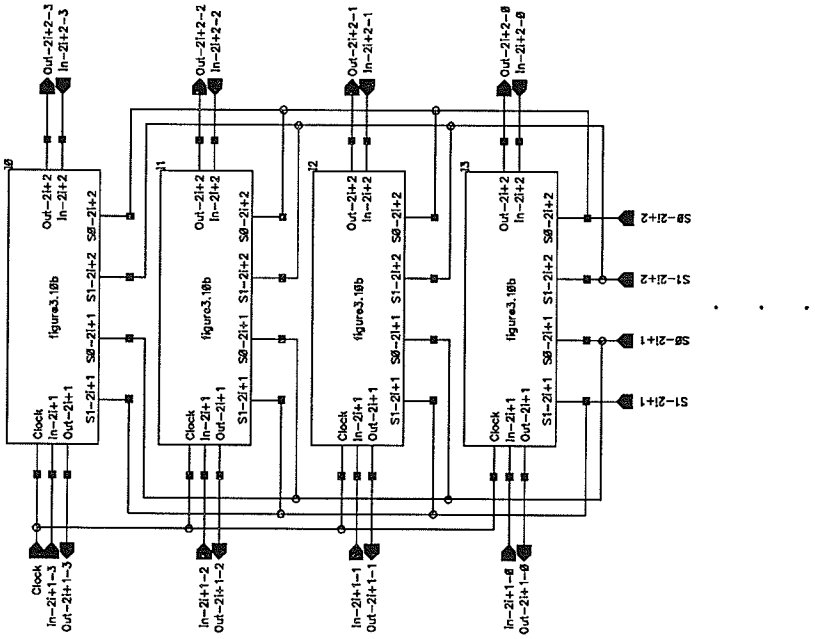


b). Register pair (A_{2i+1} , A_{2i+2}).

Figure 3.10: Basic building units of the address SR Systolic Priority Queue for the stack algorithm.



a). Register A_0. Note: It is not part of either the SPQ–MU ASIC or the SPQ–AU ASIC.



b). Register pair (A_{2i+1}, A_{2i+2}). Note: M–bit building block of the SPQ–AU ASIC.

Figure 3.11: Paralleling the building units of the address SR Systolic Priority Queue for the stack algorithm.

3.4.2 Path Metric Computer (PMC)

The Path Metric Computer calculates the path metric of an extended path, provides an input/output port for the Systolic Priority Queue–Metric Unit and determines the operation modes of the systolic priority queues. Its hardware configuration is illustrated in figure 3.12. It consists of a 16–bit Adder, Register A_0 of the metric systolic priority queue, and a 3–bit Counter.

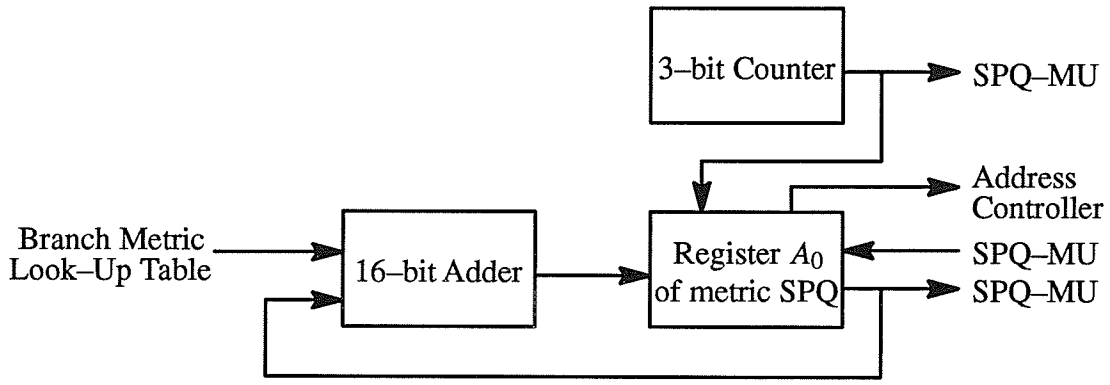


Figure 3.12: Block diagram of the Path Metric Computer.

The 16–bit Adder calculates the path metric of an extended path. It adds the branch metric obtained from the Branch Metric Look–up Table to the current largest path metric from the Register A_0 of the metric systolic priority queue.

Register A_0 functions as an input/output port for the Systolic Priority Queue–Metric Unit. It consists of a 16–bit version of figure 3.8a connected to a comparator and a selector. The output of the selector is connected to the Address Controller for controlling the multiplexers in Register A_0 of the address systolic priority queue. At the beginning of each extension step, Register A_0 outputs the current largest metric to the 16–bit adder. At the end of the each branch extension, the path metric of the extended path is input into Register A_0.

The 3–bit Counter determines the eight operation modes/steps of the stack algorithm (p. 59). Its output, together with the output of the comparator, control the selectors and thus the multiplexers in Register A_0. The output of the 3–bit Counter also apply to the Systolic Priority Queue–Metric Unit. It combines with the outputs from the two adjacent comparators of each register in the Systolic Priority Queue–Metric Unit. The result controls the multiplexers in each register of the Systolic Priority Queue–Metric Unit and the Systolic Priority Queue–Address Unit.

3.4.3 Address Controller (AC)

The Address Controller keeps track of the next available address in RAM1–6, provides an input/output port for the Systolic Priority Queue–Address Unit and controls the addresses of RAM1–6. Its hardware configuration is illustrated in figure 3.13. It consists of a 10–bit Counter, Register A_0 of the address systolic priority queue, Multiplexer1 and Multiplexer2.

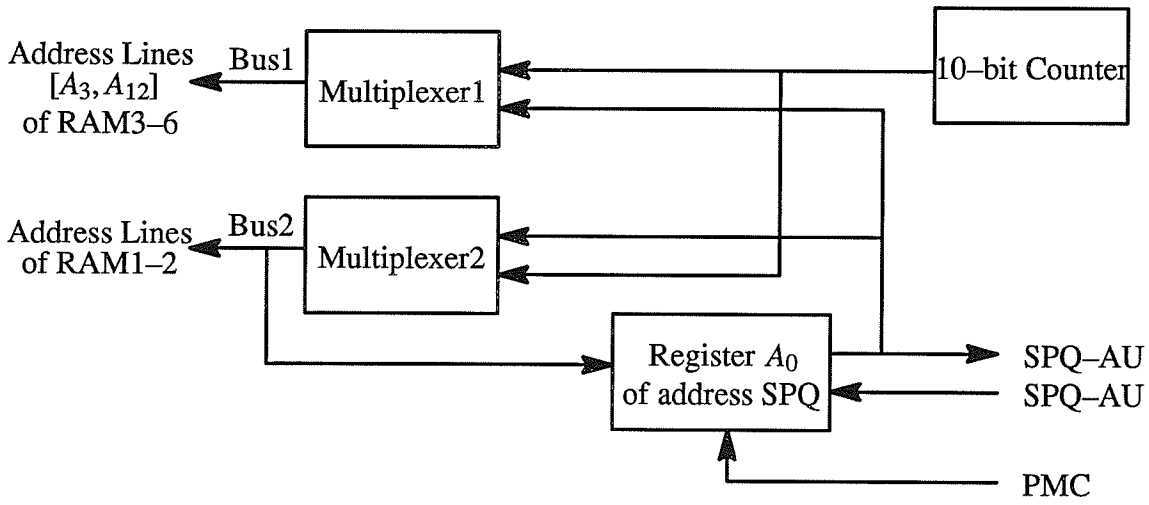


Figure 3.13: Block diagram of the Address Controller.

The 10–bit Counter keeps track of the next available address location in RAM1–6. At the end of each extension step, it up–counts its current value by one to locate the next available address in RAM1–6 for the next extension step.

Register A_0 functions as an input/output port for the Systolic Priority Queue–Address Unit. It consists of a 10–bit version of figure 3.11a. The operation of the Register A_0 is controlled through the selector output from the Path Metric Computer. At the beginning of each extension step, Register A_0 outputs the associated address of the current largest metric to the two multiplexers. At the end of the first branch extension, Multiplexer2 inputs the associated address into Register A_0. After the second branch extension, Multiplexer2 inputs the value of the 10–bit Counter into Register A_0.

Multiplexer1 and Multiplexer2 control the addresses of RAM1–6. Each multiplexer is constructed from 10 2–to–1 line multiplexers. They accept the associated address of the current largest path metric from Register A_0 and the next available address from the 10–bit Counter as input, then output the appropriate address to RAM3–6 and RAM1–2 respectively.

3.4.4 L–Previous Information Bits Buffer (LPIBB)

The L–Previous Information Bits Buffer duplicates the data in RAM1, controls the address lines $[A_8, A_{17}]$ of the Branch Metric Look–Up Table and updates the 9–previous information bits of an extended path. Its hardware configuration is depicted in figure 3.14. It is a 10–bit Shift Register which consists of ten D–flip–flops and ten 3–to–1 line multiplexers. Refer to section 3.3 for details.

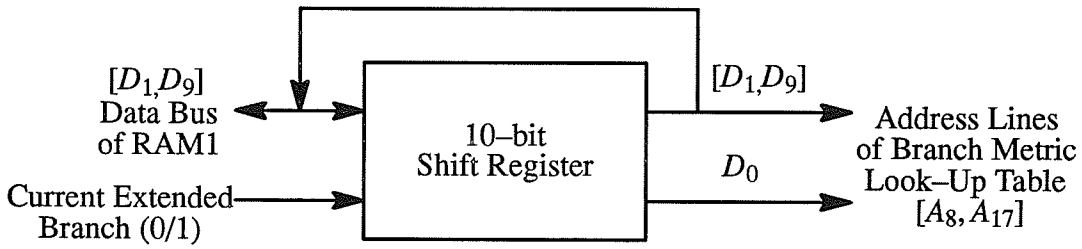


Figure 3.14: Block diagram of the L–Previous Information Bits Buffer.

3.4.5 Path Length Buffer (PLB)

The Path Length Buffer duplicates the data in RAM2, controls the address lines $[A_0, A_2]$ of RAM3–6, controls the address lines of the Sufficient Statistic Look–Up Table, controls the input of the Extended Path Controller for updating an extended path in RAM3–6, and updates the path length of an extended path. Figure 3.15 shows the hardware configuration. It consists of an 8–bit Counter, a 3–bit Counter and a Multiplexer Unit. Refer to section 3.3 for details.

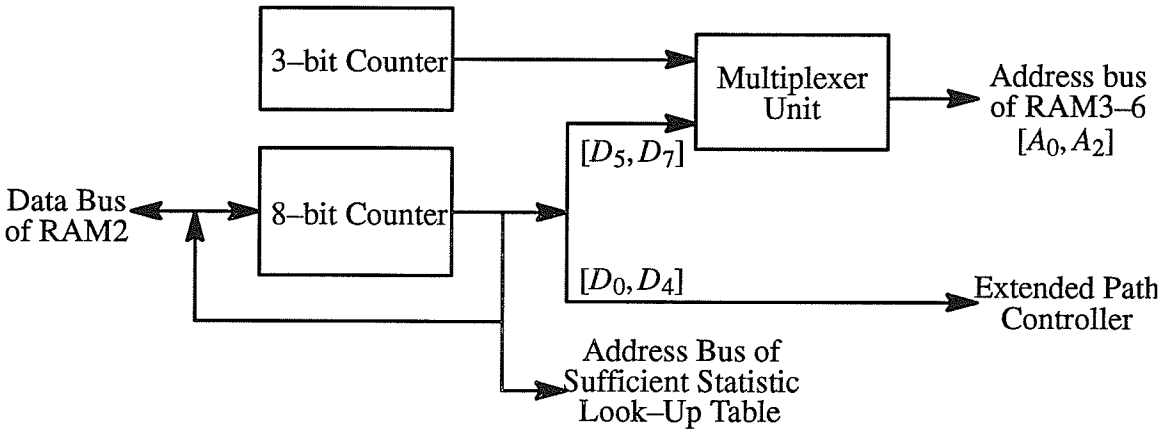


Figure 3.15: Block diagram of the Path Length Buffer.

3.4.6 Explored Path Controller (EPC)

The Explored Path Controller controls the updating of an explored path in RAM3–6. Figure 3.16 shows the hardware configuration of the EPC. It consists a Multiplexer Unit, a D–flip–flop unit and a 5x32 Decoder. Refer to section 3.3 for details.

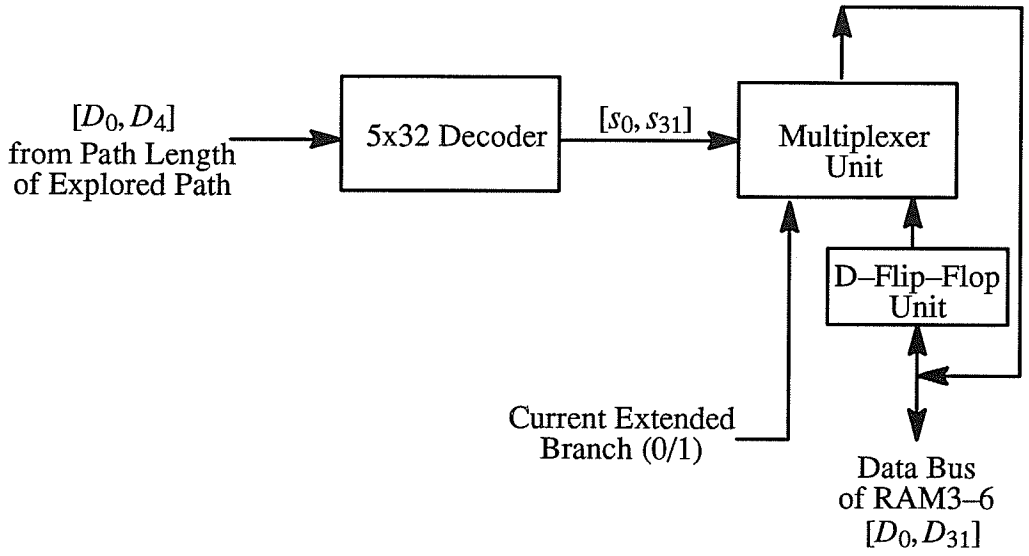


Figure 3.16: Block diagram of the Explored Path Controller.

Having described the hardware design of the seven ASIC's, the testing result of the ASIC's and a discussion of the sequential demodulator are given in the next chapter.

Chapter 4 Test Result and Discussion

In this chapter, the test result of the seven ASIC's is first presented. Then a discussion of the sequential demodulator based on its applicability is given.

4.1 Test Result

The ASIC's were designed using the CAD tool Cadence™ (also known as EDGE, SDA, or CDS) which uses a design methodology called schematic capture. In this environment, the schematic of each ASIC is drawn, with the placing and routing of standard cells done automatically from the schematic. In other words, silicon level representations of the ASIC's are derived from very simple circuit sketches.

Before sending the designs for fabrication, the schematics were simulated under the SILOS simulator for verification of design logic. SILOS is executed within Cadence using a netlist generated from Cadence. The output of the simulation is viewed by the Waveform Display package provided in the Cadence environment. Since no design errors were found in the simulation, the layouts (i.e. the silicon representations) were translated to the Caltech Intermediate Form (CIF) files and submitted to the Canadian Microelectronics Corporation (CMC) where they were fabricated with the three micron CMOS process.

Upon receipt of the ASIC's, they were tested with an ASIX–2 tester. A static test was performed on each ASIC. During the test, input test vectors were applied to an ASIC and the outputs were compared with the designated output test vectors. None of the ASIC's passed the static test. In all the ASIC's, there are inputs and outputs that are stuck at '1'.

Further investigation was done to find out the cause of the stuck at '1' problem. Each ASIC was inspected under the microscope. The inspection revealed unconnected pads which corresponded to the inputs and outputs that exhibited the stuck at '1' problem in the static test. In order to determine whether the routing errors are in the fabrication or in Cadence, layouts of the ASIC's were examined. This examination confirmed that the unconnected

pads are due to the layouts generated by Cadence. The unconnected pads resulted in four ASIC's which could not be tested, the other three were only partially tested.

The four ASIC's that were unable to be tested are the Systolic Priority Queue–Metric Unit, the Systolic Priority Queue–Address Unit, the Path Metric Computer and the L–Previous Information Bits Buffer. The Systolic Priority Queue–Metric Unit and the Systolic Priority Queue–Address Unit have fourteen and six unconnected pads respectively, however the crucial pad that makes the two ASIC's not testable is the input pad *CK*, which drives the clock signal. The Path Metric Computer has ten unconnected pads. It was unable to be tested because the two input pads *SET* and *RESET* which drive the set and reset signals of the ASIC are not connected. The L–Previous Information Bits has two unconnected pads, however the reason for test failure was not caused by the unconnected pads. The ASIC was unable to be powered up. It is suspected that there exists a short between power and ground. Unfortunately, the fault in this ASIC could not be located.

During the test of the Address Controller, malfunction of the 10–bit Counter and the output *A7* were observed. The malfunction was caused by two unconnected pads, *CE_AC* and *A7*, which appeared as stuck at '1' faults. The counting operation of the 10–bit Counter could not be disabled since the count enable input *CE_AC* of the 10–bit Counter is stuck at '1'. Similarly, the counting operation of the 8–bit Counter in the Path Length Buffer could not be disabled because its count enable input *PL_UP_COUNT_EN* is stuck at '1'. In addition, the outputs *DPL3*, *DPL7*, *A3* and *A7* of the 8–bit Counter are not connected and all appear as stuck at '1' faults. Finally, the 5x32 Decoder in the Explored Path Controller did not perform properly. One input, *AZ1*, of the 5x32 Decoder is not connected and stuck at '1'. Thus whenever the input to *AZ1* was '0', the decoder gave an incorrect output.

To summarize, the main problem of the ASIC's is the unconnected pads originated from the layouts. All the unconnected pads appeared as stuck at '1' faults in the test. The problem was caused when the layouts were generated using the router in Cadence. The Detail

Routing stage was proceeded before all nets were routed in the Global Routing stage. This resulted in unconnected pads. In the Global Routing stage, there is no guarantee that all nets are completely routed by just one trial regardless of the constraints imposed on the locations of the I/O and standard cells. If some of the nets are reported partially routed, then the global routing has to be undone and the Global Routing stage needs to be repeated. The above is repeated until all nets are completely routed. If all nets are completely routed, then the Detail Routing stage can proceed. The layouts of the seven ASIC's are regenerated in Cadence. With several trials in the Global Routing stage for each layout, layouts with all pads connected are obtained.

4.2 Applicability of the Sequential Demodulator

The applicability of the sequential demodulator is discussed in terms of the number of interference terms in the ISI channel, the length of the input to the ISI channel and the size of the stack required.

4.2.1 Number of interference terms in the ISI channel

The sequential demodulator shown in figure 3.1 is designed to handle ISI channels with at most nine interference terms. If the ISI channel has exactly nine interference terms, no modification to the design shown in figure 3.1 has to be made. Specific branch metric look–up tables have to be prepared and used for different ISI channels. However if the number of interference terms changes, the design has to be modified accordingly.

If there are less than nine interference terms, only the L–Previous Information Bit Buffer and the Branch Metric Look–Up Table need to be modified. If there are $n < 9$ interference terms, then the $9-n$ most significant output lines of the L–Previous Information Bit Buffer, and therefore the $9-n$ most significant address lines of the table, are not required. They should hardwired to either '0' or '1'. If they are hardwired to '0', then only the first 2^{8+n+1} addresses of the look–up table are used, else the last 2^{8+n+1} addresses of the look–up table are used. The table has to be programmed accordingly or can be substituted by a smaller dimension memory IC.

4.2.2 Length of input to the ISI channel

The sequential demodulator is designed to decode a block of 256 bits. If the sequence length of the input is just one block, then the decoding proceeds as described in chapter 3. However, if the length of input is longer than one block, then the following assumption has to be made. The node with the largest path metric from the previous block is assumed to be the root node of the current input block. That is, the path corresponding to that particular node is assumed to be the desired path for that block. The last nine symbols of the decoded path from the

previous block is assumed to be the 9–previous information bits corresponding to the first symbol in the current input block. To continue the decoding, the 9–previous information bits is input into the address (0000000000) of RAM1 and the rest of the system is reset to the initial state. Then the decoding of the current block proceeds as described in chapter 3. The final decoded path is obtained by concatenating the results obtained from the input blocks.

4.2.3 Stack overflow problem

The sequential demodulator is designed to have a stack size of 1024. If stack overflow occurs, an overflow flag signifies the occurrence of overflow. If the decoding continues, the result becomes meaningless. When overflow occurs, the value of the 10–bit Counter in the Address Controller goes back to (0000000000) which is the location of the next available address in RAM1–6. Due to the storage scheme of the design, address (0000000000) of RAM1–6 might contain data for the most likely path. A better storage scheme has to be developed to handle the stack overflow problem.

Chapter 5 Conclusions and Suggestions for Further Study

5.1 Conclusions

In this thesis the VLSI implementation of a sequential demodulator based on the Shift Register Systolic Priority Queue architecture applicable to ISI channels is developed and performed. The sequential demodulator consists of eight standard memory IC's and seven ASIC's that are fabricated with the three micron CMOS process. The design is capable of handling input blocks of 256 bits and a maximum of 9 interference terms. It has a maximum allowable stack size of 1024. An 8–bit representation is used for the input sufficient statistic z_k and a 16–bit representation for the metric.

The idea of Chang and Yao [2] in using the Shift Register Systolic Priority Queue to substitute the stack in the conventional sequential stack algorithm is used for the VLSI implementation of the sequential demodulator. The proposed algorithm is investigated and implemented into hardware. Two Shift Register Systolic Priority Queues applicable to the stack algorithm are implemented for manipulating the path metrics and their associated addresses. The Shift Register Systolic Priority Queue architecture is used to eliminate the time consuming and number–of–entries dependent stack reordering problem in the conventional stack algorithm. With the systolic priority queue architecture, complete stack reordering is no longer required and the largest path metric is guaranteed to appear at the top of the stack within a fixed and short interval of time regardless of the number of metrics in the stack.

To complete the design of the sequential demodulator, algorithms for storing the explored paths and evaluating the metrics are developed and hardware implemented. The input sufficient statistic z_k and all possible branch metric values are available as two look–up tables. The output explored paths are stored in a RAM. To facilitate the storage of explored paths and the evaluation of branch metrics, the path lengths and the 9–previous information

bits of the explored paths are also stored. The technique of indirect addressing is used to store and locate the path length, the 9–previous information bits and the explored path of a particular path metric. Associated with each path metric in the metric systolic priority queue, there is a value in the corresponding position in a second systolic priority queue that represents the address of the above three data in the RAMs. The knowledge of the path length is used to locate the specific position for storing the extended path and retrieve the specific sufficient statistic z_k. The retrieved sufficient statistic z_k and the 9–previous information bits are then used as the address to retrieve the expected branch metric from the branch metric look–up table.

5.2 Suggestions for Further Study

The area requirement of the design is quite large and thus limits the feasibility of the design. Further investigation needs to be done to overcome this problem. The 16–bit metric representation and the 10–bit associated address of each path metric are the two main factors contributing to the area requirement of the design. Since a 16–bit representation is chosen for the metric, only eight entries can be accommodated in the Systolic Priority Queue_Metric Unit. Thus a total of 128 Systolic Priority Queue–Metric Unit ASIC's are required to construct a stack size of 1024. More research is definitely needed to establish a new metric representation that requires fewer bits and provides satisfactory error performance. Similarly, 128 Systolic Priority Queue–Address Unit ASIC's are required to construct the stack of size 1024 for storing the associated addresses. The 10–bit associated address is required because of the indirect addressing technique used in the design. New algorithms for storing the explored paths and evaluating the metrics need to be developed in order to eliminate the requirement of the 10–bit associated address. Also, modification of the design to handle the stack overflow problem and multiple stack decoding are two other possible extensions to the work presented in this thesis.

REFERENCES

[1] J. B. Anderson and C. F. Lin, "M–Algorithm Decoding of Channel Codes," *Proc. 13th Biennial Symp. on Commun., Queen's University, Kingston, Canada*, pp. A3.1–A3.4, June 2–4, 1986.

[2] M. E. Austin, "Decision–Feedback Equalization for Digital Communication over Dispersive Channels," *M.I.T. Lincoln Lab., Lexington, Mass.*, Tech. Rep. 437, August 1967.

[3] C. Y. Chang and K. Yao, "Systolic Array Architecture for the Sequential Stack Decoding Algorithm," *SPIE Conf.*, San Diego, U.S.A., pp. 196–203, 1986.

[4] L. W. Couch, *Digital and Analog Communication Systems*, Macmillan, New York, 1990.

[5] A. Duel and C. Heegard, "Delayed Decision Feedback Sequence Estimation," *23rd Annual Allertor Conference on Communication, Control, and Computing*, pp. 878–887, 1985.

[6] M. V. Eyuboglu and S. U. Qureshi, "Reduced–State Sequence Estimation with Set Partitioning and Decision Feedback," *IEEE Trans. Commun.*, vol. COM–36, pp. 13–20, January 1988.

[7] G. D. Forney, Jr., "Maximum–Likelihood Sequence Estimation of Digital Sequences in the Presence of Intersymbol Interference," *IEEE Trans. Inform. Theory*, vol. 18, pp. 363–378, May 1972.

[8] G. D. Forney, Jr., "The Viterbi Algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, March 1973.

[9] G. D. Forney, Jr., and M. V. Eyuboglu, "Combined Equalization and Coding Using Precoding," *IEEE Commun. Mag.*, pp. 25–34, December 1991.

[10] H. Harashima and H. Miyakawa, "A Method of Code Conversion for a Digital Communication Channel with Intersymbol Interference," *Trans. Inst. Electron. Commun. Eng.*, Japan, vol. 52–A, pp. 272–273, June 1969.

[11] H. Harashima and H. Miyakawa, "Matched–Transmission Technique for Channels with Intersymbol Interference," *IEEE Trans. Commun.*, vol. COM–20, pp. 774–780, August 1972.

[12] E. R. Kretzmer, "Binary Data Communication by Partial Response Transmission," *Conference Record, IEEE Annual Commun. Convention*, pp. 451–455, 1965.

[13] E. R. Kretzmer, "Generalization of a Technique for Binary Data Communication," *IEEE Trans. Commun.* Tech., vol. COM–14, pp. 67–68, February 1966.

[14] W. U. Lee and F. S. Hill, "A Maximum–Likelihood Sequence Estimator with Decision–Feedback Equalization," *IEEE Trans. Commun.*, vol. COM–25, pp. 971–979, September 1977.

[15] A. Lender, "Correlative Digital Communication Techniques," *IEEE Trans. Commun.* Tech., vol. COM–12, pp. 128–135, December 1964.

[16] A. Lender, "Correlative Level Coding for Binary–Data Transmission," *IEEE Spectrum*, vol. 3, no. 2, pp. 104–115, February 1966.

[17] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications.* Prentice–Hall, Englewood Cliffs, N. J., 1983.

[18] R. W. Lucky, J. Salz, and E. J. Weldon, Jr., *Principles of Data Communication.* McGraw–Hill, Inc., New York, N.Y., 1968.

[19] H. Nyquist, "Certain Topics in Telegraph Transmission Theory," *Trans. AIEE (Commun. and Electronics)*, vol. 47, pp. 617–644, April 1928.

[20] A. Polydoros and D. Kazakos, "Maximum Likelihood Sequence Estimation in the Presence of Infinite Intersymbol Interference," *Proc. ICC'79, Boston, MA*, June 1979.

[21] S. Qureshi and E. Newhall, "An Adaptive Receiver for Data Transmission over Time–Dispersive Channels," *IEEE Trans. Inform. Theory*, vol. 19, pp. 448–457, June 1973.

[22] S. Qureshi, "Adaptive Equalization," *IEEE Commun. Mag.*, pp. 9–16, March 1982.

[23] N. Seshadri and J. B. Anderson, "An M–Algorithm Receiver for Severe Infinite Intersymbol Interference Channels," *Abstract of Papers, International Symposium on Information Theory, Kobe, Japan*, pp. 68, June 1986.

[24] M. Tomlinson, "New Automatic Equalizer Employing Modulo Arithemetic," *Electron. Lett.*, vol. 7, pp. 138–139, March 1971.

[25] G. Ungerboeck, "Channel Coding with Multi–Level/Phase Signals," *IEEE Trans. Inform. Theory*, vol. 28, pp. 55–67, January 1982.

[26] F. Xiong, A. Zerik, and E. Shwedyk, "Sequential Sequence Estimation for Channels with Intersymbol Interference of Finite or Infinite Length," *IEEE Trans. Commun.*, vol. COM–38, pp. 795–804, June 1990.