Softbit Detector / Equalizer for GSM release 7

Xia Wu

Kongens Lyngby 2008
IMM-MSc-2008-91

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

This thesis deals with the implementation of a power/area optimal equalizer according to the recently updated GSM specification. The equalizer should support GMSK, QPSK, 8PSK, 16QAM and 32QAM modulation types. For the high order modulation types, the traditional Maximum Likehood Sequence Estimation(MLSE) algorithm is not able to process the burst within the required time. Therefore two different algorithms with reduced computation complexity are studied in the thesis: the Reduced-State Sequence Estimation with Set-partitioning(RSSE) algorithm and the Sphere Decoding(SD) algorithm. The RSSE algorithm reduces the complexity by grouping the states into subsets in the trellis structure, while the SD algorithm addresses the problem by constraining the trellis search with a threshold. Since the two algorithms optimize the MLSE approach in different aspects, a hybrid algorithm (RSSE_T) is proposed. The project implemented all three algorithms in RTL and compared their performance in area, power and delay. The SD algorithm is implemented in two different approaches, named SD_I and SD_II. The performance evaluation shows the area cost of the RSSE, the SD_I, the SD_II and the RSSE_T equalizers is 62102 μm^2, 94050 μm^2, 182965 μm^2, and 78317μm^2 respectively, for the given 45nm CMOS technology. When clocked at 200MHz and given a normal operating voltage between 1.0-1.3V, the power consumption of the four equalizers is typically less than 8.2mW, 6.8mW, 10.3mW and 7.8mW, respectively. The SD_I equalizer is unable to satisfy the timing requirements, thus is the least interesting candidate for physical implementation. The RSSE_T equalizer consumes constantly 30% less power than the SD_II equalizer, and in the best case reduces 50% of the power consumed by the RSSE equalizer, therefore it is the most efficient implementation among these four designs.

Keywords: GSM, trellis algorithm, channel equalization, modulation, ASIC

Preface

This thesis was written in partial fulfillment of the requirements for acquiring the Master of Science degree in engineering. The project has been carried out in 6 months' duration from 1st March to 30th September, 2008 in Modem IP, Copenhagen, Nokia Denmark. The workload corresponds to 35 ECTS points. The supervisor of the project are Roy Hansen, Nokia Denmark, and Alberto Nannarelli, Informatics and Mathmatical Modelling, Technology University of Denmark(DTU).

First of all, I would like to thank my supervisors, who have given me invaluable help and guidance throughout the project. Also I would like to thank Associate Professor Flemming Stassen, who has showed me a lot of support and good suggestions during my 2 years' master study at DTU. A special thank to Morten Hansen for several discussion of the subjects related to my thesis work. Also I would like to thank the manager of Modem IP, Peter Mårtensson, and the team leader, Stig Rasmussen, who gave me the opportunity to work on the project in their department. Finally, my husband Kehuai and my daughter Sophie, who are the source of all my inspirations.

Copenhagen, September 2008

Xia Wu

Contents

List of Abbreviation

AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
DF	Decision Feedback
GMSK	Gaussian Minimum Shift Keying
GSM	Group Special Mobile (Global System for Mobile communication)
HT	Hilly Terrain
I	In-phase
ISI	Intersymbol Interference
LMMSE	Linear Minimum Mean Square Error
MLSE	Maximum Likelihood Sequence Estimation
OCP	Open Core Protocol
PSK	Phase Shift Keying
Q	Quadrature
QAM	Quadrature Amplipude Modulation
QPSK	Quadrature Phase Shift Keying
RAM	Random Access Memory
ROM	Read Only Memory
RSSE	Reduced-State Sequence Estimation with Set-partitioning
RTL	Register Transfer Level
SD	Sphere Decoding
SNR	Signal-to-Noise Ratio
TCM	Trellis Coded Modulation
TU	Typical Urban
VA	Viterbi Algorithm
VCD	Value Change Dump
VHDL	Very high speed integrated circuit Hardware Description Language

Chapter 1

Introduction

Global System for Mobile communications (GSM) has become the current mainstream standard for mobile phones. More future applications have been identified by the mobile phone manufacturers. These applications have increasingly demanding requirement for cell phone's energy-efficiency and data processing capability. The GSM specification has been updated regularly to meet the increasing demand of the mobile phone users and manufacturers.

For the release 7 of the GSM specification, it is proposed to add 16QAM, 32QAM, and high-rate QPSK to the modulation types that are being used today. These modulation types have larger alphabet size, which consequentially increase data rate. E.g. for 32QAM, the data rate is 5 times as high as the data rate of GMSK. Unfortunately, the computational complexity of high-rate modulation types such as 32QAM increases significantly.

It has always been a concern in wireless communication that when transmitted through a noisy communication channel, linearly modulated uncoded data is subject to severe Inter-Symbol Interference(ISI). An optimal way for restoring signal in detection is to perform Maximum Likelihood Sequence Estimation(MLSE) proposed by [2], which estimates the most likely sequence by using Viterbi Algorithm(VA) [3]. The VA is well known for its effectiveness but exponential growth of complexity, thus cannot be directly applied to higher-order modulation types. Considerable amount of research has been carried out

to find a sub-optimal algorithm which can still reach the BER performance of the MLSE at reduced computational complexity. One method is to use a structured reduced-state sequence estimator [4]. This method uses Viterbi Algorithm(VA) with decision feedback to search a reduced-state "subset trellis". Another method for performing MLSE in a computational efficient way is the Sphere detection(SD) algorithm [5] [6].

Since low area/power cost is a key element in the mobile phone design, it is of great importance to examine and compare the power consumption of these near-optimal algorithms. Also, the physical size of the equalizer should be kept as small as possible.

1.1 Project Outline

The main task of the project is to design the equalizer using RSSE algorithm and using SD algorithm, to optimize both design for area and power, and to compare their performance. Furthermore, a hybrid algorithm, called RSSE_T, which combines the characteristics of the two algorithms is proposed and evaluated. The algorithms used in the project are based on [4], [5], and [6], with some modifications which make the algorithms feasible for hardware implementation.

To verify the function of the hardware model, a Matlab reference model of the equalizer has also been made. It can both generate test vectors of different modulation types, and provide the reference results of the equalizer. The reference model supports the RSSE algorithm, the SD algorithm and the RSSE_T algorithm.

The RTL level hardware models have been made for all three algorithms using VHDL-93. The functional tests have been carried out in ModelSim. The design has been synthesized by Synopsys tool suite. The critical path latency and area cost of the gate-level model was checked by using Synopsys Design Compiler (DC). The power consumption of these equalizer models of different algorithms was measured by PowerTheater 2008 at RTL level. The energy consumption was calculated based on the power consumption and the computation time. The design is optimized for area and power at architecture and algorithm level.

The target is 45nm Low Power CMOS technology. In the hardware design, some RAM modules have been used. The RAM modules are simulation models provided by Denali software PureView.

1.2 Thesis organization

Chapter 1 gives a brief introduction of the project.

In chapter 2, the background knowledge of the project has been presented. These include the modulation schemes and frame format of GSM, frequency selective channel properties and equalizer, and a short description of the MLSE, the RSSE, the SD and the RSSE_T algorithms.

In chapter 3, the design of the equalizer using the RSSE algorithm, the SD algorithm and the RSSE_T algorithm has been described. The major difference among these algorithm have been explained in details. A comparison of the complexity of design has been made.

The simulation environment has been explained in chapter 4. The functional tests are described and the simulation results are presented. The area, power consumption and energy consumption of the four implementations are measured and compared.

Chapter 5 describes the future work of further optimization in power and area and concludes the project.

Chapter 2

Technology Overview

2.1 Modulation Schemes in GSM

In GSM modulation schemes, the symbols are often represented by vectors in 2-dimensional space. Here the x and y-axis are called In-phase(I) and Quadrature(Q) projection of the signal. As shown in figure 2.1, the symbol S0 has the I and Q projection of (I_0, Q_0). By representing a transmitted symbol as a vector and modulating a cosine and sine carrier signal with the I and Q parts, the symbol can be sent with two carriers on the same frequency. A diagram of the ideal positioning of the symbols in a modulation scheme is called a constellation diagram. The figure 2.1 shows the symbol S0 on a GMSK constellation diagram, where the ideal positions of the symbols '0' and '1' are marked by black dots.

On the receiver side, the equalizer in the demodulator examines the received symbols which may have been corrupted considerably by the channel. It evaluates the position of the received symbol and calculates the probability of each bit in the symbol being 1 or 0. This can be visualized as calculating the distance of the received symbol to each of the reference point on the constellation diagram, and choose the closest point to be the received symbol. In the example of figure 2.1, the closest point to symbol S0 is the symbol '1'.

2.1.1 GMSK

One of the most commonly used modulation schemes in GSM is Gaussian minimum shift keying(GMSK). In GMSK constellation diagrams shown in 2.1, the symbols are (-1,0) and (1,0) marked by black dots, representing symbol '0' and '1'. The bits are mapped to symbols according to [7].GMSK has the advantage of having the largest distance between constellation points, and thus the best immunity to corruption among the modulation schemes. However, the GMSK contains only one bit in the symbol and therefore requires more power to transmit the same amount of data than other modulation schemes.

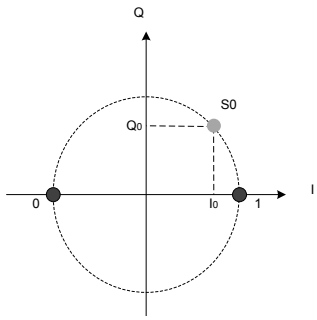

Figure 2.1: The In phase and Quadrature projection of a symbol in a GMSK constellation diagram

2.1.2 PSK

M-ary PSK is a phase shift keying modulation scheme used in GSM. In PSK, the constellation points are usually placed on a unit circle with equal spaces between the adjacent symbols, which gives maximum phase-separation. The same amplitude ensures the same energy to transmit the symbols.

Each symbol in 8PSK contains three bits. The symbols are Gray mapped into the constellation diagram, of which the two successive symbols only differ in one bit. In the GSM standard, a modified 8PSK is used, of which the constellation diagram is rotated by $3/8\pi$. The constellation diagram of 8PSK, defined by [7], is shown in the left side of figure 2.2.

QPSK is one of the more recently supported modulation schemes proposed in GSM release 7. Similar to 8PSK, QPSK is a phase shift keying of four symbols,

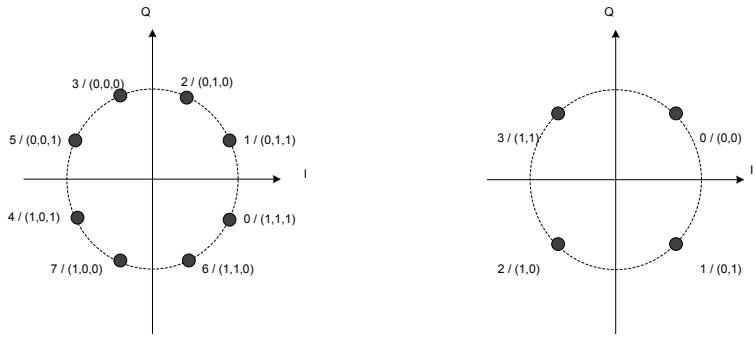

Figure 2.2: Left: 8PSK constellation diagram, right: QPSK constellation diagram

and each symbol contains two bits. Unlike the GSM and 8PSK modulation, QPSK modulation is only used at high symbol rate of 325k symbol/s, while the former two operate at a normal symbol rate of 270.8k symbol/s. A $3/4\pi$-rotated QPSK is proposed by the GSM standard. The constellation diagram of QPSK, defined by [7] is shown in the right side of figure 2.2.

2.1.3 QAM

In the situation of more than eight constellation points, the Quadrature Amplitude Modulation(QAM) are used instead of Phase Shift Keying. In the GSM standard, the constellation points in QAM are arranged in a rectangular lattice with odd-integer coordinated($\pm 1, \pm 3, \cdots$). Compared to PSK, the constellation points in QAM are distributed more evenly on the constellation plane, which give less probability of error. But the inter-symbol difference in both amplitude and phase makes the modulation more complicated. QAM usually contains more than 4 bits in one symbol, thus it is used in application which required high transmission rate.

The GSM release 7 proposed 16QAM and 32QAM, at both the normal symbol rate and the higher symbol rate. The constellation diagram of these two modulation schemes, according to [7], are shown in figure 2.3,

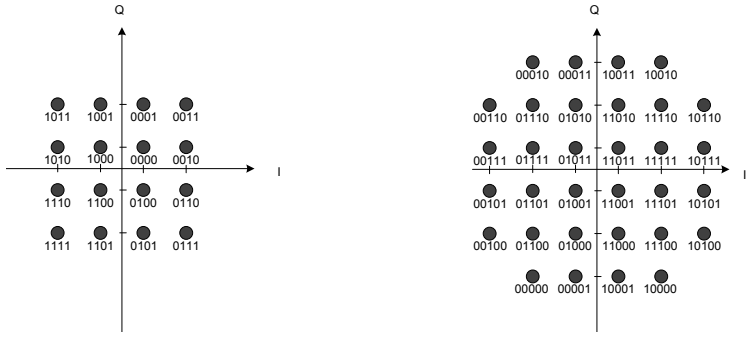

Figure 2.3: Left: 16QAM constellation diagram, right: 32QAM constellation diagram

2.2 Pulse shaping

The modulating symbols excite a linear pulse shaping filter. This filter is a linearized GMSK pulse, i.e. the main component in a Laurant decomposition of the GMSK modulation. The impulse response of the pulse shaping filter $C_0(t)$ can be expressed as

$$C_0(t) = \begin{cases} \prod_{i=0}^{3} S(t+iT), & \text{for } 0 \leq t \leq 5T \\ 0, & \text{else} \end{cases} \tag{2.1}$$

and the baseband signal becomes

$$y(t') = \sum_i \hat{s}_i \cdot C_0(t' - iT + 2T) \tag{2.2}$$

where $\hat{s}_i$ is the originally transmitted symbols. The time reference t' = 0 is the start of the active part of the burst. From the equation 2.1 and 2.2, it can be seen the current transmitted signal depends on the previous transmitted symbols, which leads to Inter-Symbol Interference.

2.3 GSM frame format

In GSM standard, each TDMA frame contains 8 time slots. The duration of one time slot is 0.577ms. Each time slot contains a normal rate burst of 156.25

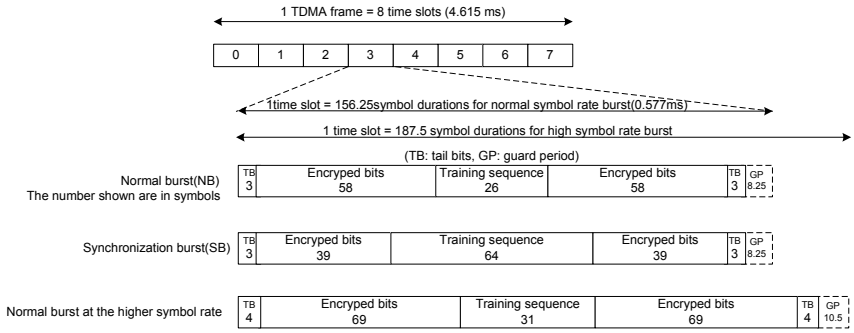

Figure 2.4: GSM burst format

symbols or a higher rate burst of 187.5 symbols. Different types of burst exist in the system. In this project, the Normal Burst and the Synchronization Burst are the burst types that need to be decoded. The burst format is shown below in figure 2.4.

As shown in the figure, the burst contains 4 parts: two tail-bits (TB) segments at each end of the burst, two segments of encrypted bits, a training/synchronization sequence, and a guard period (GP). The critical part of the burst is started and ended with tail-bits symbols, ie. 147 useful symbols for the normal symbol rate burst, and 177 for the high symbol rate burst. There are 3 tail-bits symbols in the normal rate burst, and 4 in the higher rate burst. The tail-bits are different in each modulation type. An overview of the tail-bits in each modulation type is shown in the table A.1 in Appendix A. The training sequence is a sequence of fixed bits which is used to train the receiver for e.g. noise-detection. The encrypted bits are the actual payload data, while the rest of the burst are coding overhead. In this project there is no special processing of the training/synchronization sequence, so they are considered as the normal bits. The guard period is a certain time duration which is inserted between burst to allow the time for the transmission signal to be attenuated.

2.4 Channel model

When a signal has been transmitted over a radio channel, it is often corrupted when it gets to the receiver. This is because the reflectors in the environment create a sequence of transmitted signal with attenuation, delay and phase shift. In wireless communication, the channel which the signal has been transmitted

over is often referred to as fading multipath channel. The noise presented in the channel is often assumed as Additive White Gaussian Noise(AWGN). Such a channel can be modeled as a tapped delay line with weight coefficients h which

$$h = [h_0, h_1, ..., h_{L-1}] \tag{2.3}$$

and added with a noise term z, as shown in figure 2.5. The h, also called channel impulse response, has finite length L. The length L is also known as the length of the channel memory. h_0 is often assumed to be unity, but in this project to be scaled to ensure the integer calculation.

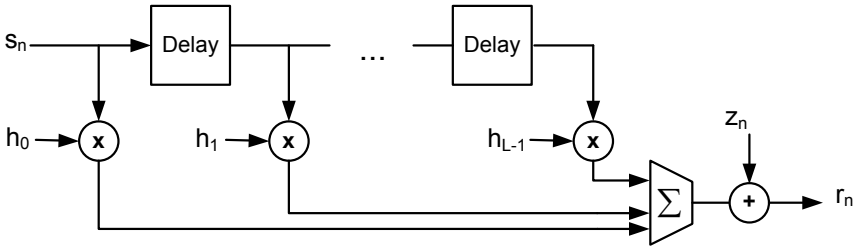

Figure 2.5: The model of a fading multipath channel

When a sequence of symbol s_n is transmitted over the multipath channel shown as above, the received signal will be the inner product of the channel impulse response with the L most recent transmitted signal, added with a noise term. A discrete-time model of the received symbols can be expressed as

$$r_n = \sum_{n=0}^{L-1} h_i \cdot s_{n-i} + z_n, \text{ where } n = 1, 2, .., N_{BL} + L - 1 \tag{2.4}$$

where r_n is the received symbols, s_n is the transmitted symbols, N_{BL} is the burst length, h_n is the channel impulse response and z_n is the complex Gaussian noise with zero mean and variance σ^2.

The GSM standard already specified several channel models, such as Typical Urban(TUx) and Hilly Terrain(HTx). The channel impulse response used in this project is HT0.

2.5 Equalizer

Due to the channel characteristics, severe intersymbol interference(ISI) may occur to the transmitted signals. Therefore it is necessary to use an equalizer to counter the effect of the channel. Figure 2.6 gives an overview of the system.

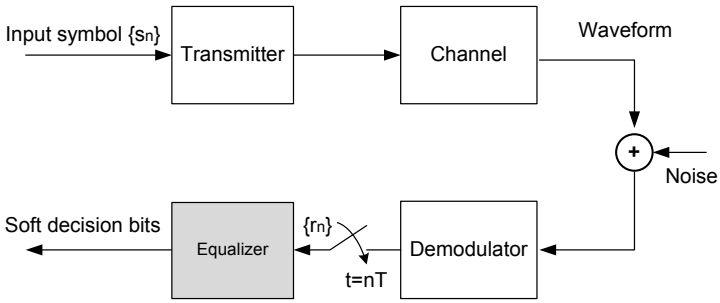

Figure 2.6: Overview of the transmission system

In this transmission system, the input signal bits are firstly mapped into complex-valued symbols s_n, according to its modulation type. The in-phase and quadrature components of the symbol will be multiplied with the high frequency carrier and be transmitted over the AWGN channel. On the demodulator side, the received signal is demodulated with correct carrier phase into its in-phase and quadrature components, and sampled with the correct timing intervals. The equalizer can be considered as a part of the demodulator. The input to the equalizer is the $\{r_n\}$ sequence, which represents the complex-valued symbols with ISI distortion, and the output is the soft decision result, which is the probability of each received bit being 1 or 0.

There are various approaches to implement the equalizer. In this project, the equalizer basically functions as a sequence estimator. Given the received sequence $r_n = s_n + z_n$, where s_n is the original sequence from the transmitter, and z_n is an additive white Gaussian noise process, the equalizer determines among the set C of all symbol sequences, the sequence $\{\hat{s}_n\}$ with minimum sum of squared errors (squared Euclidean distance) from $\{r_n\}$,

$$|r_n - \hat{s}_n|^2 = \min_{\{\hat{s}_n\} \in C} \sum |r_n - s_n|^2 \tag{2.5}$$

The optimal way to solve the problem is by using Maximum Likelihood Sequence

Estimation (MLSE) implemented with the Viterbi Algorithm (VA), which will be introduced in section 2.6. The computation complexity of this method increases exponentially with the size of the symbol in the modulation type. Many researches have been undertaken to find an algorithm with reduced complexity and without losing too much BER performance. In this thesis, three different algorithms have been implemented. These algorithms will be introduced in section 2.7, 2.8 and 2.9.

2.6 Maximum Likelihood Sequence Estimation (MLSE)

2.6.1 Euclidean Distance and Hamming Distance

A straight line between every two constellation points is called a *Euclidean distance*. It is same as the concept of *distance* we use in our daily life. Another distance term which appears often in the communication is the *hamming distance*. Hamming distance stands for the distance between binary sequence. Taking two binary sequence: 011000 and 100001. The distance between these two sequence is the number of bits these two sequence differ, which is 4 in this example. A zero hamming distance means the two sequence are the same, which has the same interpretation for the zero Euclidean distance.

When discussing the distance between sequences in the following sections, Euclidean distance are meant.

2.6.2 Trellis diagram

Trellis diagram is a method to present the state diagram with emphasis on the time progress of the state transition. Figure 2.7 is an example of a trellis diagram of a GMSK transmission.

In the trellis diagram, the time is expressed as the horizontal axis on the diagram. The four states S0-S3 in the left-most column are the states at time t=0. The states S0-S3 in the second left-most column are the states at t=1, etc. The states shown in the figure 2.7 are from t=0 to t=4, progressing from the left side to the right side. The states which have the same value on the x-axis belong to the same *stage*.

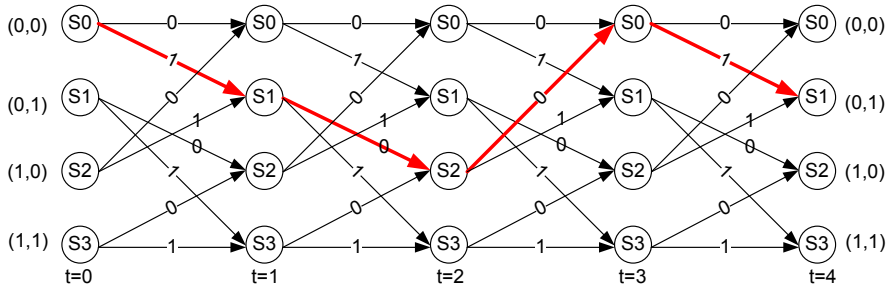

Figure 2.7: Trellis diagram of GMSK

There are four states in each stage, which are marked as S0(0,0), S1(0,1), S2(1,0), S3(1,1). The two number in the bracket indicates the two most recent transmitted symbols, with the left item being the oldest and the right item being the newest. The states representation in this GMSK modulation type example can be generalized as

$$State\_rep : (symb_1, symb_2, ..., symb_N) \tag{2.6}$$

$$and \quad symb_i = (b_{i1}, b_{i2}, ..., b_{iM}) \tag{2.7}$$

where $symb_i$ is the i^{th} symbol in the represented by the states, N is the number of symbols in the state memory, b_{ij} is the j^{th} bit in the i^{th} symbol and M is the total bits in one symbol. In this way, a total of 2^N states are present in each stage in the trellis diagram. For example, for an 8PSK 64 state trellis, the memory length N is 2, the bits per symbol M is 3. If one of the states represents ("001","100"), it means that two previous transmitted symbols of this state are symbol "001" followed by symbol "100".

Each state has 2^M possible edges from the previous stage, and 2^M possible edges to the next stage, where M is the bits per symbol. These edges are called *branches* or *transitions* in the trellis diagram. Each branch is labeled with the corresponding transmitted symbol. For the trellis diagram in this project, the transmitted symbols from the same stage are always arranged in their numerical order, e.g. in GMSK, the uppermost branch for symbol 0, the bottom branch for symbol 1.

Trellis diagrams are originally used in the binary convolutional codes to visualize the process of encoding and decoding. Here the trellis diagrams are modified such that the branch are labeled with modulation symbols instead of binary code symbols.

Path metric

With the time progresses, the branches are extended into *path*. The branches marked by red line in the figure 2.7 is a path. This path has the initial state of S0 and the symbol sequence of [1,0,0,1].

Each time the state forks into branches, the weight of each branch is calculated. The weight is the squared Euclidean distance of the received symbol to the expected symbol. The weight is also called a *branch metric*. The cumulative distance(sum of squared error) along a path is called a *path metric*. In figure 2.8 shows an example of the path metric. In this example, the initial state is assumed to be S0, the distance of the received symbol to symbol 0 is calculated to be 20, and the distance to symbol 1 is 2. Then, at stage t=1, four branches are calculated. And at stage t=2 and t=3, all eight branches are calculated for branch metric. This process is called *path extension*.

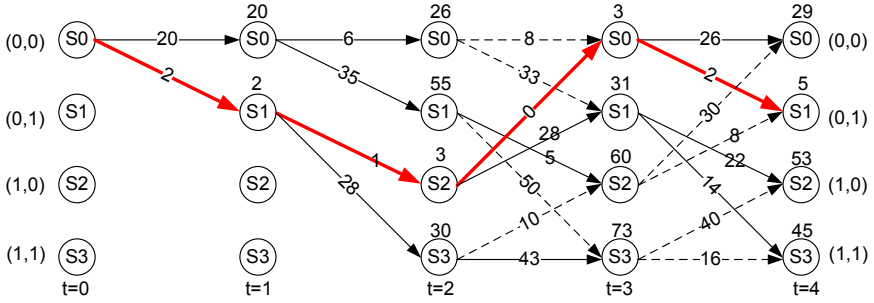

Figure 2.8: Trellis diagram of GMSK, with path extension and path merge

Path merge

In the example in figure 2.8, the eight branches from stage t=2 meets in pairs at stage t=3. When two or more paths meet at the same state, it is only necessary to keep track of the one with the smallest path metric, because this "winning" path will always have smaller path metric in the following stages. The paths with bigger path metric are rejected, which are shown as dashed lines in the figure 2.8. This process is called *path merge*. The remaining transition from the path merge process is called a *surviving* path. Only one path with the minimum path metric survives among all the possible paths that end at the same state, and the path metric of this surviving path is called the *state metric* of this

state, which is shown in the numbers above the each state in the figure 2.8. By using path merge mechanism, it is guaranteed that at any moment in the trellis progress, the number of surviving paths never exceeds the number of states in each stage. This process reduces the computation complexity significantly.

Surviving symbol memory and Decision feedback

Each state stores a path of length L with the minimum path metric which ends at this state. L is equal to the length of the channel impulse response, which in this project, is set to be 6. In the example above, at the stage t=4, the state S1 stores the path marked by the red line, which represented in symbols is [0,0,1,0,0,1]. This sequence of length L is refereed to as the *surviving symbol memory* (Surmem). It consists of 2 parts: the sequence [0,1] represented by the state S1, and a *decision feedback memory* [0,0,1,0]. The surviving symbol memory is used to calculate the channel reference.

The decision feedback memory, though also a sequence of stored symbols in each state, should not be confused with the sequence of symbols represented by the state, in that the former one is the most optimal paths of L stages ended at each state, while the later one is all the possible path happened in the previous N stages.

Soft decision bits

The aim of the trellis diagram is to find out what a transmitted symbol at a certain stage is, which more specifically is to decide each bit in the symbol being 0 or 1. A decision formed in a binary way of either '0' or '1' is called a *hard decision*. This is equal to making a 100% positive decision of a certain bit. Due to the present of noise, this kind of decision will cause irreversible loss of information in receiver. The remedy for this problem is by using soft decision, which instead of deciding directly the bit being '0' or '1', gives a "probability-like" measure of the bit being '0' or '1'. The soft decision for a bit sequence [1,1,0,0] could be [20,50,-10,-90], meaning that the first two bits are probably two one's, with the second bit more likely to be one than the first bit, and the last two bits probably zero's, while the last bit much more likely to be zero than the third bit. The soft decision used in this project is unquantized value, which is taken directly from the path metric. The soft-bit decision is used by the convolutional decoder following the equalizer.

The soft decision is made before the trellis search moves from one stage to

another stage. A temporary soft decision is calculated at each state, and the minimum soft decision value for each bit is chosen after all the state are calculated. The soft decision is made for the oldest symbol represented by the state, which is the transmitted symbol from the N^{th}-previous stage. And temporary soft decision is calculated by adding the state metric with the minimum branch metric to the next stage.

Taking the same example from figure 2.8, at stage t=3 state S0, the soft decision is made for oldest symbol represented by the state, which is symbol '0'. The temporary soft decision is the state metric of state S0 of 3, added with the current transition of the minimum error, which is 2, and the result is 5. So for the first state S0, the temporary soft decision made for symbol '0' is 5. Similarly, at state S1, the temporary soft decision for symbol '0' is 45. At state S2, the soft decision made for symbol '1' is 68. At state S3, the soft decision made for symbol '1' is 89. Then, the smallest temporary soft decision for each bit is chosen, which is 5 for symbol '0' and 68 or symbol '1'. Finally the result of the soft decision for symbol '0' minus the soft decision for symbol '1' is stored, which is -63 in this case. This number can be understood as: the transmitted symbol from the second previous stage (t=1) is more likely to be 0, with the confidence of the decision being -63. The more the number is to the negative side, the larger the probability the symbol being 0 is, and vice versa.

When a state in the trellis diagram represents N recent transmitted symbols, the soft decision at stage T can be made for the symbol at the previous (T-N) stage. The bigger the N is, the more stages the decisions-making stage are aparted from the calculated symbol's stage, and thus the more precise the result is, because it allows the error to be cumulated during the past N stage. However, the increase in N will cause the number of total states to grow exponentially.

Summary

By using trellis diagram, the problem of finding the sequence with the minimum error has become of finding the path with the minimum path metric. The recorded path has the finite length which is equal to the channel memory length. And the soft decision for a previous transmitted symbol is made at each stage. The trellis structure shown here is a full trellis structure, which covers all the possible situations of a sequence. This kind of trellis structure is also called a Maximum Likelihood Sequence Estimation(MLSE) trellis. A detailed description of how to calculate the path metric, merge the path and find the soft decision will be described below in the section 2.6.3.

2.6.3 The MLSE algorithm

The algorithm using the full trellis structure is described in this section. Most calculations in this section can also be applied to the algorithms described in the following sections 2.7, 2.8, and 2.9.

It has been described in the section 2.6.2 that each state has a surviving symbol memory $Surmem$ which contains the best path of length L, with L equal to the channel impulse response length. Assume that there are Q points in the constellation diagram and each state represents r previous transitions, there will be a total of P states in the trellis which $P = Q^r$. The surviving symbol memory for all the P states at stage x can be expressed as

$$Surmem_x = \begin{bmatrix} symb_{11} & symb_{12} & \cdots & symb_{1L} \\ symb_{21} & symb_{22} & \cdots & symb_{2L} \\ \cdots & & & \\ symb_{P1} & symb_{P2} & \cdots & symb_{PL} \end{bmatrix} \tag{2.8}$$

Where each row is a surviving symbol sequence for a state. Taking the surviving symbol memory of state n (the n^{th} row in the $Surmem$), and rewriting the symbol with their IQ coordinates, the estimated sequence of state n becomes:

$$\hat{S}(n) = [\hat{s}_{n1}, \hat{s}_{n2}, \cdots, \hat{s}_{nL}] \tag{2.9}$$

where $\hat{s}_{ni}$ is the complex representation of $symb_{ni}$. As described in equation 2.3, the channel impulse response is $h = [h_0, h_1, \cdots, h_{L-1}]$. Assume the current received symbol is r_x, which is a also complex number indicating the IQ coordinates of the received symbol. To counter the channel ISI effect, the inner product of the channel impulse response and the estimated sequence is calculated, and is subtracted from the received signal. The result is called a *decision variable* of the current state.

$$dec\_var(n) = r_x - \sum_{i=0}^{L-1} h_i \cdot \hat{s}_{n(L-i)} \tag{2.10}$$

Then the distance (squared error) of the decision variable to each point on the constellation diagram is calculated. Since there are a total of Q points, and the list of the IQ coordinates of all these Q points are

$$Ref = [ref_1, ref_2, \cdots, ref_Q] \tag{2.11}$$

where ref_i is the vector indicating the location of the constellation point on the complex plane. The error to each point can then be calculated as

$$Error(n) = [err_{n1}, err_{n2}, \cdots, err_{nQ}] \tag{2.12}$$

$$where \qquad err_{nj} = dec\_var(n) - Ref(j); \qquad j \in (1, Q) \tag{2.13}$$

Then, the cumulative metric is calculated by adding each error with the state metric of the current state $state\_met(n)$,

$$Cumu\_met(n) = [cm_{n1}, cm_{n1}, \cdots, cm_{nQ}] \tag{2.14}$$

$$where \qquad cm_{nj} = err_{nj} + state\_met(n); \qquad j \in (1, Q) \tag{2.15}$$

Since there are Q possible new transitions at each state, a number of Q cumulative metrics are calculated. A P*Q matrix of cumulative metrics will be obtained for P states, with each row representing one state, and each column representing a transition to a certain reference point:

$$Cumu\_met = \begin{bmatrix} cm_{11} & cm_{12} & \cdots & cm_{1Q} \\ cm_{21} & cm_{22} & \cdots & cm_{2Q} \\ \cdots & & & \\ cm_{P1} & cm_{P2} & \cdots & cm_{PQ} \end{bmatrix} \tag{2.16}$$

Now the metrics that end at the same state should be merged. As already discussed in section 2.6.2, for each state, there are Q transitions to the next stage, and Q transitions from the previous stage. For the convenience of explanation, the Cumu_met matrix is first transposed into 1D array column by column:

$$\begin{aligned} Cumu\_met\_1d &= [cm'_1, cm'_2, \cdots, cm'_{P*Q}] \\ &= [cm_{11}, cm_{21}, \cdots, cm_{P1}, cm_{12}, \cdots, cm_{P2}, \cdots, cm_{PQ}] \end{aligned} \tag{2.17}$$

Then the paths that need to be merged for state n' in the stage x+1 can be written as:

$$Merging\_met(n') = [cm'_{n'*Q+1}, cm'_{n'*Q+2}, \cdots, cm'_{n'*Q+Q}] \tag{2.18}$$

And the surviving metric can be found by selecting the minimum value in the Merging_met array. It should be noted that the state metric of the state n' in stage x+1 is the same as the surviving metric of the state n'.

$$Surviving\_met(n') = state\_met(n') = \min_{i \in (1,Q)} cm'_{n'*Q+i} \tag{2.19}$$

And if there exists I for which

$$cm_{n'*Q+I} = \min_{i \in (1,Q)} cm_{n'*Q+i} \tag{2.20}$$

The surviving transition $tr_{n'}$ is assigned the value of I. The surviving transition for all P state in the state x+1 is

$$Surviving\_trans = [tr_1, tr_2, \cdots, tr_P] \tag{2.21}$$

Combining with the matrix in 2.8, the new surviving symbol memory can be obtained for each state in the stage:

$$Surmem_{x+1} = \begin{bmatrix} tr1 & symb_{11} & symb_{12} & \cdots & symb_{1(L-1)} \\ tr2 & symb_{21} & symb_{22} & \cdots & symb_{2(L-1)} \\ \cdots & & & & \\ tr_P & symb_{P1} & symb_{P2} & \cdots & symb_{P(L-1)} \end{bmatrix} \tag{2.22}$$

The initial value of the Surmem can be chosen randomly, as long as the value matches the state representation. The initial value of the state metric should be set in a way that the initial state has the state metric of zero, and all the other states have the state metric of a maximum value.

Finally, the soft decision should be calculated. Taking the minimum cumulative metric of each state (row) of $Cumu\_met$ from equation 2.16,

$$m\_cm_n = \min Cumu\_met(n) = \min_{i \in (1,Q)} cm_{ni} \tag{2.23}$$

For all P state, the minimum cumulative metric is

$$M\_cm = [m\_cm_1, m\_cm_2, \cdots, m\_cm_P] \tag{2.24}$$

The soft decision is made for each bit in the symbol. Supposing a total of B bits in each symbol, and $2^B = Q$. The minimum cumulative metric is firstly extended to B rows:

$$M\_cm\_2d = \begin{bmatrix} m\_cm_{11} & m\_cm_{12} & \cdots & m\_cm_{1P} \\ m\_cm_{21} & m\_cm_{22} & \cdots & m\_cm_{2P} \\ \cdots & & & \\ m\_cm_{B1} & m\_cm_{B2} & \cdots & m\_cm_{BP} \end{bmatrix} \tag{2.25}$$

where each row in the matrix is same as M_cm. The soft decision is made for the previous r^{th} transition, which is stored in the $symb_{nr}$ of state n. If the b^{th} bit in $symb_{nr}$ is 0, we define $m\_cm_{bn} \in g0$, otherwise $m\_cm_{bn} \in g1$. The soft decision for the b^{th} bit in the symbol can be expressed as:

$$SoftDec_b = \min(m\_cm_{bi} \in g0) - \min(m\_cm_{bj} \in g1) \tag{2.26}$$

The soft decision is made for all the bits in the symbol.

2.7 Reduced-state sequence estimation with set partitioning and decision feedback (RSSE)

2.7.1 Set-partitioning

Set-partitioning is proposed by Gottfried Ungerboeck in his publications [9] and [10]. The general idea of the set partitioning is to group constellation points into subsets, so that the Euclidean distance among the points in the same subset is maximal.

When the constellation points are set partitioned, the points in a same subset are the points that are most unlikely to be mistaken with each other. Since the most errors are made by picking the neighboring points, rather than the far-away points, it will be easier to make a decision among the points that are far-away from each other first. It will be described in the following section that when combining with the trellis diagram, the set partitioning helps to reduce the computation complexity.

As described in his publications, Ungerboeck applied the set-partitioning on the Trellis coded modulation(TCM), which is a combination of convolutional coding scheme and modulation, to achieve an improved coding gain. It should be noted that in this project, although the set partitioning concept has also been applied together with the trellis diagram, the algorithm has no relation with TCM.

Below are the figures 2.9, 2.10, 2.11 showing how the set partitioning are applied in the 8PSK, 16QAM, and 32QAM modulation schemes. In this project, the constellation points are partitioned into 4, 8 and 8 subsets for the 8PSK, 16QAM and 32QAM respectively. It is possible to make further partitioning in 32QAM.

2.7.2 The RSSE algorithm

The algorithm of reduced-state sequence estimation with set partitioning and decision feedback is proposed by Eyuboglu et.al. in [4]. This algorithm applied set-partitioning on the trellis diagram to achieve reduced complexity in computation.

The basic concept in this algorithm is that in the trellis structure, when using the state to represent the previous transmitted symbols, the state represents the sequence of the subsets instead of the sequence of the symbols. Taking 8PSK as an example. The 8PSK is set-partitioned as in figure 2.9, and the constellation

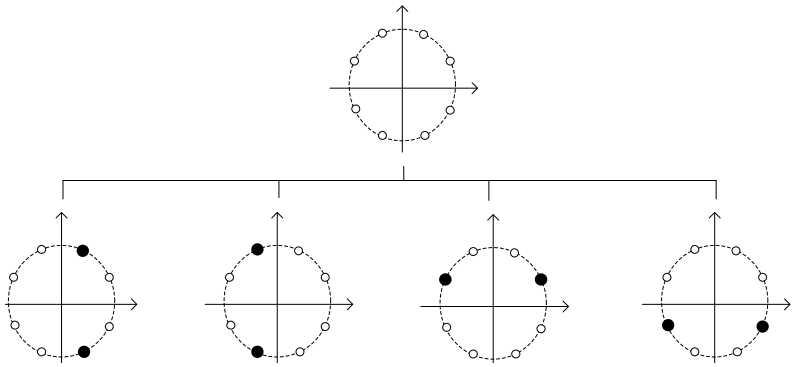

Figure 2.9: Set-partition of 8PSK, 4 subsets

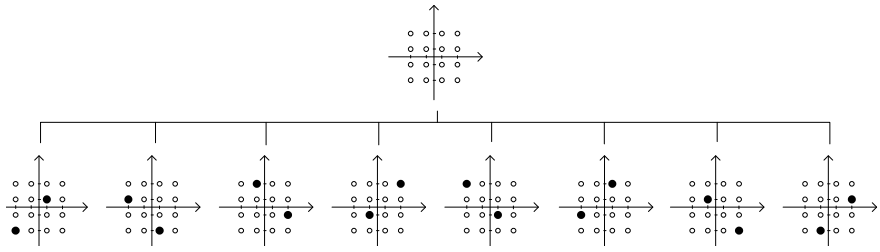

Figure 2.10: Set-partition of 16QAM, 8 subsets

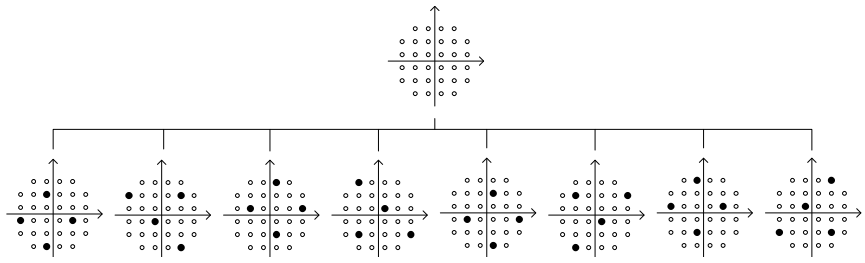

Figure 2.11: Set-partition of 32QAM, 8 subsets

points are numbered as in figure 2.2. The 8PSK constellation points are grouped into 4 subsets: 0/4, 1/5, 2/6, and 3/7. The trellis diagram shown in the left side of figure 2.12 is a MLSE trellis structure for 8PSK, with each state representing the previous two transmitted symbols. In the right side of figure 2.12, an RSSE

trellis is shown. Here each state represents the previous two subsets which the transmitted symbols belong to. S0 represents (0/4, 0/4), S1 represents (0/4, 1/5), etc. This can be interpreted as S0 has the oldest transmitted symbol being 0("111") or 4("101"), and the newest transmitted symbol being 0("111") or 4("101"). S1 has the oldest transmitted symbol being 0("111") or 4("101"), and the newest transmitted symbol being 1("011") or 5("001"), and so on.

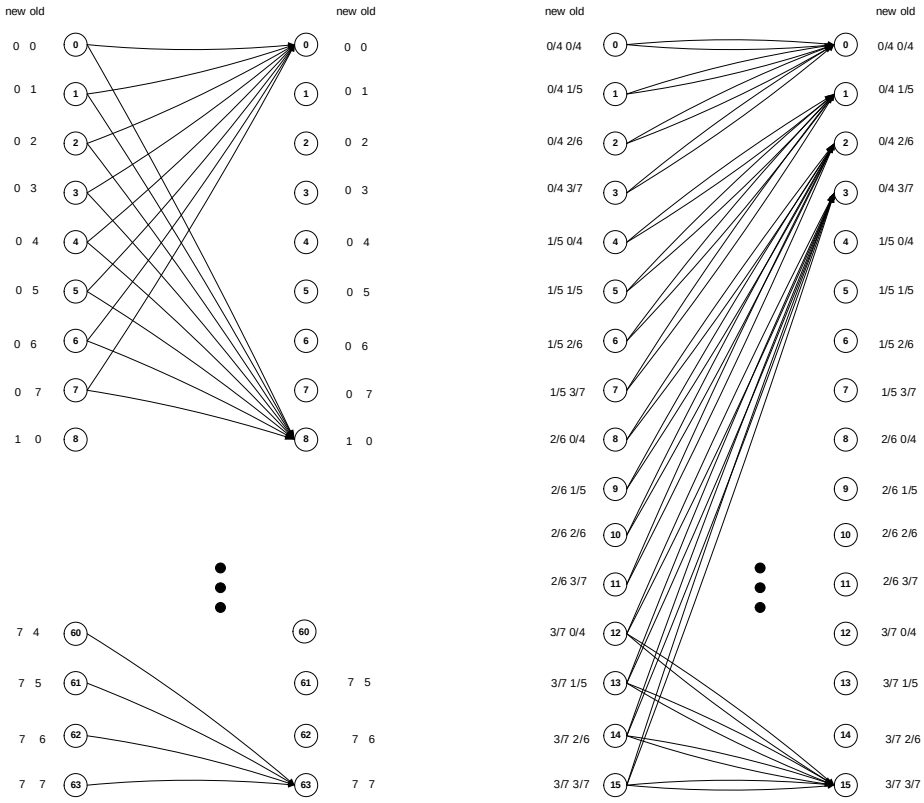

Figure 2.12: Left: full trellis for 8PSK, Right: RSSE trellis for 8PSK

In this example, the state S0(0/4, 0/4) is the combination of the four states in the MLSE trellis: S0(0,0), S4(0,4), S32(4,0), and S36(4,4). In this way, the 64 states in the MLSE trellis are reduced to 16 states. The effect of combining these four states into one is the same as merging the paths leading to these four states into one. This approach is feasible because point 0 and point 4 are almost in the opposite direction, thus a received point can only be mistakenly decided when the error power is high. Therefore the shortest path can be reliably distinguished and preserved.

Since the state represents more combination of symbols, more than one transition have the same source state and destination state. In the example above, there are two transition from S0 to S0: symbol 0 and symbol 4. The transition which has the same source and destination is called a *parallel transition*. The transitions shown in the RSSE trellis are also arranged in the numerical order: transition of a smaller symbol on the top and transition of a bigger symbol at the bottom.

Since the number of states decreases in the RSSE trellis, the number of transitions in the RSSE trellis decreases too. A total of 512 transitions exist in each stage in MLSE trellis, while only 128 transitions exist in the RSSE trellis. This will reduce the calculation in the path extension and path merge process to approx. 25%. In this way, the algorithm can reduce the power consumption significantly, at the cost of the loss of BER.

In the larger alphabet modulation schemes, the set-partitioning can reduced the computation complexity even further. For 16QAM, the MLSE trellis needs 256 states to achieve a memory length of 2, and for 32QAM 1024 states. While using set-partitioning, the constellation points are grouped into 8 subsets with 2 points in each subsets for 16QAM, and 8 subsets with 4 points in each subsets for 32QAM. The RSSE trellis only needs 64 states to achieve the same memory length for both modulation schemes.

Most of the calculation in section 2.6.3 still can apply on the RSSE algorithm. One of the most distinctive change is that since there are parallel transitions, a new variable should be used to store the minimum metric of the parallel transitions. It will be used in the merge process.

2.8 Sphere detection

Sphere detection(SD) is another low power algorithm used to find the MLSE solution. It is proposed by Hassibi et.al. in the publications [5] and [6].

Like the RSSE algorithm, the author of the SD algorithm also tries to reduce the number of paths by reducing the amount of states. One state in the trellis search will lead to N calculation of error in the path extension process, and N comparisons in the path merge process, where N is the number of the symbols in the constellation. When the state metric of a certain state gets to a certain big value, it is almost apparent that the path towards this state can not be the "winning path". If these state can be reliably detected and safely removed, the system can avoid much unnecessary calculation in the trellis search.

The principle behind the Sphere detection is simple. The algorithm only searches a limited number of points inside a circle around the received point in the IQ signal space. Since the algorithm can be used in the multi-dimensional search space, the circle will become a hyper-sphere in the multi dimensional space, thus it is named "sphere detection".

When applied together with the trellis search, the algorithm corresponds to setting a threshold on each transition. Only the transition with a branch metric smaller than the threshold will survive, otherwise it will be removed. If no surviving transition ends at a certain state, this state is *pruned*. Figure 2.13 shows a trellis search using the SD algorithm in a GMSK transmission.

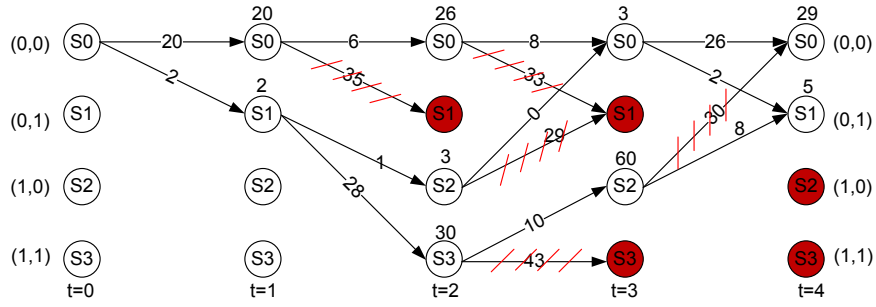

Figure 2.13: Trellis search using Sphere detection algorithm in a GMSK transmission. Paths with metric over 28 are pruned.

In this example, the threshold is set to be 28, which means all the transitions with error bigger than 28 will be removed. In the stage t=1, the transition from state S0 to state S1 in the next stage has an error of 35, and it is removed, as crossed by red lines in the figure. Since there is no surviving transition to state S1 in the stage t=2, the state is pruned, as marked by a red fill. When the trellis search progresses to stage t=2, no path extension is made from state S1. Similarly, the states S1 and S3 at t=3, and S2 and S3 at t=4 are pruned.

It is apparent that the choice of the radius of the sphere, or the threshold, can have a great impact on the complexity of the algorithm. If the threshold is too low, there will be no surviving path. If too high, almost all states will survive, and the complexity will still be high. Since the comparison with the threshold is also expensive in the power consumption, the threshold should be set in a way that it should in the worst case counter the power spent on the comparison.

Furthermore, the threshold value also depends on the Signal-To-Noise ratio of

the burst. And SNR can vary a lot from burst to burst. So a threshold value depending on the SNR of the current burst is more preferable than a constant value. There are several ways to find the threshold value, such as by analyzing the pre-defined training sequence and find the SNR, or using one state trellis search method to find the upper bound of the error. The later method is to check the burst through by a one-state trellis structure to find the maximum error of the current burst, and use this error with a scale factor as the threshold value. Since the simulation result shows that it is a simple and efficient way to find the maximum error of the burst, this method is chosen to find the threshold in this project.

Since the SD algorithm makes the trellis structure more flexible, it is also necessary to store the surviving elements in a dynamic way, as not to consume too much memory. The book-keeping mechanism is also a challenging part. If not carefully designed, the module can easily consume more power than necessary.

The original SD algorithm also proposed some other interesting issues to improve the trellis search, such as using sorting mechanism to find the best searching sequence of the states, and limiting the total number of searching states etc. Due to the limitation of the hardware, this project only implements the simple SD algorithm.

The SD algorithm uses a MLSE trellis structure. The algorithm in section 2.6.3 should be modified in the following way:

- A new list should be created to store the surviving status of the state.
- The surviving transition from the merge process should be compared with the threshold. And if the transition exceeds the threshold, the state the transition goes to is pruned.
- If the state is pruned in the previous trellis stage, no calculation should be performed for this state.
- In the merging process, the value of the transition from the pruned state is set to be maximal value.
- Only the initial state is calculated in the first stage of the trellis.

2.9 Using RSSE in combination with Sphere detection

As mentioned earlier, the RSSE algorithm reduces complexity by modifying the trellis structure, while SD algorithm sets a constraint on the computation. They both seem to be promising in making the trellis search more computational efficient. Since these two algorithms focus on different aspects on improving the performance, and their approaches do not conflict with each other, a hybrid algorithm which explores the RSSE trellis with an SD constraint might reduce the computation even further. However, by doing combining these two algorithms, it might not give the BER-optimal solution found by the MLSE trellis. Since the RSSE trellis has much fewer states and surviving paths than the MLSE trellis, it has bigger probability of pruning too many states and thus leads to no surviving path in the trellis search. Although the system will resolve the situation by increasing the threshold and restarting the search process from the beginning of the burst, the system becomes very inefficient and non-deterministic. Thus this situation should be avoid to a feasible extent. Therefore the choice of the threshold has even bigger influence on the performance. Furthmore, there are some issues that need to be concerned:

1. Will this algorithm be more power efficient than the RSSE and the SD algorithms?
2. How much more hardware will it take to implement the hybrid algorithm, compared with the RSSE and the SD algorithms?
3. How much BER will the algorithm lose?

The first two issues are the main questions that this project needs to answer. Since this project focuses on the hardware implementation, the BER performance of the algorithms is only shortly examined. Several matlab simulations show that the hybrid algorithm only loses very little BER performance comparing to RSSE. However, the BER analysis doesn't reach the maturity to be presented in the thesis.

In the following chapters, this algorithm will be refereed to as RSSE_T algorithm.

Chapter 3

Design Specification

In this chapter, the hardware implementation of the equalizer is described. The equalizer is implemented using three algorithms: RSSE, SD and RSSE_T, which are described in section 3.4, 3.5 and 3.6 respectively. The basic MLSE algorithm has not been implemented, since the computation cost is too high. When implementing the SD algorithm, two different approaches are used. Although having the same function and the same soft-bit output, the two approaches have significant difference in the area and power consumption. These two different approaches are described in section 3.5.1 and 3.5.2 in details.

3.1 Requirements

In this section, the functional requirements are specified. It should be noted that although different algorithms are used to implement the equalizer, the functional requirements are the same for the design.

3.1.1 Top-level Overview

The equalizer should be able to process a total of five modulation schemes: GMSK, 8PSK, 16QAM, 32QAM and QPSK. The GMSK and 8PSK modulation schemes are of normal symbol rate, and the burst length is 143 symbols. This length does not include the two tail-bits symbols at the start of the burst, and the three tail-bits symbols at the end of the burst. The two tail-bits symbols at the start are used to determine the initial state of the trellis structure, and the three tail-bits symbols at the end of the burst are unnecessary to be processed. For the 16QAM and 32QAM modulation types, the bursts are of either normal rate or high rate, which means the burst length can be either 143 for the normal rate burst or 171 for the high rate burst. For the QPSK, the burst is of 171 symbols.

The state in the trellis structure should represent at least two transitions in each modulation schemes, as to make the results more immune to noise. The state in GMSK should represent 4 or 5 most recent transitions, the state in 8PSK should represent 2 or 3 most recent transitions, in 16QAM and 32QAM 2 most recent transitions, and in QPSK 3 most recent transitions. Different algorithms will have different numbers of total states in the trellis structure. According to the requirements stated above, the table 3.1 shows the total states used in each algorithm.

Modulation type	RSSE	SD	RSSE_T
GMSK	16/32	16/32	16/32
8PSK	16/64	64	16/64
16QAM	64	256	64
32QAM	64	1024	64
QPSK	64	64	64

Table 3.1: Number of total states in three different algorithms by the modulation types

The modulation schemes, the burst type and the total state of the current burst is configured by the DSP unit, which acts as the master in the system. The DSP unit configures the equalizer unit by setting a operation mode control byte(OpMode) stored at a specific address in the equalizer's memory module. This is done before the DSP unit writes the burst data into the equalizer's memory module. The operation mode control word OpMode is of 8-bit, and is encoded as in figure 3.1. The first three bits (MT) define the modulation type, the next three bits (TST) define the total state number, and the last two bits

(TSB) define the total symbols in the burst. The encoding of the operation mode byte is shown in table 3.2, 3.3 and 3.4.

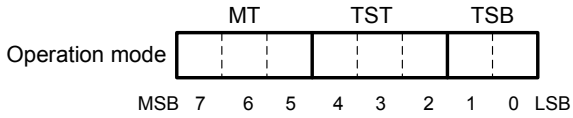

Figure 3.1: The operation mode control byte

OpMode[7:5]			Modulation type
0	0	0	GMSK
0	0	1	QPSK
0	1	0	8PSK
0	1	1	16QAM
1	0	0	32QAM

Table 3.2: Encoding of the MT (modulation type), OpMode[7:5]

OpMode[4:2]			Total number of state
0	0	0	16
0	0	1	32
0	1	0	64
0	1	1	256
1	0	0	1024

Table 3.3: Encoding of the TST (total number of state), OpMode[4:2]

OpMode[1:0]		Total symbols in the burst
0	0	143
0	1	171

Table 3.4: Encoding of the TSB (total symbols in the burst), OpMode[1:0]

The equalizer unit processes one burst at a time. As mentioned above, the DSP unit first sets the operation mode control byte for the burst. Then the whole burst is transmitted to the equalizer unit and stored in the memory of the equalizer unit. After the entire burst is placed into the memory, the equalizer starts to perform sequence estimation according to the algorithm by the trigger of DSP unit. The output from the equalizer, which is the soft-bit decision result for each sample in the burst, will be stored into the memory first. When all the samples in the burst are processed, the equalizer will send an interrupt signal to inform the DSP unit that the calculation is finished. After the DSP unit fetches

the soft-bit result, it will set the operation mode byte for the next burst and send the next burst into the equalizer unit. The interface between DSP unit and the equalizer unit is compatible with the Open core protocol (OCP) interface standard.

3.1.2 Equalizer core

Introduction
This sub-unit is the core of the equalizer unit, which estimates the maximum likelihood sequence by using one of the three different computational efficient algorithms: RSSE, SD and RSSE_T. The equalizer core processes data in burst, which means it will not process a new burst until the current burst is finished, even if the DSP unit sends a new burst to equalizer during the processing, which is considered as an error situation. The OpMode byte and some other parameters, such as the threshold value for pruning the transitions, are read by the equalizer core in the beginning of a burst. Any change of these parameter during the process of a burst will not be read by the equalizer, thus will have no effect on the result.

In the SD and RSSE_T algorithms, the equalizer should also be able to function as a one-state equalizer to calculate the threshold of the current burst.

Input
The data input to the equalizer is the I and Q projection of the symbols in a burst. The amplitude of the I and Q parts are scaled to ensure the integer computation. The unit circle for the GMSK, QPSK, 8PSK is scaled by 512, and each grid in the 16QAM and 32QAM is scaled by 128. Considering the channel effect and a reasonable noise amplitude, the input data length is selected to be 12 bits, which represents a range of [-2048,2047] in two's compliment format. The data are stored in the memory by the DSP unit. For the same symbol, the I part is one address prior to the Q part in the memory. The operation mode and parameters of the burst are also stored in the memory, thus they are read by the equalizer as the data input.

The control input includes a pin to start the equalizer, and a pin to switch the equalizer to function as a one-state equalizer to calculate the threshold. These two input pins are controlled by the DSP unit via OCP interface.

Processing
The equalizer is started when the "start" pin is asserted by the DSP. For a RSSE equalizer, it reads the operation mode from the RAM, and decodes the modulation type, total number of states and total numbers of samples in the

burst. The equalizer then sets the initial state and reads the initial surviving symbol memory (Surmem) from a certain address in the ROM, according to the modulation type and the total states. Then the trellis search is performed on each sample. This process includes the following steps for each sample:

1. Read the I and Q of a sample in the RAM and the Surmem of all the states.
2. Extend and merge the path by performing the trellis calculation according to the algorithm used by the equalizer.
3. Calculate the soft-bit decision at the end of each stage and write the results to the memory.
4. Update the Surmem in the memory at the end of the stage.

The above actions are repeated for each sample in the burst until all the samples are calculated.

The SD equalizer and the RSSE_T equalizer differ slightly from the RSSE equalizer. Firstly, the ROM is not used, since only the Surmem of one initial state is stored. Secondly, after the path extension and merge process of item 2, the path that has a path metric higher than the threshold is pruned.

When the entire burst has been processed, the equalizer core asserts a "done" pin to indicate the calculation is finished. And the registers in the equalizer core is cleaned up.

Output
Output data is soft bit decision in 16 bits. The soft-bit decision is made for each bit in the symbol, which means there is one soft-bit result for each GMSK symbol, two soft-bit results for each QPSK symbol, three for 8PSK symbol, four for 16QAM symbol and five for 32QAM symbol. The range of soft-bit result is of [-32768,32767]. The output data is stored in the RAM to be read by the DSP unit, with the soft-bit for the LSB of the symbol first, and MSB of the symbol last.

3.1.3 Data storage

As mentioned above, the input and output data is stored in the memory of the equalizer unit. The memory module is implemented as a RAM. The RAM has the data port of 16 bits, which matches the burst data and the soft-bit

result's width. The burst that needs the largest storage is a 32QAM burst in high symbol rate, which has a total of 2*171 = 342 input symbol and 5*171 = 855 output soft-bit word. Although the input data is of 12-bit length, each of them is still placed into a RAM address of 16-bit word for the convenience of memory access, even though only the 12 least significant bits are used. Each of the output data also occupies one address in the RAM, and that will make a total of 342+855 = 1197 addresses. The data RAM is also used to store the state metric of each stage in the trellis search, of which the maximal number of states is 1024. Each of the state metric also occupies one address. Thus the minimum required address is 1197+1024 = 2221. The RAM has therefore been generated by the PureView tool as a 12-bit address port RAM, as to be able to store 4096 words of 16-bit. The memory allocation of the data RAM is shown in figure B.1 in appendix B.

Another RAM is used to store the surviving symbols(Surmem). This RAM is implemented as a 11-bit address and 32-bit data Dual port RAM. For each state in the trellis structure, the Surmem are six elements of 5-bit each, which makes a total width of 30 bits. Therefore the data port of this RAM is selected as 32-bit, as to be able to place the Surmem of the same state at the same RAM address. This allows the Surmem of the same state to be accessed in one clock cycle. While computing the Surmem of a new stage, the Surmem of the previous stage is required. And the access sequence is not in the numeric order. Therefore RAM contains both the Surmem of a previous stage and the Surmem of a new stage. The maximal number of Surmem is thus the maximal number of states multiplied by 2, which is 1024*2=2048. This makes the address port of the RAM to be 11-bit. Furthermore, it is decided to use the Dual port RAM. This is because that calculation of the new Surmem can be very time consuming when there are many states, and it is preferable to have the read-modify-write action to be done in one clock cycle. This can be accomplished by using Dual port RAM and pipeline access.

The Surmem RAM is separate from the data RAM for two reasons:

- The width of the data port is different.
- The Dual port RAM is more expensive in area and power consumption, thus it should be kept as small as possible.

For RSSE equalizer, a ROM structure is also required to store the initial value of the Surmem. This ROM has a data port of 32-bit width, which is the same as the Surmem RAM. And the ROM contains the Surmem of following trellis structures:

- 16-state GMSK
- 32-state GMSK
- 16-state 8PSK
- 64-state 8PSK/QPSK
- 64-state 16QAM
- 64-state 32QAM

From the list above, a total of 256 states of Surmem are stored in the ROM structure. Therefore the width of the address port should be defined as 8. The memory allocation of the ROM is shown in figure B.1 in appendix B.

Other variables and buffers used in the trellis structure are stored either as constant (such as the channel impulse response, set-partitioning), or as register files.

3.1.4 Interface design

The equalizer unit interfaces with the DSP unit by Open Core Protocol (OCP) interface. The interface implements the basic OCP interface, and it is shown in figure 3.2. Below is a description of each signal in the interface. Most of the definition is taken from [13].

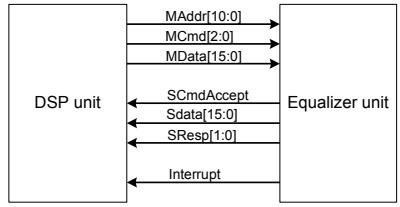

Figure 3.2: The interface between the DSP and the equalizer unit

MAddr: This is the transfer address set by the DSP unit. The width is configured to be 11 to be used according to the data RAM width.

MCmd: This is the transfer command set by the DSP unit. This signal indicates the type of OCP transfer the DSP unit is requesting. The commands used in the project are encoded as in table 3.5.

MCmd[2:0]			Command
0	0	0	Idle
0	0	1	Write
0	1	0	Read

Table 3.5: Encoding of the MCmd

MData: The write data from the DSP unit to the equalizer unit. The width of the MData is configured to be 16 to match the data RAM width.

SCmdAccept: Slave accepts transfer. A value of 1 on the SCmdAccept signal indicates that the equalizer unit accepts the DSP unit's transfer request.

SData: This signal carries the requested read data from the equalizer unit to the DSP unit. The width of SData is the same as the MData.

SResp: This is the response field from the equalizer unit to a transfer request from the DSP unit. The response encoding is in table 3.6 as follows.

SResp[1:0]		Response
0	0	No response
0	1	Data valid / accept
1	0	Request failed
1	1	Response error

Table 3.6: Encoding of the response SResp

Interrupt: This signal is not a part of the OCP protocol. It is set by the equalizer unit. A value of 1 on the interrupt signal indicates that the processing current burst is finished, and the output data is ready to be fetched.

Figure 3.3 illustrates a simple write and a read transfer on the OCP interface.

The sequence shown in the figure is described as follows:

1. On clock 1, the master (DSP) starts a request by set the MCmd field to Write (WR). At the same time, it presents a valid address (A1) on MAddr and valid data (D1) on MData. The slave (equalizer unit) asserts SCmdAccept in the same cycle.
2. On clock 2, the equalizer captures the values from MAddr and MData and uses them internally to perform the write.
3. On clock 4, the DSP starts a read request by set the MCmd to Read (RD). At the same time, it presents a valid address on MAddr. The equalizer unit asserts SCmdAccept in the same cycle.
4. On clock 5, the equalizer unit captures the value from MAddr and uses it internally to determine what data to present. The equalizer unit starts the response by switching SResp from NULL to DVA (Data valid), and then drives the selected data on SData.
5. On clock 6, the DSP recognizes that the SResp indicates data valid and captures the read data from SData. This transfer has a request-to-response latency of 1.

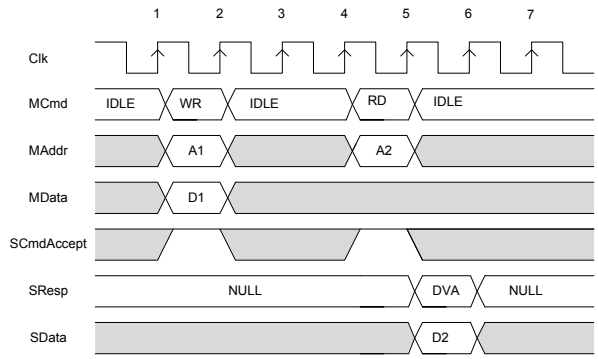

Figure 3.3: The waveform of the OCP interface signals

3.1.5 Timing requirements

The timing requirement is that the equalizer should be able to process a burst (at normal symbol rate or high symbol rate) in 1/2 GSM slot time, which is $0.577/2 = 0.288$ ms, with some margin. The maximum possible clock frequency is 200MHz.

3.1.6 Evaluation Focus

The focus of the project is to compare the power consumption and area cost, as well as the feasibility of the equalizer cores of various algorithms. The bit error rate of those algorithms is not a concern in this project.

3.2 Structure Overview

In this section, the structure of the top level equalizer is described. As mentioned in section 3.1, the top level structures of the equalizer of the three algorithms are very similar. The only difference is that the RSSE algorithm uses a ROM module, while the other two algorithms don't. This section introduces the design of the OCP interface controller, which is used in all three algorithm. The RSSE, SD, and RSSE_T main algorithms are implemented in the equalizer core unit, which will be described in the corresponding sections.

The structure of the equalizer unit is shown in figure 3.4.

As shown in the figure, the equalizer unit's top level consists of a OCP interface controller, a equalizer core, a data RAM, a Surviving symbol memory(Surmem) RAM, and a ROM.

The OCP interface controller allows the DSP unit to communicate with the rest of the equalizer unit. The OCP interface controller has access to the data RAM, and it handles all the memory transactions between the DSP and RAM.

The controller interfaces with the equalizer core by three control signals *start*, *done*, and *one_state*. When the *start* signal is asserted, the equalizer core starts to process the input data. When the equalizing process is finished, the equalizer core asserts the *done* signal. The *one_state* signal is needed to switch the equalizer core's mode between normal equalizer (when '0') and one-state equalizer (when '1').

The equalizer core implements the algorithm to perform the sequence estimation. It also has access to the data RAM, since it reads the input burst data from the data RAM and writes the output soft-bit to the RAM again. The equalizer core has also access to the Surmem RAM, which stores the surviving symbols for the trellis search. A ROM structure is connected to the equalizer core. The ROM stores the initial Surmem elements.

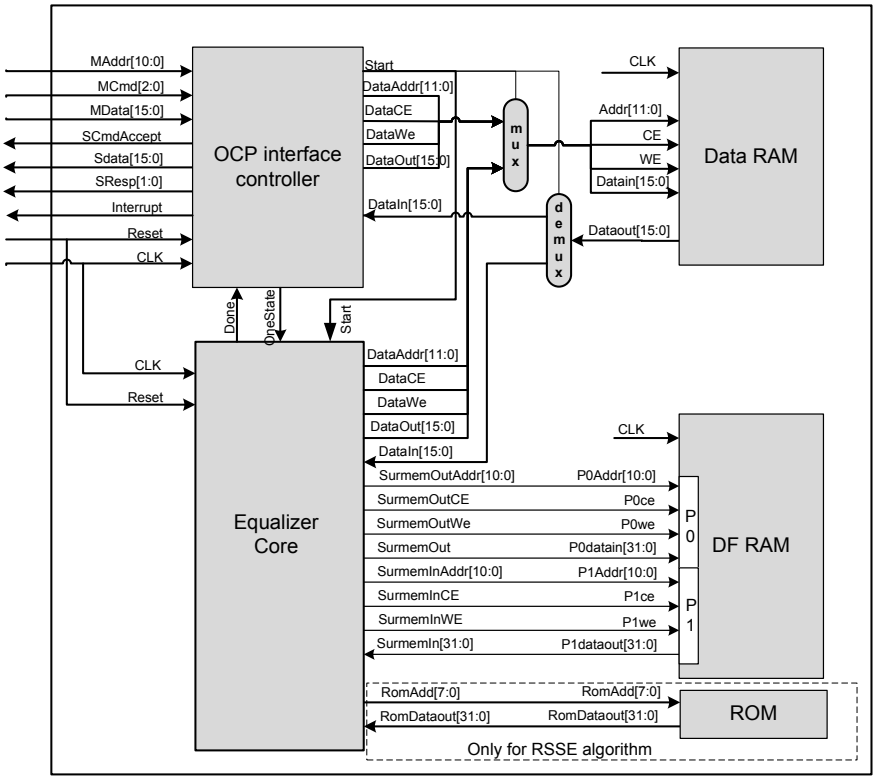

Figure 3.4: The structure of the equalizer unit

The data RAM is a single port synchronous RAM of 64kb. The Surmem RAM is a dual port synchronous RAM of 64kb. These two RAMs are not physical cells, but simulation models generated by Denali Software Pureview instead. The ROM is of 8kb, and is implemented as a constant array. The ROM is not needed in the SD and RSSE_T algorithms.

3.3 OCP interface controller

As mentioned in the section 3.2, the OCP interface controller is the interface between the DSP and the data RAM, as well as the interface between the DSP and the equalizer core. It performs the arbitration of data RAM access between DSP and equalizer data path. Besides, the controller also controls the sequence

of processing a burst.

The OCP interface standard has been described in section 3.1.4. Besides the OCP interface, the controller should also interpret the request from the DSP internally, and gives data RAM access to DSP.

The physical interface is implemented in the following way: The MData, SData and MAddr is connected directly to the DataIn, DataOut and DataAddr of the RAM. And the controller controls the RAM access by switching the CE and WE signals.

The interpretation of the DSP request is stated below, and an example showing the waveform of the interface between DSP and RAM is in figure 3.5.

- When the MAddr is 0x00, and MCmd is "Write", the MData contains an command word that will be written into a internal command register in the controller (Cmd_reg[7:0]). Only the two LSB's of the command is used. A '1' of Cmd_reg[0] sets the equalizer unit in slave mode and the DSP has full access to the data RAM. A '0' of Cmd_reg[0] means the DSP releases the control of the data RAM. The equalizer then works in the master mode and the equalizer core may access the data RAM. A '0' of Cmd_reg[1] bit switch the equalizer core to be the one-state equalizer, while a '1' sets the core to function as a normal equalizer. This command from the DSP is stored in a register, and will not be changed until the DSP sends a new command at MAddr 0x00. This situation is shown in figure 3.5 clock 1 and 6.

- When the MAddr is other than 0x00, and the MCmd is "Write", the controller should write data to the RAM. It simply asserts both the CE and the WE signal of the RAM. Since the MAddr and MData is connected directly to the address bus and data_in bus of the RAM, the data on the MData bus will be written immediately to RAM at the address on the MAddr bus. This situation is shown in figure 3.5 clock 2 and 3.

- When the MAddr is other than 0x00, and MCmd is "Read", the MAddr contains the address of the RAM the DSP unit tries to access. The controller should immediately perform an Read action of the RAM by asserting the CE signal. After the latency of the read action of RAM, the RAM drives the required data on the data bus, which is connected directly to the SData bus. This situation is shown in figure 3.5 clock 4 and 5.

The sequence of processing a burst is controller by the OCP interface controller. The controller should

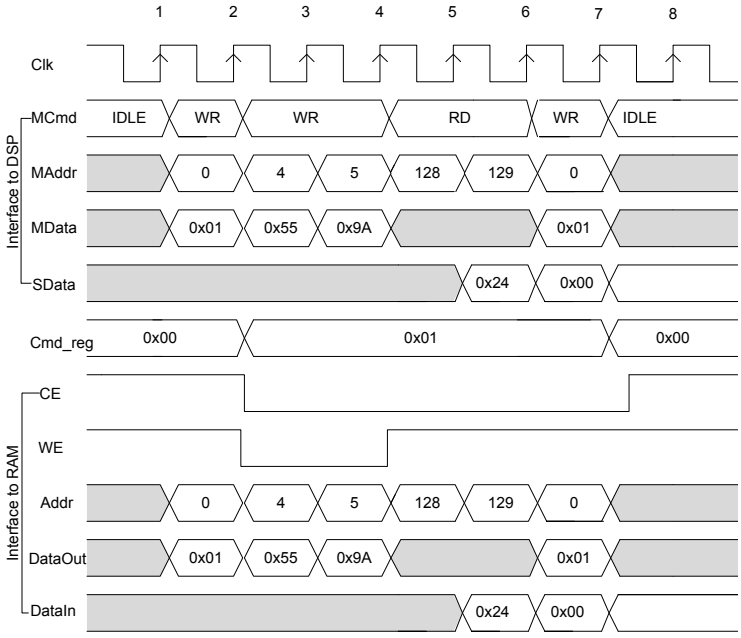

Figure 3.5: The waveform of the controller interfacing DSP to RAM

1. Store the input burst data from DSP into RAM.
2. If the threshold needs to be calculated, switch the equalizer core into one-state equalizer, and start computation.
3. Start the equalizer core unit in normal mode to find the soft-bit.
4. When the result of the burst is ready, inform the DSP to fetch the data.
5. When DSP requests data, read output data from RAM and send it further to DSP.
6. Be ready to store the next burst into RAM.

According to the tasks stated above, a state machine is implemeted in the OCP interface controller, of which the state transition shown in figure 3.6. Below is a description of the function of each state in the state diagram.

IDLE This state is the initial state when the system is powered up. In this state, the OCP interface controller waits for the Cmd_reg to be set by the

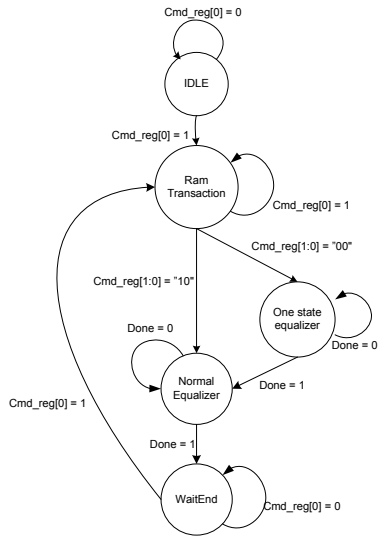

Figure 3.6: The state diagram of the OCP interface controller

DSP unit. When the DSP sets the Cmd_reg(0) to '1', the control moves to the RAM_TRANSACTION state and starts the equalizing process. During equalizing process, the system will not enter this IDLE state again.

RAM_TRANSACTION This state handles the Read/Write process by the DSP unit. In this state, the DSP unit has full access to the RAM, and the equalizer unit works in slave mode. The DSP first reads out the result of the previous burst, and then writes a new burst to the RAM. When the DSP unit finishes the read/write access to the RAM, it sets the Cmd_reg[0] to '0'. The controller enters the NORMAL_EQUALIZER state if the Cmd_reg[1] is '1', or the ONE_STATE_EQUALIZER state, if the Cmd_reg[1] is '0'.

NORMAL_EQUALIZER In this state, the equalizer unit works in master mode. The controller starts the equalizer core by asserting the *start* signal. The equalizer core has the full access to the data RAM in this state. When the core finishes processing one burst of data, it asserts the *done* signal, and the controller enters the WAIT_END state.

ONE_STATE_EQUALIZER This state is used for calculating the threshold value of the trellis structure. The controller in state functions the same as in the NORMAL_EQUALIZER state. It asserts the *start* signal and the *one_state* signal to start the equalizer in one-state equalizer mode, and waits for the *done* signal to be asserted by the equalizer core. When this state ends, the controller assumes the threshold value has been calculated. Then it will always enter the NORMAL_EQUALIZER state. This state should always be skipped in the RSSE equalizer by keeping the Cmd_reg[1] to be '1', since no threshold calculation is needed.

WAIT_END In this state, the controller asserts the *interrupt* signal to inform the DSP to fetch the soft-bit result. Then it waits for the DSP to request data. When DSP unit sets the Cmd_reg[0] to '1', the controller leaves this state and enters the RAM_TRANSACTION state.

3.4 RSSE Equalizer core

3.4.1 Structure overview

This equalizer core implements the RSSE algorithm (refered to as RSSE equalizer below). The structure design of the RSSE equalizer is shown in figure 3.7.

A brief description of the submodules and their function is as followed:

Channel symbol LUT This module implements the inner product of the channel impulse response with the Surmem elements. The Surmem elements, together with the input of the modulation type, select a value from the lookup table which stores the multiplication result of the channel impulse response with the complex representation of the symbols. Six values from the LUT are selected, and the sum of these six value is the output of this module. This module implements the summation part of equation 2.10.

Error Calculation This module takes the modulation type and the decision variables calculated as in equation 2.10 as input. Then it calculates the distance(squared error) of the decision variable to each of the reference point on the constellation diagram. The squared error to each point is the output of this

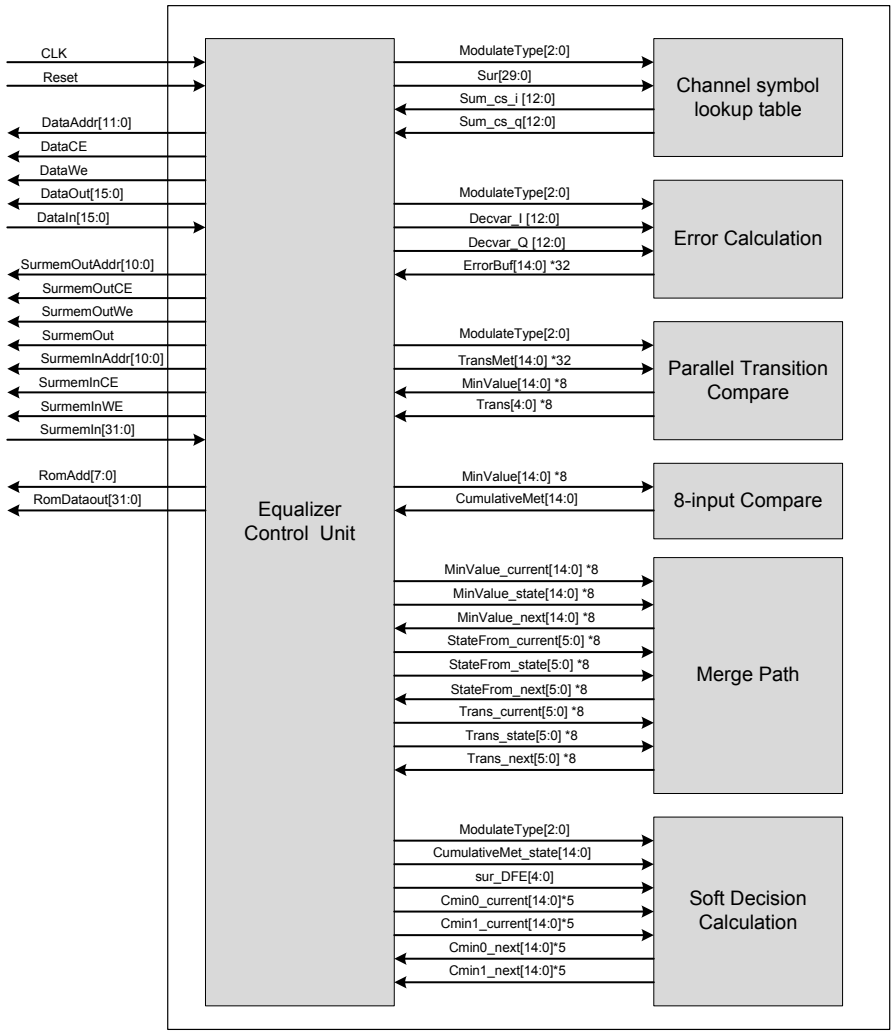

Figure 3.7: Overview of the equalizer core

module. The maximum number of the errors is 32 as in 32QAM modulation type. Therefore the output ErrorBuf is defined as an array of 32 elements.

Parallel Transition Compare This module compares the parallel transitions to one state and merge the path. Likewise, it takes the modulation type as an input, since the parallel transitions differ in each modulation type. The

other input, TransMet, stores the path metric value, which is the sum of the registered ErrorBuf value from the Error Calculation module and the state metric of the current state. The parallel transition are merged by choosing the minimum value of the path metric that have the same destination state, and stored in the output array: MinValue, and the transition associated to this minimum value is stored into another output array: Trans. Together they are the output of this module. The maximum number of MinValue is 8, so the output are arrays of 8 elements.

Merge Path This module compares and merges the path of different trellis states that needs to be merged. It compares the array which stores the current minimum metric MinValue_currrent, with the registered MinValue result (MinValue_state) from the Parallel Transition Compare, which is the metric of the current state. The eight values in the array MinValue_state represent the metric of the path going to eight destination states, and they should be compared with the corresponding metric in the array MinValue_current that has the same destination state. The minimum value after the comparison is stored into the array MinValue_next, and the registered MinValue_next is the new MinValue_current input of the next clock cycle. Likewise, the transitions associated to the minimum value are stored into the Trans_next array, and the source states associated to the minimum value are stored into the StateFrom_next array. When no more paths need to be merged for these destination states, the MinValue_next result will be stored as the state metric of the destination state in the next stage. The Trans_next and the StateFrom_next will be saved into the register files and used for updating the Surmem of the next stage.

8-input Compare This module takes the registered MinValue output from the Parallel Transition Compare module as the input, and finds the minimum value among these MinValue. This value is called CumulativeMet and will be used as the input for the Soft Decision Calculation module.

Soft Decision Calculation This module calculates the soft-bit result according to the modulation type, the CumulativeMet from 8-input compare module, the surviving symbol memory element, and the registered temporary soft-bit result. This module implements the equation 2.26, and calculates the soft-bit decision from the input signals. The output signals are the temporary soft-bit result Cmin0_next and Cmin1_next of the current state. The output will be register and used as the input for the next clock cycle. When all the states in the trellis structure have been calculated, the result will be saved as the soft-bit result for the current stage.

3.4.2 State diagram

A state machine is implemented in the equalizer's controller unit. The state machine handles the memory read and write, and sends the control signal to the data path according to the RSSE algorithm. The state diagram is shown in figure 3.8, and the operations the controller performs in each state are explained below.

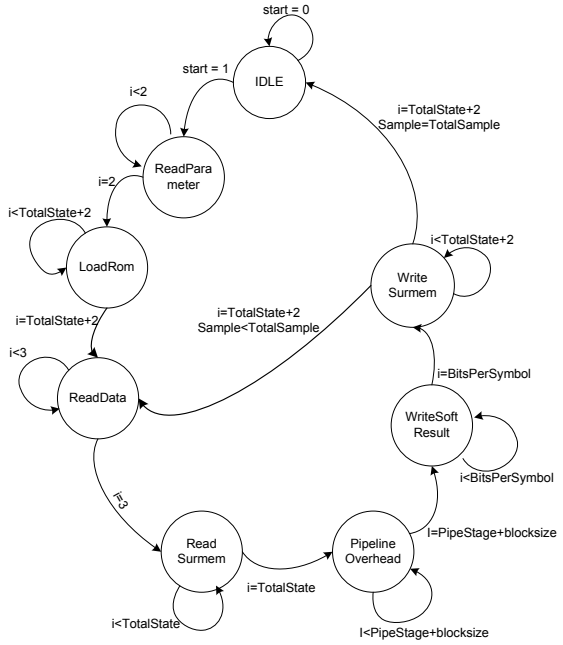

Figure 3.8: The state diagram of the RSSE equalizer

IDLE The controller unit is in the state when the system is powered up, or when it finishes processing a burst of data. The controller waits in this state until the top level controller asserts the *start* signal. And it will go to the state READ_PARAMETER.

READ_PARAMETER In this state, the controller reads the operation mode control byte stored in the data RAM and decodes the byte. The modulation type, total number of states used in the trellis structure, and total number of samples in the burst are stored into the registers. Since the data RAM is a synchronous RAM, it will drive the data on the data bus in the next clock cycle.

The input data from the RAM is stored in a register, so one more clock cycle is need for the controller to get the data. A counter i is used to record the number of the times the state is executed. After 3 clock cycles, the data is obtained and decoded, the controller moves on to the LOAD_ROM state. The controller only enters this state before it processes the first sample of each burst.

LOAD_ROM The controller load the initial Surmem elements from the ROM according to the Modulation type and total states it decoded in the READ _PARAMETER state. If N states in the trellis are required, the state will loop for N clock cycles. Each clock it reads out 6 elements for the same trellis state, and stores them into a certain address in the Surmem RAM. When all the initial Surmem elements have been loaded into the RAM, the controller enters the READ_DATA state. Same as the READ_PARAMETER state, this state is only executed before the burst samples are processed.

READ_DATA In this state, the controller reads the I and Q of the current processed sample from the data RAM, and stores them into the registers. This state starts the trellis calculation for each stage. Since two addresses in the RAM have been read, the state will be executed for four times to get the required data.

READ_SURMEM This is the state where most of the calculation is taking place. Depending on the total trellis state N, the state will be executed for N clock cycles. Each time it reads out six Surmem elements associated to one trellis state and inputs them into the data path. The data path performs path extension and merging for a new trellis state at each clock cycle. The trellis states have further been grouped into blocks. The states in the same block have the same sequence of destination state, therefore in-block merging is performed. When the controller finishes calculating one block, it sends out a control signal to store the path merging result into register files. After N clock cycles, the controller assumes that all the trellis state have been calculated, and it enters the PIPELINE_OVERHEAD state.

An overview of the number of block and block size in each modulation types are shown in the table 3.7.

PIPELINE_OVERHEAD The data path of the equalizer core is pipelined into 7 stages to increase the throughput of the critical path. Due to the pipeline, the data path will have a latency of 6 clock cycles. When N trellis states are calculated in the data path, a N+6 clock cycles are needed for pipe-cleaning.

Modulation type	BlockSize	Number of blocks		
		16 states	32 states	64 states
GMSK	2	8	16	-
QPSK	4	-	-	16
8PSK	4	4	-	16
16QAM	8	-	-	8
32QAM	8	-	-	8

Table 3.7: Block size and number of blocks for each modulation type

Furthermore, storing the path merging result of the last block will have a delay of "BlockSize" clock cycles, where the BlockSize is the number of states in a block(seen in table 3.7). Therefore this state will be executed for (BlockSize+6) clock cycles. Then it enters the WRITE_SOFT_RESULT state.

WRITE_SOFT_RESULT The controller writes the soft-bit result of the current sample back to the RAM. The number of the results is the same as the bits per symbol rate, e.g. the 32QAM burst will have 5 soft-bit results for each sample, 16QAM will have 4, etc. The state is therefore executed for BitsPerSymbol clock cycles. Then the controller enters the WRITE_SURMEM state.

WRITE_SURMEM In this state, the controller updates the Surmem elements according to the path merging result. For each clock cycle, the controller updates the Surmem of one state in the next stage. This is done in the numerical order of the trellis state of the next stage. For example, if the current stage = 4, the controller will first update the Surmem elements for state 0 in stage 5. Supposing the merging result for state 0 in stage 5 is the path from state 3 in stage 4, the controller will read the Surmem elements for state 3 from the Surmem RAM, shift it and concatenates the new transition value, and store the new Surmem into the RAM at the address assigned for the state 0 of the new stage. At the same clock cycle, the state metric of the new state is stored into the data RAM. This state is executed for total state + 2 cycles, where 2 is the delay of reading from the synchronous RAM. After that, the controller checks whether there is more uncalculated samples in the burst. If there is, the controller enters the READ_DATA state to start the calculation for a new stage. If all the samples have been calculated, the controller asserts the *Done* signal to inform the OCP interface controller that the processing is finished, and it empties all the registers and goes into IDLE state.

3.4.3 Timing analysis

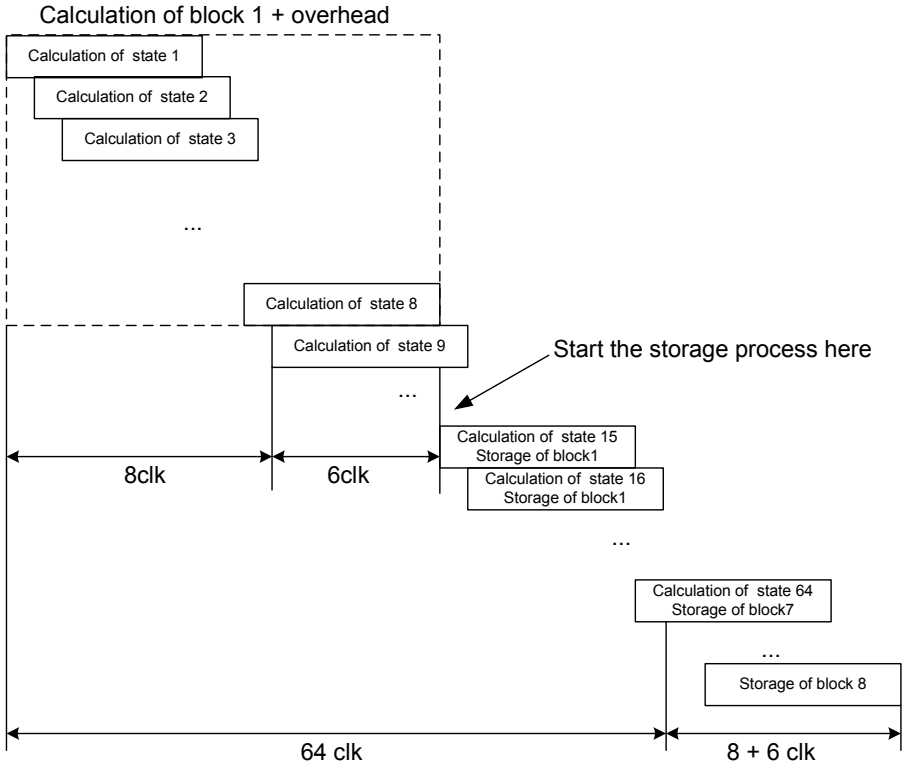

Figure 3.9: Pipelining the process of the RSSE controller

The figure 3.9 shows the pipeline organization at any trellis stage when processing a 32QAM burst. In the figure, each trellis state has a throughput of one clock cycle, and the storage of the first block starts at BlockSize + PipelineStage - 1 clock cycle. The total clock cycles needed for processing a stage (exclusive read/write from RAM) is $64 + 8 + 6 = 78$ cc. The computation time of the entire burst can also be estimated. Here the worst case, a high symbol rate 32QAM of 171 samples, is calculated. Adding the clock cycles for each state in the state diagram.

$$3 + (64 + 2) + [4 + 78 + 5 + (64 + 2)] * 171 = 26232cc$$

When using the clock with the maximum frequency = 200MHz, the clock period is 5ns, and the computation time is

$$26232 * (5 * 10^{-9}) = 131\mu s < \text{ required } 577/2\ \mu s$$

If comparing the computation time with the timing requirements, it can be seen that the design holds the timing requirements. In fact the design only occupies half of the maximum allowed time, which makes it possible to optimize the area by decreasing the use of the parallelism. However, this approach is not carried out in this project.

3.4.4 Module design

The data path of the RSSE equalizer is shown in figure 3.10. The data path is 7-stage pipelined, as to increase the throughput of the critical path.

The data path is designed according to the algorithm described in section 2.6.3. The pipeline stage 0 is store the input data and Surmem elements into registers. The pipeline stage 1 implements the summation part in equation 2.10. The pipeline stage 2 implements the rest part of equation 2.10. The pipeline stage 3 implements equations 2.12 to 2.15. The pipeline stage 4 merges the parallel transition, and together with stage 5, they implement equations 2.18 to 2.21. Stage 6 implements equation 2.26.

It can be seen that the clock-gating is applied on the equalizer by propagating an enable signal along the data path. The pipeline register stalls when the enable is '1'. In this way, the unwanted switching activities will not be performed by the data path.

The modules shown in the data path are implemented as combinatorial logics. All the input and output signals of the modules are registered in the controller's pipeline registers. This approach makes it easy to modify the RSSE equalizer into an equalizer using another algorithm, simply by replacing modules in the data path.

Below is a description of the hardware implementation of each module in the data path.

3.4.4.1 Channel symbol LUT

This module uses the Surmem memory elements as well as the modulation type as the index to search for the required channel symbols, as shown in figure 3.11.

The look-up table is implemented as a pre-calculated constant array. Each

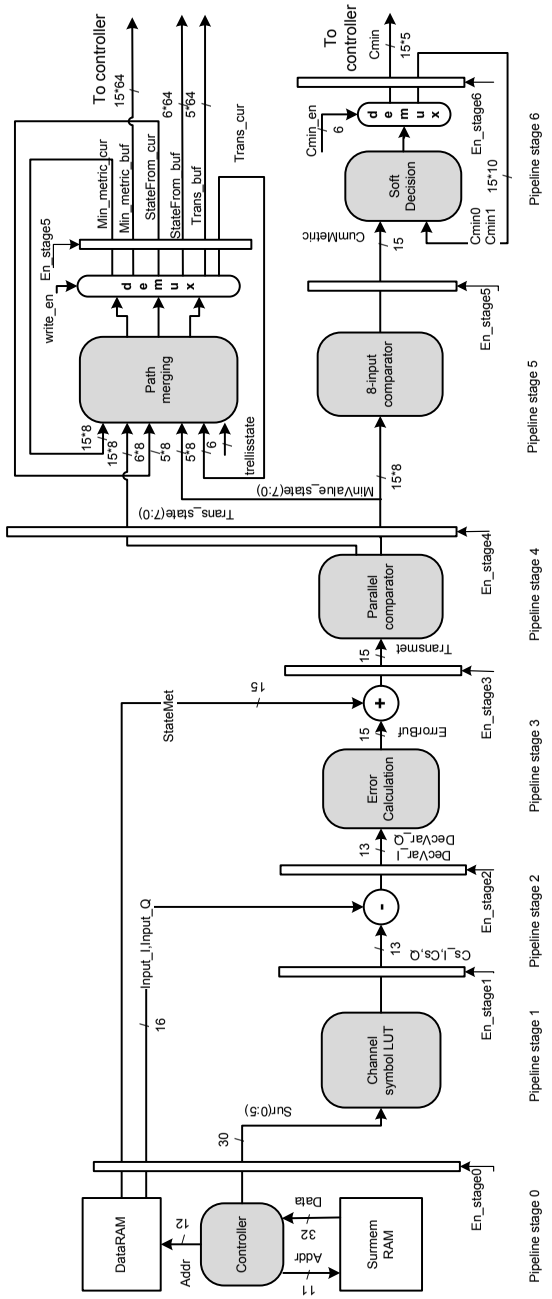

Figure 3.10: The data path of the RSSE equalizer

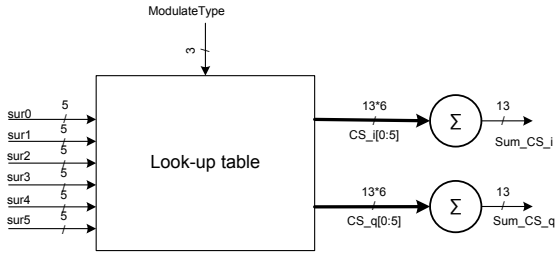

Figure 3.11: Channel symbol LUT module

address contains 6 integer numbers of 11 bits each, representing a range of [-1024,1023]. The pre-calculation is to multiply the channel impulse response with the complex representation of the constellation point. The result is an array of six complex numbers, called six channel symbols. The real part of the six channel symbols are saved at one address of the look-up table, and imaginary part are saved at the next address. Figure 3.12 illustrates the contents of the first few addresses in the LUT. From the figure, the real part of the channel symbols of symbol 0 are saved at address 0, while the imag part are saved at address 1. The real part of the channel symbols of symbol 1 are saved at address 2, while the imag part are saved at address 3. The GMSK channel symbol look-up table is followed by the QPSK channel symbol table etc., in the memory space.

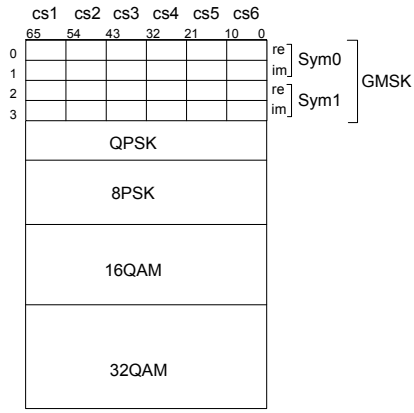

Figure 3.12: The data organization in the look-up table

When the controller sends an array of Surmem elements of one trellis state, the elements are used to select in the look-up table. E.g. when the modulation type

is GMSK, and the Surmem is [0,0,1,0,1,1], the channel symbols are selected as [CS1 of symbol 0, CS2 of symbol 0, CS3 of symbol 1, CS4 of symbol 0, CS5 of symbol 1, CS6 of symbol 1], where CSn is the n'th complex channel symbol.

This module can be implemented alternatively by doing run-time multiplication instead of using a pre-calculated look-up table. This approach requires six complex multipliers, which can be implemented by four multipliers of 8*4 bits each. A total of twenty four 8*4 multipliers should be used to make the module function the same way as the LUT approach. Since the LUT is of the size 11*6*124b = 8184 hardwired bits, it can be estimated that the LUT approach is better both in area and in power than the complex multiplier approach, thus it is chosen to be implemented.

3.4.4.2 error calculation

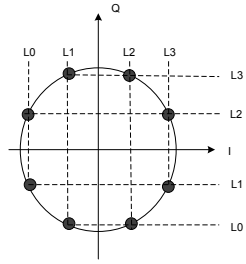

Figure 3.13: The level of 8PSK

This module calculates the squared error of the decision variable to the reference point on the constellation diagram. The constellation diagram is selected first by the modulation type input, and then the error towards each constellation point is calculated. Normally the squared error is calculated by

$$squared\_error_0 = (decvar_i - ref0_i)^2 + (decvar_q - ref0_q)^2$$

This method will perform 2*N times minus-and-square calculation, while N is the number of constellation points. However, since the constellation points are always arranged in a rectangular lattice, it is unnecessary to calculate each point by its I and Q coordinates. For example, for a 8PSK constellation shown in figure 3.13, the points are placed in four levels along the I-axis, and four levels along the Q-axis. Therefore, it is only necessary to calculate the partial error of decision variable I with the 4 levels, and the partial error of decision variable Q with the 4 levels. The 8 partial error results are selected for each constellation points, and added to form 8 squared error value. In this way, only 8 calculation are done instead of 16. Below in figure 3.14 shows the data path for calculate

the error of decision variable I with level L0, and decision variable Q with level L0. In the Error Calculation module, 6 of these units shown in figure 3.14 are used, and they form a maximal of 32 squared error outputs.

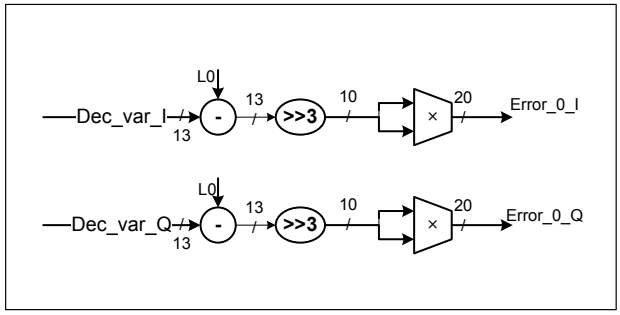

Figure 3.14: The error calculation unit, one set of error

The diagram of the complete module is shown in Appendix B figure B.2.

3.4.4.3 Parallel Transition Compare

The output from the Error Calculation module is organized in a way that the error to the points of the same subset of set-partitioning is placed together. For example, the constellation points of the 8PSK is set-partitioned as [(0,4),(1,5),(2,6),(3,7)], and the output array ErrorBuf from the Error Calculation is $ErrorBuf(0:7) = [Err_0, Err_4, Err_1, Err_5, Err_2, Err_6, Err_3, Err_7]$, and the rest the of the ErrorBuf is all 1's (maximal value). Then the comparison is carried out within the subset, which in this example is Err_0 with Err_4,Err_1 with Err_5, etc. The symbol is passed as the parameter of the comparison. The smaller one is kept as the output. The comparison unit for the subset (0,4) is shown in figure 3.15.

In 32QAM, the points are set-partitioned into 8 subsets with 4 points each. Therefore the 4 points are compared two and two by using two of the comparison units shown in figure 3.15, and the results are compared again using one comparison unit. A total of 24 comparison units are required for the 32QAM, and the result is 8 path merging survivors, each to a different destination state.

While the burst of modulation types with smaller alphabet, like 16QAM and 8PSK, is calculated, not all the comparison units are required. The input to the unused logics remain unchanged during the processing to avoid switching activity by applying operand-isolation.

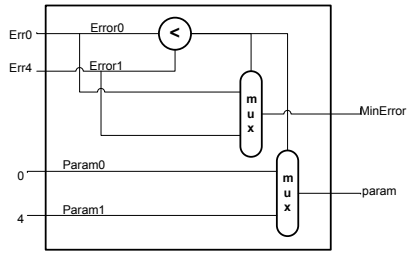

Figure 3.15: The comparison unit with one parameter

For QPSK and GMSK, there is no set-partitioning performed on the symbol set, and thereby no parallel transition. The module should be passed through.

The diagram of this module is shown in Appendix B figure B.3.

3.4.4.4 Merge Path

The Merge Path module compares and merges the path of different trellis states. The input to the module is two arrays, MinValue_state(0:7) and Trans_state(0:7), containing the parallel transition merge result and the current trellis state number, respectively. They are compared with the current minimum metric array, MinValue_current(0:7). The comparison is done in a way such that MinValue_state(0) is compared with MinValue_current(0), MinValue_state(1) is compared with MinValue_current(1),etc. The comparison for one path metric is shown in figure 3.16. Supposing the MinValue_state(0) is smaller, it will be stored in MinValue_next(0). Its transition Trans_state(0) will be stored in Trans_next(0), and the current trellis state number will be stored in StateFrom_next(0). Otherwise the MinValue_current and its parameters are stored. One clock cycle later, the "_next" signals is stored into the registers "_current", and used to compare with the metric of the next trellis state.

The controller sends a signal *write_en* to indicate if the merge process is finished. And if the signal is asserted, the MinValue_next, Trans_next,and StateFrom_next are written in the register files MinValue_buf, Trans_buf, and StateFrom_buf respectively. The register "_current" are cleared for next block of merging process. This will be introduced in details in the section 3.4.4.7 book-keeping mechanism.

A total of 8 units shown in figure 3.16 are used in the module. Only the logics inside the box in the figure are the logics in the module. The logics outside the

box belong to the controller unit.

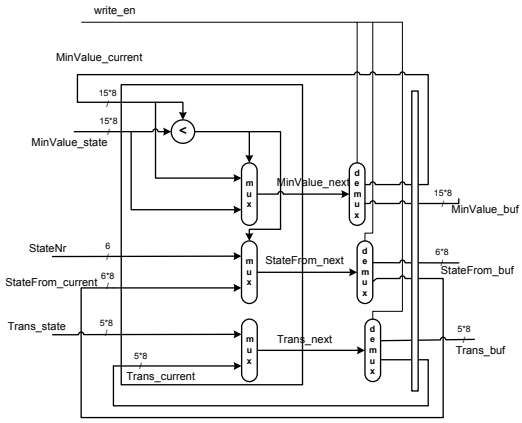

Figure 3.16: The Merge path unit for one pair of metric

3.4.4.5 8-input Compare

This modules compares the 8 input path metrics and finds the minimal metric among them. The implementation of this 8-input compare is by employing an

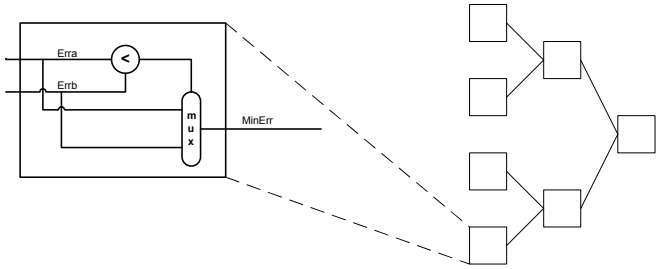

Figure 3.17: Left: The comparison unit with no parameter. Right: Binary tree structure of the 8-input Compare module,each block is a comparison unit

array of four comparator units shown in the left side of figure 3.17. The array is organized in a binary tree structure as shown in the right side of figure 3.17. Similar to the Parallel Transition Compare unit, the input to the unused logics is always set to 1's.

3.4.4.6 Soft Decision Calculation

This modules calculates the soft-bit decision. The input is taken from the 8-input compare module's output. As described in the previous chapters, The soft decision is made for the transmitted symbol from the N^{th}-previous stage, where N is decided by the modulation types and the total states. The controller sends this transmitted symbol SurDFE to the module, and the bit format of the symbol is found by using a bit format LUT. If symbol contains 5 bits, 5 units shown in figure 3.18 are used in the module. The BitN indicates which bit in the symbol is calculated. Each unit compares the input minimum metric with Cmin0_current, if the bit is 0, or with Cmin1_current, if the bit is 1. The minimum value of the comparison are stored, and used as the compare value Cmin0_current or Cmin1_current for the next state.

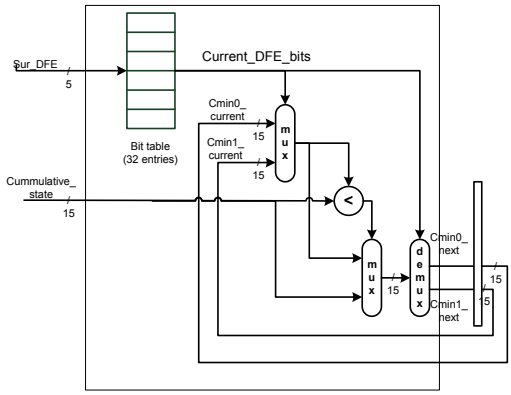

Figure 3.18: The Soft Decision Calculation unit

3.4.4.7 Book-keeping Mechanism

The trellis structure used in the project has the characteristics that the states in the same block have the same destination states. The example shown in figure 3.19 is a 16-state 8PSK trellis structure, separated into 4 blocks. From the figure, it can be seen that all the states in the first block have the destination states of the first state in four blocks.

This trellis structure requires an advanced book-keeping mechanism, since the path metric, source state, etc, of the destination state are not calculated in the numerical order. In the example in figure 3.19, the sequence of the calculation

of the destination states are: (0,4,8,12), (1,5,9,13),(2,6,10,14),(3,7,11,15), with each set (shown in the bracket) computed at the end of the block.

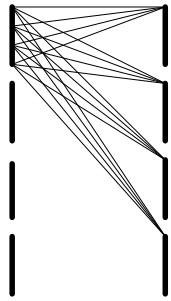

Figure 3.19: The trellis structure of 8PSK 16 state modulation type in blocks

Basically two register files are used for the book-keeping mechanism: one used for the temporary path merging result inside a block (Block Registers), and one to store the result of the stage(Stage Register Files). The Block Registers have 8 entries, and are cleared when the calculation of a block finishes. The Stage Register Files have 64 entries, and are cleared at the end of each stage.

The book-keeping mechanism is closely related with the Merge Path module. The output from the Merge Path module are stored into the registers: Min_metric_current, Trans_current, and StateFrom_current, as shown in figure 3.16. These are the actual content of the Block Registers' entry. Each of them contains a maximum of 8 elements going to 8 different destination states. When the computation of a block is finished, the Block Registers are written into the State Register Files directly. A pre-calculated destination state value (StateTo) is added to each entry of the State Register Files. As shown in the example in figure 3.20, for the first block, the 4 entries in the Block Registers are written into the first four entries of the Stage Register Files. For the next block, the Block Registers will be written into the next 4 entries of the Stage Register Files.

When the calculation for a stage is finished, the data in the Stage Register Files are written into the RAM. Since the data in RAM are placed according to their state number, the StateTo signals are used to calculate the destination state.

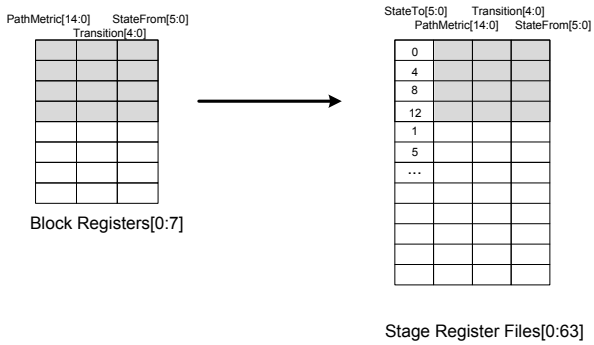

Figure 3.20: The Block Registers and the Stage Register Files for the 8PSK 16 state example, block 1

3.5 SD Equalizer core

This section describes the implementation of the equalizer core of the SD algorithm. When implementing the SD algorithm, two approaches are used. They differ in the way that the first approach (referred to as SD_I equalizer) checks the living status of all the states, while the second approach (referred to as SD_II equalizer) maintains a smaller list of surviving path, and only performs sequence estimation on the surviving states. Since the first approach need one clock cycle to check every state in the trellis structure, it uses considerably long time for burst with a large trellis structure, such as 1024 state 32QAM. While the second approach requires a more complicated controller unit to find the survived states and a more flexible book-keeping mechanism, which are expensive in area. Both approaches are introduced in this section. Since most modules in the data path are the same as the RSSE equalizer, only the modified parts are introduced.

3.5.1 SD_I equalizer design

3.5.1.1 Structure design

The structure design of the SD equalizer is shown in figure 3.21.

Compared with the RSSE equalizer, it can be seen that the module **Parallel transition compare** and the **8-input Compare** are replaced by a new mod-

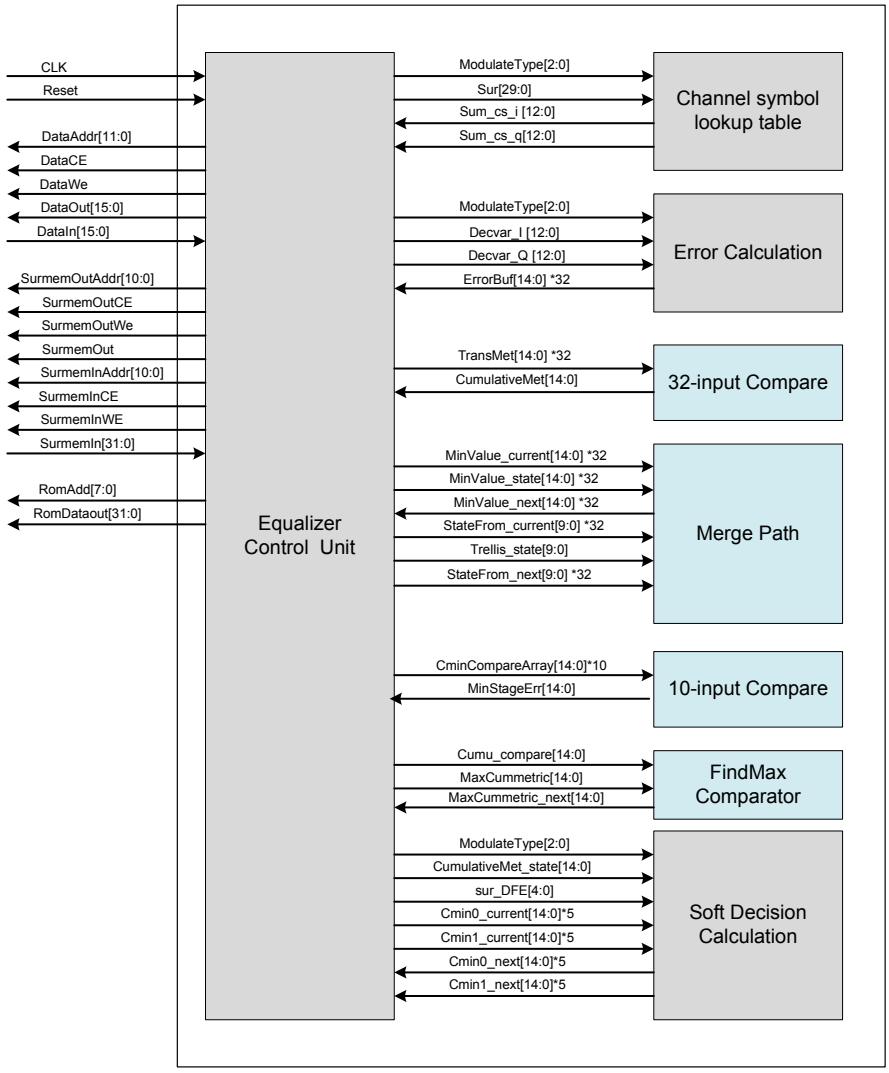

Figure 3.21: Overview of the equalizer core of SD_I algorithm

ule **32-input Compare**. This is because the SD algorithm uses a full trellis instead of a reduced one with set-partitioning, no parallel transition exists in the trellis structure. Accordingly, the module **Merge path** can now find out the transition according to their locations in the array, therefore the Trans input is not necessary, and input of this module is reduced to 6. The other modules that marked in blue are the new modules. They include:

- **32-input Compare**: This module is to replace the module **Parallel transition compare** and **8-input compare**. It simply combines the 2-stage comparators in the module **Parallel transition compare** and the 3 stage comparators in the module **8-input compare** together, and makes a 5-stage binary tree structure with comparators with 1 parameter, as shown in figure 3.15.

- **10-input Compare**: This module is very similar to the module 32-input Compare. It compares the ten elements in the input array in a paralleled way. It has a tree structure similar to the **8-input compare**, of which each stage contains 5-2-1-1 comparators as shown in 3.17. This module is used to compare among the soft-bit decisions to find the cumulative error of the stage.

- **Findmax Compare**: This is a single comparator similar as shown in 3.17 with a opposite bigger-than operator. The module is used by the one-state trellis to find out the maximal error of the stages.

The data path of the SD_I equalizer is shown in figure 3.22. The difference with the data path of RSSE equalizer is that there are 6 pipeline stages instead of 7, and new modules introduced above are plugged into the data path.

Although the structure and data path of the SD_I equalizer are similar to the RSSE equalizer, the controller design is very different. One major difference is that an one-state equalizer is designed to find the maximum stage error of the burst. The section below introduces the design of the one-state equalizer.

3.5.1.2 One-state equalizer

The SD equalizer should be able to function as a one-state equalizer. Since one-state equalizer can be taken as a special form of the normal equalizer, it should be possible to reuse the data path with some different control signals. If the one-state equalizer is configured, the state machine goes into a different sequence than the normal equalizer. The state diagram is shown in figure 3.23.

The IDLE state, the READ_PARAMETER state, and the READ_DATA state are the same as in the RSSE state machine. The new states are:

ONE_STATE_EQUALIZER This state controls the most part of data processing. It captures the Surmem elements sent by the RAM, and feeds them

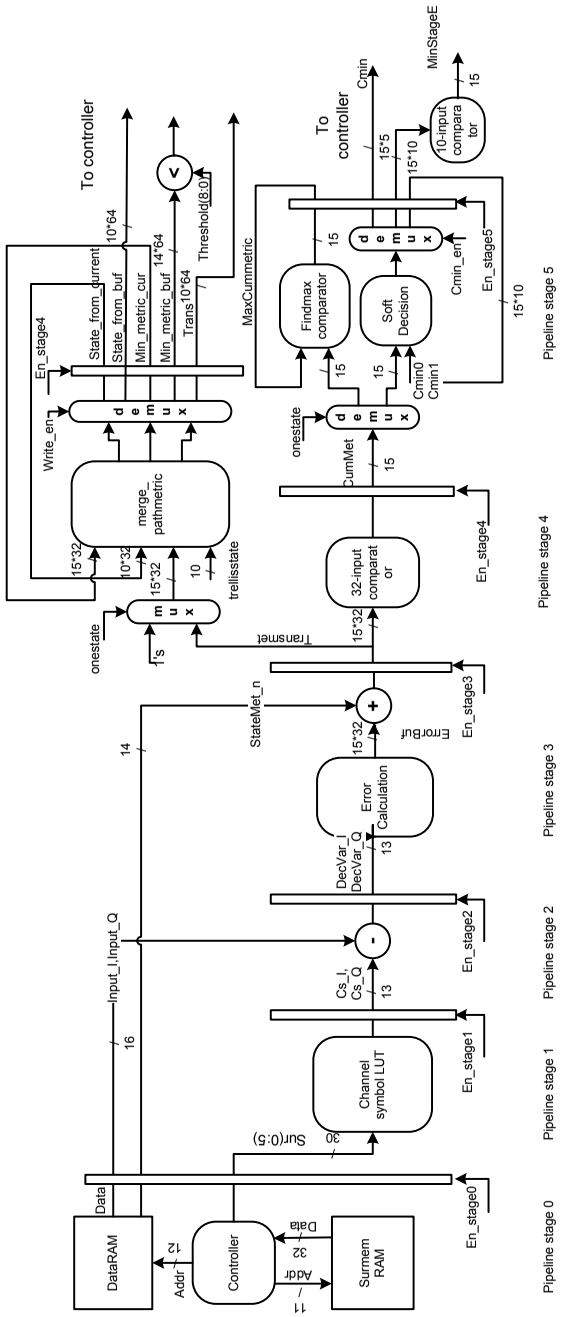

Figure 3.22: The data path of the SD equalizer

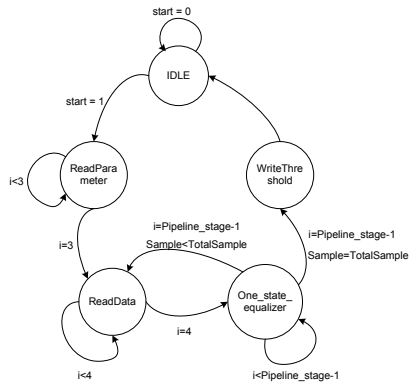

Figure 3.23: State diagram of the one-state equalizer

into the data path. Since for each stage, only one trellis state needs to be calculated, and the calculation for next stage cannot be started until the current stage is finished, the pipeline structure cannot be used to optimize the throughput. Each stage will take 6 clock cycles to finish. Then the controller goes to the READ_DATA state, if there is more data in the burst. Otherwise it goes to the WRITE_THRESHOLD state.

WRITE_THRESHOLD In this state, the controller takes the MaxCummetric from the module **Findmax Comparator**, scaled it by 2 to make a "pessimistic" threshold and save the value at the address 0 in the data RAM. The controller also asserts the *done* signal to indicate the OCP interface controller that the calculation is finished.

3.5.1.3 SD_I controller unit

One of the most significant difference in the SD algorithm is that not all the trellis states in the trellis structure need to be calculated. Therefore the additional tasks in the SD_I controller implementation are:

- A method to detect the surviving state, and perform calculation only for the surviving state.
- A new book-keeping method to store the more flexible structure of path metric and other parameters of the surviving states.

Surviving state detection The first problem can be solved by using a bit array *surviving_state_list[0:N-1]*, of which each bit represents the surviving status of a trellis state, and N is the total number of states in the trellis structure. Two lists of this are needed in each trellis stage. One for the states in the current stage, and the other for the states in the next stage.

There are various ways to decode the *surviving_state_list*. In SD_I equalizer, the controller scans through the list at the rate of one state per clock cycle, and checks if the state survives by reading the corresponding bit in the *surviving_state_list*. At the same time, the controller reads the input I, Q data and the Surmem elements from the RAM. But the pipeline registers are disabled by the controller, as not to invoke unnecessary switching activity. If the state is a surviving state, the controller enables the pipeline registers to perform the calculation. In this way, the equalizer only performs calculation of the surviving states.

The Book-keeping mechanism Similar to the book-keeping mechanism used in RSSE equalizer, two register files are used for the book-keeping mechanism: one used for the temporary path merging result inside a block (Block Registers), one to store the path-pruning result of the stage(Stage Register Files).

As in the RSSE controller unit, the trellis structure is also divided into blocks. Since the full trellis structure is used in stead of the RSSE trellis structure, the number of states in a block (BlockSize) is equal to the number of the transitions from each state, which is also the number of the symbol set. The BlockSize and the total block number in a trellis stage is shown in table 3.8.

Modulation type	BlockSize	Number of blocks
GMSK 16 state	2	8
GMSK 32 state	2	16
QPSK 64 state	4	16
8PSK 64 state	8	8
16QAM 256 state	16	16
32QAM 1024 state	32	32

Table 3.8: Block size and number of blocks for each modulation type in SD equalizer

The path merging is performed at each clock cycle on the currently calculated path extended from the survived states. And at the end of each block, each entry of the Block Registers contains the minimum path metric of a path going to a certain destination state, together with the source state of the minimum path.

The index of the entry indicates the transition. Some states don't have any surviving path due to the pruned source states, and they will have the default value of all '1's in the Block Registers. The path metric in each entry of the Block Register will then be compared with the threshold minus the cumulative error from previous stage. And only the entries with the path metric lower than the compared parameter survive, otherwise the state is pruned. The path metric of a pruned state will not be recorded, but instead an all 1's value is written to the path metric of this entry of the Block Register, and all 0's value are written to the source state (StateFrom) of this entry of the Block Register, as shown in figure 3.24.

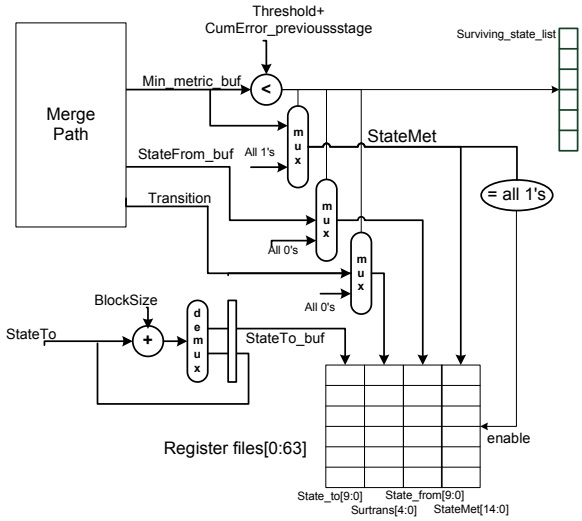

Figure 3.24: Book-keeping mechanism of the SD_I equalizer

The controller then examines the Block Register that contains the path metric. And at each clock cycle, it scans the Block Registers with path metric, and writes a path metric which is not the all 1's to the Stage Register Files, together with its source state, transition value and the destination states, as shown in figure 3.24. The surviving_state_list of the next stage is also updated. Since the Block Registers have the number of entries of BlockSize, it will take "BlockSize" number of clock cycles to finish. This writing process is carried out in parallel with the reading and calculating new state process, which takes BlockSize + pipelinestage -1 clock cycles to finish. The size of the Stage Register Files is limited: it only has 64 entries, which means that at each stage, only a maximal of 64 states are allowed to survive. This value is both based on the simulation of matlab reference model, and the fact that if the SD algorithm is expected to have better performance than the RSSE algorithm, the number of the total state

should not exceed the number used in the RSSE algorithm. A fixed number of maximum surviving states is of course not an optimal approach in the term of BER, but is the simplest way to implement.

When all the calculation blocks have been calculated and the surviving states have been written to the Stage Register Files, the controller output the data in the register files to the RAM. The data will be written to the address according to their trellis state number, e.g. the Surmem elements of state 30 will be written to the address: offset_address + 30, where the offset_address is the starting address where the Surmem elements are stored in the RAM. This memory allocation scheme has the advantage of simple indexing and efficiency in accessing. The disadvantage is that the RAM has low utilization rate, especially in the high SNR situation.

State Transition In order to handle more complicated control signals, the controller's state machine is extended. The state diagram is shown in figure 3.25. It can be seen that the states from RSSE controller: READ_SURMEM and PIPELINE_OVERHEAD have been replaced with four new states: COMPUTE_FIRST_SAMPLE, COMPUTE_BLOCK, PIPELINE_OVERHEAD and CHECK_LAST_BLOCK. The other states in the state diagram have the same function as in the RSSE equalizer.

When the controller is started and the first input sample is read, it goes to the COMPUTE_FIRST_SAMPLE state. As indicated by its name, the COMPUTE_FIRST_SAMPLE is to calculate the initial state in the first stage. The situation is treated separately since only one state is calculated, and its Surmem elements is stored as a constant array in the equalizer unit. After one clock cycle, the state machine enters the PIPELINE_OVERHEAD state, which finishes the pipeline calculation in the data path, and then it goes into the CHECK_LAST_BLOCK state. As mentioned above, the process of writing the surviving states into the register files is carried out together with the process of calculation the next block. But when the last block is calculated, no calculation of the new block is performed. The CHECK_LAST_BLOCK state is designed to handle the write-register-file process alone. Since in the first stage, only one block containing the initial state is calculated, it is also the last block of this stage. After executing this state for a number of "BlockSize" times, the controller goes to the WRITE_SOFT_RESULT state and WRITE_SURMEM state. These two states perform the same as in the RSSE controller.

Then the controller reads the second input sample, and it goes to the COMPUTE_BLOCK state. This is the major part of the trellis calculation. It is executed continuously for BlockSize times. The state performs both calcula-

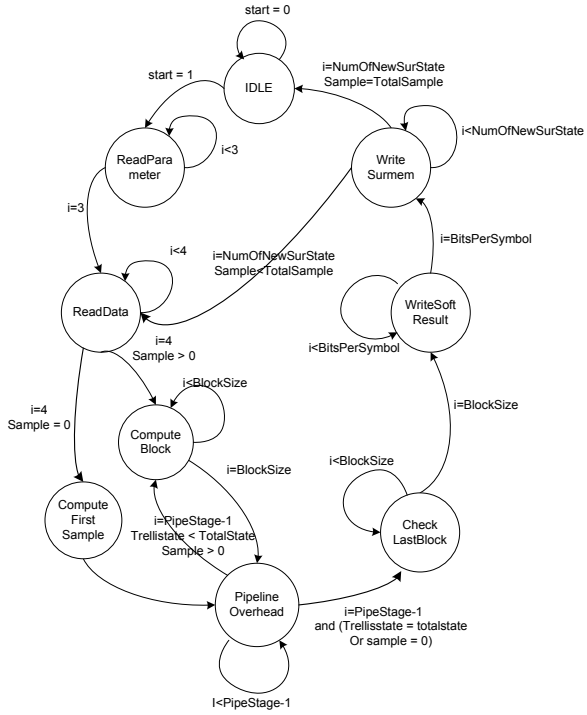

Figure 3.25: State diagram of the SD_I equalizer

tion for the new block, and storage of surviving states of the last block into Stage Register Files. After being executed for BlockSize times, it enters the PIPELINE_OVERHEAD state. After the PIPELINE_OVERHEAD state is finished, the controller will go back to the COMPUTE_BLOCK state to calculate the next block, if there are more blocks in the trellis structure, or go to the CHECK_LAST_BLOCK state.

3.5.1.4 Timing analysis

Like the RSSE controller, the computation time to process a burst by SD_I controller is examined. The pipeline structure of the controller is not as efficient as in the RSSE controller. Each state in RSSE takes one clock cycle, and each sample is calculated in TotalState + PipelineStage - 1 + BlockSize clock cycles. The SD_I controller will not start calculation for a new block until the previous block is totally finished (including the pipeline overhead part). Therefore, it takes BlockSize + PipelineStage -1 clock cycles to calculate a block, (BlockSize

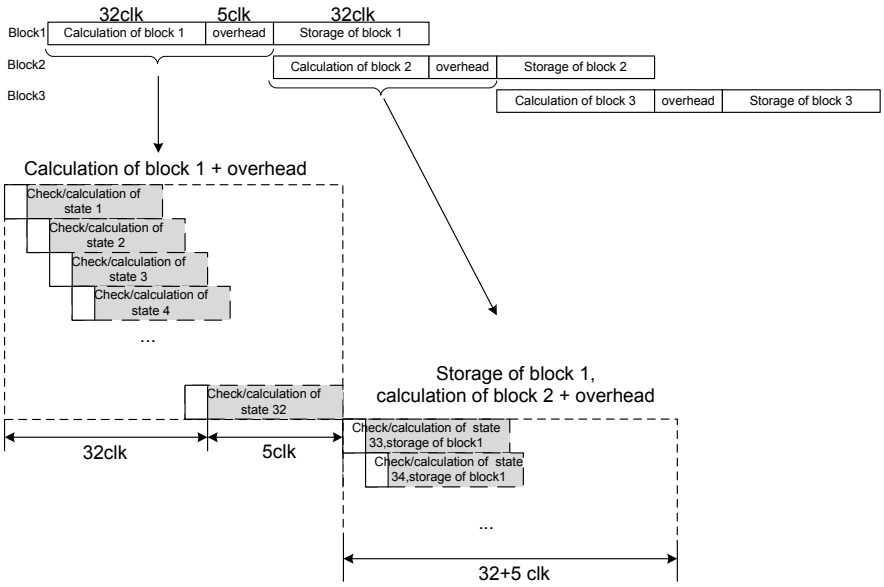

Figure 3.26: Pipelining the process of the SD_I controller

+ PipelineStage -1)*BlockNum + BlockSize clock cycles to finish a stage, as shown in figure 3.26. The computation time of a 32QAM normal symbol rate burst will be

$$3 + [4 + (32 + 5) * 32 + 32 + 5 + (64 + 2)] * 143 = 184616cc$$

When using the clock with the maximum frequency = 200MHz, the clock period is 5ns, and the computation time is

$$184616 * (5 * 10^{-9}) = 923\mu s > \text{ required } 577/2\ \mu s$$

The 32QAM burst will have too long computation time which cannot hold the timing requirements. The computation time for the high rate 16QAM burst is 365μs, which holds the timing requirements. The other smaller alphabet modulation types also hold the timing requirements.

This timing analysis shows that the SD_I equalizer cannot be used for equalizing the 32QAM bursts within the required time. Only the modulation types with

less complexity can be used. However, this approach has the advantage of power and area efficiency, therefore it is still implemented in the project. Its performance has also been evaluated.

3.5.2 SD_II equalizer design

3.5.2.1 SD_II Controller unit

This section introduces an alternative approach to implement the SD_equalizer, which is referred to as the SD_II equalizer. The SD_II equalizer differs with the SD_I equalizer in its controller unit. The rest part are the same. Therefore only the controller design is described here.

The timing analysis shows that the SD_I approach failed to hold the timing requirement of processing a burst in 585 ms. The problem of the SD_I approach is that the controller checks one entry of the *surviving_state_list* at each clock edge, and decides if this state need to be calculated. Also on the storage side, it takes 1 clock cycle to check if the new state survives. Since in the SD algorithm, the surviving rate is normally under 25%, which means the controller spends 75% of the time checking pruned states. This makes the SD_I approach very inefficient. If the controller can skip all the pruned state and jump directly to the next surviving state at each clock edge, the computation time to process a burst will be decreased significantly. It requires the controller to be able to:

- implement the efficient search mechanism, so that the time spent on each stage is approximately equal to the number of surviving states.
- ensure the next block is not started until the previous block is finished with calculation, since the Block Registers are shared. This is important because the time spent in each block are different, depending on the number of the surviving states in the block.

The first task is actually to find the index of the first non-zero element in the array in a efficient way. For example, if the *surviving_state_list*[7:0] of an 8PSK sample is [1,0,0,1,1,0,1,0], the SD_I controller decodes the array in a way that: *"state 0 pruned", "state 1 survives", "state 2 pruned", "state 3 survives", "state 4 survives", "state 5 pruned", "state 6 pruned", "state 7 survives"*, each of which takes one clock cycle. The aim of the SD_II controller is to "leave the pruned states out" and decodes the array such that: *"state 1 survives", "state 3 survives", "state 4 survives", "state 7 survives"* in four consecutive clock cycles.

The solution is to use a shift (logic right shift) algorithm. An example is used here to explain this shift algorithm, as shown in Table 3.9.

clock	before right shift[7:0]								after right shift[7:0]								shift right	sum - 1
1	1	0	0	1	1	0	1	0	0	0	1	0	0	1	1	0	2	1
2	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	1	2	3
3	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1	4
4	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	3	7

Table 3.9: An example of the shifter algorithm

In the example, the shifter searches for the first bit from the right that is '1', which is bit 1 of sequence "10011010" in clock 1. Then it shifts this bit out by performing a logic right-shift of 2 bits: the bit 1 and bit 0, which will leave a sequence of "001000110". The shift value of 2 is stored. At each clock cycle, the shift value should be accumulated with the value from the previous calculation, and minus 1 to get the bit location. Here the shift value 2 plus the previous accumulative value 0 and minus 1 will get the location of 1. In clock 2, it searches again and finds the bit 1 is '1'. So it shifts two bit out again, obtains a new sequence of "00010001" and a new shift value of 2. And the location for the non-zero bit is 2 + 2 - 1 = 3. Same actions are performed again in clock 3 and 4. The result of the non-zero bits' location are [1,3,4,7] respectively, which conform to the expected result.

Such a shift algorithm is implemented in hardware by using an array of multiplexer, as shown in figure 3.27.

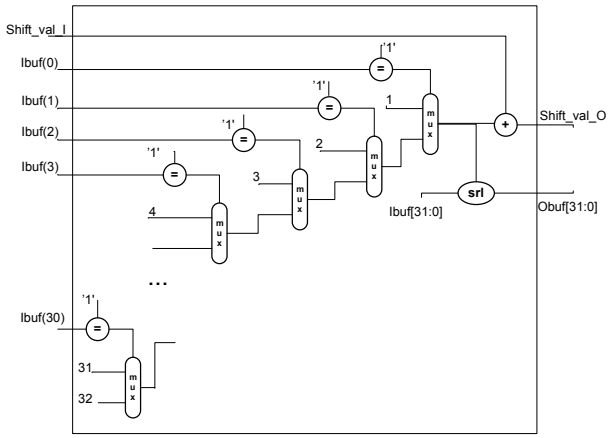

Figure 3.27: The shifter that finds the first non-zero element

In the figure, a total of 31 multiplexer are used. Ibuf[31:0] is the input sequence that needs to be decoded. Obuf[31:0] is the shifted output sequence. Shift_val_I is the accumulative shift value. Shift_val_O is the shift value of this iteration. When subtracted by 1, the Shift_val_O is the bit location of the first non-zero bit. The two output will be registered and become the input in the next clock cycle, until the sequence becomes all zeros.

The state diagram of the controller is shown in figure 3.28.

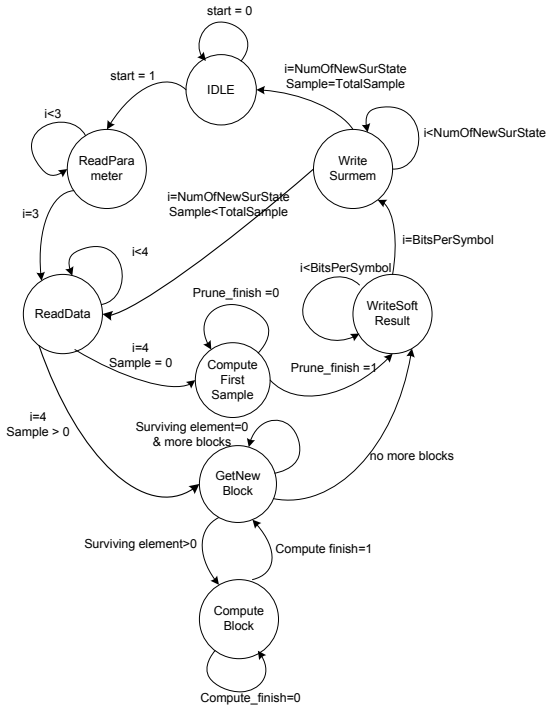

Figure 3.28: The state diagram of SD_II controller

The computation in SD_II equalizer is still organized in blocks. Like the SD_I controller, the main tasks of the controller are still: computing the surviving source states, and pruning the destination states. Two new control signals are used to record the status of these two operation: *Compute_finish* indicates the current block is calculated by the pipeline, and *Prune_finish* indicates that the survived destination states in the block and theirs parameters have been written into the Stage Register Files.

It can be seen that the state COMPUTE_FIRST_SAMPLE is still used for the

first stage of the trellis search. Normally the computing process and the pruning process are carried out in parallel. But for the first stage, only one block is computed. The block must be computed first, then be pruned. When the Prune_finish signal is asserted, the state machine goes to the WRITE_SOFT _RESULT state, which, same as in the other equalizer controllers, writes the soft-bit result to the RAM.

From the second stage, after reading the input sample from the RAM, the state machine goes to a GET_NEW_BLOCK state. This state checks if there is any surviving source state in the block by comparing the surviving list of the source states of a block with a value of all 0's. If all the states in the block are pruned, the state machine will stay in the current state, and check the next block. Otherwise, the state machine goes to the COMPUTE_BLOCK state, which handles the computing and pruning processes. Two special situations may occur in this state:

- No more new block is left. The controller will only perform pruning operation in the next COMPUTE_BLOCK state.
- After the controller finishes pruning operation mentioned above, the controller still goes back to the GET_NEW_BLOCK state. In this case the state machine will go to the WRITE_SOFT_RESULT state in the next clock cycle.

In the COMPUTE_BLOCK state, the controller performs both the computation of the new path metric of the current block, and the pruning of the destination states of the last block. It decodes both the surviving list of the source states of the current block and the surviving list of the next block, by using the shifter shown in figure 3.27. Some more control signals are used for the decoding, which include:

- shift_en: This signal is the enable signal of the shifter. This signal is used to prevent unwanted shift operations.
- index_valid: This signal is output by the shifter to indicate the non-zero index found by the shifter is valid.
- mem_valid: This signal is used to indicate the data from the RAM is valid. Since there is a 2 clock-cycle delay between the address and data, it is easier to use such a signal to capture the valid data, instead of counting the clock cycles between the address and data.
- mem_valid_advanced1: This signal is used to indicate the data from the RAM in the next clock cycle is valid.

An example showing the relations among the control signals and RAM signals are shown in figure 3.29. This example shows only the decoding of the surviving list of the source states. The controller finds the surviving state: 1 and 4 of the block in two consecutive clock cycles. Then it computes the address of the state 1 of the block, and state 4 of the block by adding the offset address of the current block, and requests the data of these two addresses from the Surmem RAM.

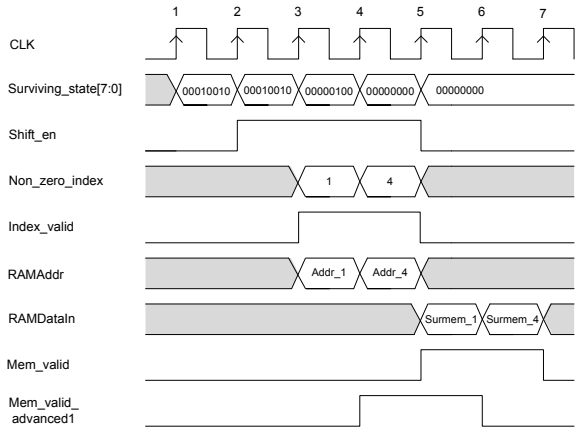

Figure 3.29: The waveform of the control signals in the SD_II controller

The control signals are used in the following way to compute and prune the block

- If Compute_finish = 1, the computing of current block is done, and the pruning of the last block is done. The state machine goes to GET_NEW_BLOCK state.
- If Compute_finish = 0
  - When the surviving list is not all zeros (clk 1-3 in the example)
    1. Perform shift algorithm on the surviving list to get the next non-zero index.In the example, the controller finds the surviving state: 1 and 4 of the block in two consecutive clock cycles.
    2. Drive the address of surviving state on the Surmem RAM address bus. The address is computed by adding the block offset with the state number in the block.
    3. If mem_valid is '1', capture the input data and feed it into the data path.

- When the surviving list is all zeros (clk 4-7 in the example)
  1. If index_valid is '1', read the last index out.
  2. If mem_valid is '1', capture the input data and feed it into the data path.
  3. If mem_valid_advanced1 is 1, one more valid data is on the way from the RAM. The state machine stays in the same state.
  4. If none of the three situation above is true, the controller waits the data to go through the pipeline. Then it checks the Prune_finish. If it is '1', the controller stores the new surviving path into the registers, and asserts the Compute_finish signal. Otherwise the state machine stays in the same state without performing any operation.

- If Prune_finish = 0
  - When the surviving list of the pruning block is not all zeros
    1. Perform shift algorithm on the surviving list to get the next non-zero index.
    2. Write the path metric of survived state and its parameters into the Stage Register Files.
  - When the surviving list of the pruning block is all zeros
    1. If index_valid is '1', read the last index out. Then the path metric of last survived state and its parameters are written into the Stage Register Files.
    2. If index_valid is '0', the pruning is finished. The Prune_finish signal is asserted.

Since the result of the computing process is stored into the same registers (Block Registers) that the pruning process uses, it will not perform the writing register operation until the pruning process is done. Therefore, Compute_finish signal will only be asserted when the Prune_finish is asserted.

3.5.2.2 Timing analysis

The timing of the SD_II controller unit is examined here. Still the worst case: 32QAM high symbol rate burst is calculated. As mentioned in the previous sections, it is assumed that a maximum of 64 states survive at each stage. And the situation that consumes longest time occurs when the surviving states exists in all 32 blocks. Assuming that each block has 2 surviving states. It takes 32 clock cycles to execute GET_NEW_BLOCK state, and 32*(2+5) clock

cycles to execute COMPUTE_BLOCK state, where 2+5 is the pipeline latency of processing 2 states. Together, the computation time for process a burst is

$$3 + [4 + 32 + (2 + 5) * 32 + 5 + (64 + 2)] * 171 = 55604cc$$

When using the clock with the maximum frequency = 200MHz, the clock period is 5ns, and the computation time is

$$55604 * (5 * 10^{-9}) = 283\mu s < \text{ required } 577/2\mu s$$

The calculation shows that the SD_II controller holds the timing requirements.

3.6 RSSE_T Equalizer core

3.6.1 Equalizer Design

The RSSE_T Equalizer is a hybrid equalizer, which uses the reduced trellis structure of the RSSE equalizer, and applies the threshold constraint in the path metric computation. So it is apparent that the RSSE_T equalizer should have a data path which is a combination of both RSSE and SD equalizers, shown in figure 3.30. It can be seen that most of the data path in SD equalizer are kept. Only the 32-input equalizer is replaced by the Parallel comparator and 8-input comparator. And the Merge path unit are changed back to the one used in the RSSE equalizer. The controller unit of the RSSE_T equalizer should be very similar to the one in the SD equalizer. A decision should be made before implementing the controller: is the SD_I controller or the SD_II controller a more suitable approach in this hybrid design?

From the description of the two SD equalizers, it can be seen that the SD_I equalizer is straightforward in implementation. The controller unit is much simpler than the SD_II equalizer. The disadvantage is that it takes too long time to calculate a trellis structure with many states. Since the RSSE trellis structure has significantly reduced states compared to the full trellis structure, the computation time problem in SD_I equalizer will not occur to the RSSE_T equalizer. Therefore the SD_I approach is selected.

The controller design has the same state diagram as for the SD_I controller. Minor modifications are applied on the SD_I controller:

- Pipeline stages are changed from 6 to 7, thus the controller signals propagated by the pipeline registers should be modified.

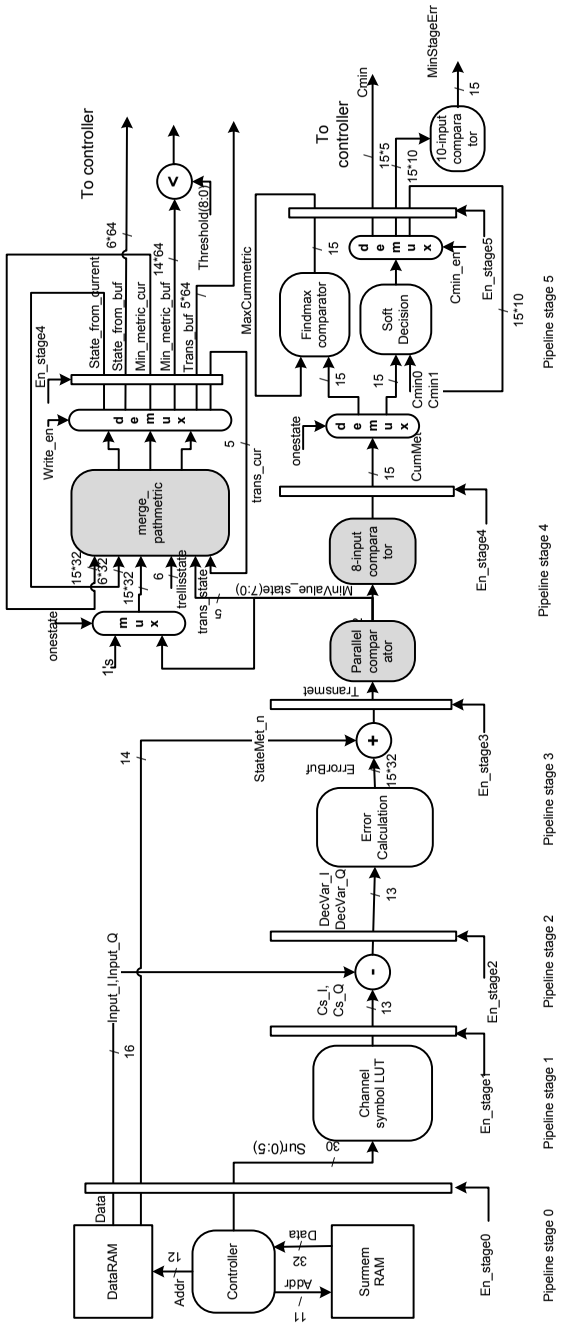

Figure 3.30: The data path of the RSSE_T equalizer

- Some signals have different length, e.g. the Trellis_state signal indicating the which state the controller is calculating is 10 bits in SD controller, and is 6 bits in RSSE_T controller.

3.6.2 Timing Analysis

Since the RSSE_T equalizer uses the SD_I approach, it is expected that it takes a little longer time than the RSSE equalizer, due to the overhead in block calculation. The time needed for processing a high symbol rate 32QAM burst is: $3 + [4 + (BlockSize + PipelineStage - 1) * TotalBlock + BlockSize + BitsperSymbol + TotalState + 2] * TotalSample$

$$= 3 + [4 + (8 + 6) * 8 + 8 + 5 + (64 + 2)] * 171 = 33348cc$$

When using the clock with the maximum frequency at 200MHz, the clock period is 5ns, and the computation time is

$$33348 * (5 * 10^{-9}) = 167\mu s < \text{ required } 577/2\ \mu s$$

For the RSSE_T equalizer, the worse case is the 64 state 8PSK burst, since it has 16 blocks in one stage. The computation time is 34180 clock cycles, which still holds the timing constraints when using 200MHz clock.

Chapter 4

Test and Performance Evaluation

4.1 Test Environment

In order to verify the function of the equalizer unit, a test environment has been set up. The test environment is made up of two parts: a Matlab Reference model and a test bench in VHDL, as shown in figure 4.1. The Equalizer unit in the hardware model is the entire unit that includes OCP interface controller, equalizer core and memory devices, as introduced in section 3.2.

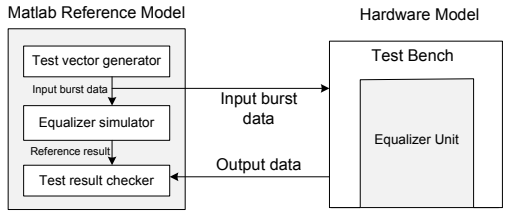

Figure 4.1: The test environment for the equalizer module

The Matlab Reference Model

As shown in the figure, the Matlab reference model has been implemented as a Test vector generator, an Equalizer simulator which simulates the behavior of the equalizer, and a Test result checker which compares the output soft-bit result from the hardware model with the reference result from the Equalizer simulator.

The Test vector generator can generates a burst of any required modulation type. The burst is formed by the correspondent tail bits of the modulation type, followed by an array of random data added with noise of a specified noise range. The burst data is saved as an *input.dat* file, which will be read by both the Equalizer simulator and the hardware model.

A separate Equalizer simulator is implemented for each of the algorithm implemented in the hardware model: RSSE, SD and RSSE_T. The equalizer simulator can be configured into processing any of the required modulation type data. Besides, a one-state equalizer simulator is also implemented to calculate the threshold used by the SD and RSSE_T equalizer simulators. The equalizer simulators function exactly the same as the corresponding hardware models. When they read the test burst data generated by the Test vector generator, they produce the reference soft-bit results. The results are stored into a *ref_out.dat* file, and will be used by the Test result checker.

The input burst data is also read by the test bench in the hardware model. The hardware model performs equalizing on the input data, and stores the soft-bit results in an *output.dat* file. This file is also used by the Test result checker. The test bench will be described in details in the next section.

The Test result checker compares the data of the *ref_out.dat* and the data of the *output.dat* file. If all the value are equal, the hardware model functions the same as the reference model, which is the expected outcome of the test.

Test bench of the Hardware Model

The test bench is implemented to verify the function of the equalizer. Besides, the test bench is also used for power evaluation of the design. When the system is powered up, it configures the equalizer unit according to the modulation type of the current burst, streams in the data from the *input.dat*, and sends it to the RAM of the equalizer unit via OCP interface. Then it waits until the equalizer finishes processing the burst, and asserts the *interrupt* signal. The test bench

then fetches data from the RAM of the equalizer, and saves the data into the *output.dat*.

4.2 Functional test

The functional test has been carried out for all 4 hardware implementation of the equalizers. The test cases cover all the required modulation types in two possible symbol rate. The test is made on both the RTL model and the gate level model with no annotated delay. The clock frequency is set at 200MHz. A list of the test cases are shown below in the table 4.1.

Not all the test case can be applied on all the equalizers. The test cases that do not match the algorithm of the equalizer are crossed out in the table. The 32QAM 1024 state case takes too long time for the SD_I equalizer to perform equalizing, which is known before the implementation. Other feasible test cases presented the correct results.

case	Modulation type	Total State	Total Sample	RSSE result	SD_I result	SD_II result	RSSE_T result
1	GMSK	16	143	correct	correct	correct	correct
2	GMSK	32	143	correct	correct	correct	correct
3	QPSK	64	171	correct	correct	correct	correct
4	8PSK	16	143	correct	-	-	correct
5	8PSK	64	143	correct	correct	correct	correct
6	16QAM	64	143	correct	-	-	correct
7	16QAM	64	171	correct	-	-	correct
8	16QAM	256	143	-	correct	correct	-
9	16QAM	256	171	-	correct	correct	-
10	32QAM	64	143	correct	-	-	correct
11	32QAM	64	171	correct	-	-	correct
12	32QAM	1024	143	-	violate timing	correct	-
13	32QAM	1024	171	-	violate timing	correct	-

Table 4.1: The test cases and the results

4.3 Timing analysis

The timing of the design is checked by Synopsys's *report_timing* function. Before synthesizing the design with the Synopsys DC, a timing constraint of clock cycle

equal to 5ns is set. After the synthesis, the Synopsys DC reports the delay of the critical path of the RSSE design is 4.87ns, which is under the required 5ns. Likewise, the delay of critical paths of the SD_I, the SD_II and the RSSE_T design are all under required 5ns. This shows the Synopsys DC is able to synthesize the models that can satisfy the timing requirements.

Normally a timed simulation is required to verify if the synthesized gate level model works with annotated delay. The delay information is gathered by the Synopsys DC tool and saved into a Standard Delay Format (SDF) file. The Modelsim can then perform the timing simulation with the gate level model and the SDF file. The timing simulation is not carried out in the project, due to the lack of layout level timing information.

In chapter 3, computation time analysis for each of the equalizer design is presented. The theoretical worst case computation time is calculated. In the simulation, the computation time of processing some test bursts is recorded to compare the efficiency of four equalizers. For the SD_I, the SD_II and the RSSE_T equalizers, the computation time depends very much on the number of the surviving states. Three cases are tested:
Minimum (Min): very few surviving states(1 state per stage).
Typical (Typ): medium number of surviving states.
Maximum (Max): near-maximal number of surviving states.

Since the burst data is generated as a sequence of random data with additive noise term, the absolute maximum number of surviving states is uncertain. Thus the situation of near-maximal number of surviving states is tested as an alternative.

The result is shown in table 4.2.

	RSSE	SD_I			SD_II			RSSE_T		
	-	Min	Typ	Max	Min	Typ	Max	Min	Typ	Max
GMSK	72	138	138	138	70	80	84	150	150	150
QPSK	258	348	352	370	100	150	252	376	380	398
8PSK	220	220	230	250	74	136	220	318	350	372
16QAM	230	586	608	620	90	212	252	236	270	280
32QAM	230	-	-	-	114	228	326	238	266	320

Table 4.2: The computation time of processing a burst by four difference equalizers($*10^2$ clock cycles)

4.4 Area analysis

The area of the equalizer design is examined by the *report_area* function of the Synopsys DC. As the equalizer core is the most interesting part to examine, the area measurements are made only on the equalizer core. The measurement below doesn't include the OCP interface controller or the memory device. The area cost of the equalizer core of four different algorithms is shown in table 4.3. It is evident that $Area_{RSSE} < Area_{RSSE\_T} < Area_{SD\_I} < Area_{SD\_II}$. The area of the SD_II equalizer is twice as much as the SD_I equalizer, and almost three times as much as the area of the RSSE equalizer.

The area of the sub-modules in the equalizer core are shown in table 4.4 and 4.5. The controller unit in the equalizer core is the biggest part in the equalizer design, most of which is contributed by the register files, as discussed in 3.4.4.7. The Error Calculation module which contains ten multipliers also contributes much to the total area cost.

Overall, RSSE and RSSE_T equalizers have lower area cost than the others.

	RSSE	SD_I	SD_II	RSSE_T
Nets	13726	25374	56790	19758
Cells	10056	18321	45522	13902
Combinational	35902	49416	110678	42383
Non-Combinational	26199	44634	72287	35933
Total Cell Area	62102	94050	182965	78317

Table 4.3: The area cost for the four equalizers of different algorithms

	RSSE		RSSE_T	
	area(μm^2)	%	area(μm^2)	%
Equalizer	62102	100	78317	100
ChannelSymbol LUT	2518	4.1	2542	3.2
Error Calculation	10703	17.2	10275	13
Merge Path	971	1.6	970	1.2
Soft Decision	1001	1.6	1001	1.3
Parallel Compare	4760	7.7	4756	6
8-input Compare	666	1.1	743	0.9
10-input Compare	-	-	1060	1.4
Controller unit	41422	66.7	57172	73

Table 4.4: The area cost of each module of the RSSE and the RSSE_T equalizers.

	SD_I		SD_II	
	area(μm^2)	%	area(μm^2)	%
Equalizer	94050	100	182965	100
ChannelSymbol LUT	2556	2.7	2543	1.4
Error Calculation	10339	11	9811	5.4
Merge Path	1830	1.9	3824	2.1
Soft Decision	1001	1.1	1001	0.5
32-input Compare	4581	4.9	4572	2.5
10-input Compare	1112	1.2	1103	0.6
Compute Shifter	-	-	925	0.5
Prune Shifter	-	-	925	0.5
Controller unit	72606	77.2	158265	86.5

Table 4.5: The area cost of each module of the SD_I and the SD_II equalizers.

4.5 Power consumption analysis

The power consumption is measured by the PowerTheater software. Similarly, only the power consumption of the equalizer core is examined. The OCP interface controller and the memory device are not included in the measurements.

The PowerTheater estimates the power consumption of the RTL level model. The RTL level model and the test bench is simulated by the Modelsim, and the switching activities during the simulation time are recorded in a Value Change Dump (VCD) file. The PowerTheater compiles the RTL level model, extracts the information from the standard cell libraries, and estimates the power consumption according to the switching activities recorded in the VCD file.

The power consumption of the four equalizer models are estimated separately, but using same test data: 3 bursts of each of the GMSK, QPSK, 8PSK, and 16QAM modulation types and 4 bursts of the 32QAM modulation types. The bursts have noise of different amplitude added, which will cause different numbers of surviving states per stage. For the RSSE_T equalizer, one additional burst of each of the 8PSK, 16QAM and 32QAM modulation types are tested, since the RSSE_T equalizer has a wider range of feasible surviving state per stage. For the RSSE equalizer, the power consumption is not related with the noise, therefore the bursts with the minimum noise of each modulation type are used for testing the RSSE equalizer.

The report generated by the Power Theater reports the static power dissipation and the dynamic power dissipation of the four equalizers. Table 4.6 shows the internal static power of each equalizer. The SD_II equalizer with the biggest

area cost has also the biggest internal static power dissipation, and the RSSE equalizer has the smallest static power dissipation.

	$P_{static}(\mu W)$
RSSE	49
SD_I	50
SD_II	114
RSSE_T	59

Table 4.6: The static power dissipation of the four equalizers(μW)

The dynamic power dissipation of the RSSE equalizer is shown in table 4.7. From the table, it is seen that the 32QAM burst consumes the most power. This is the same as expected, as the 32QAM is the modulation type of the largest alphabet the equalizer can handle, and it uses all the computational resource. For the bursts of smaller alphabet modulation types, not all the registers and logics are used. For example the Error Calculation module will use all the 32 output registers for the 32QAM burst, while only 16 of them are used for the 16QAM burst. Therefore the power dissipation of the smaller alphabet modulation types is less than that of the larger alphabet modulation types. The QPSK 64-state consumes more power than the 16QAM and 8PSK, because the QPSK has a 20% more samples in the burst. Therefore the payload vs. overhead is bigger than those two modulation types.

	$P_{dynamic}(mW)$
GMSK 16-state	7.02
GMSK 32-state	7.3
QPSK 64-state	8.15
8PSK 16-state	7.63
8PSK 64-state	7.88
16QAM 64-state	7.85
32QAM 64-state	8.19

Table 4.7: The dynamic power consumption of processing a burst by RSSE equalizers(mW)

The dynamic power dissipation of the SD_I and the SD_II equalizers is shown in table 4.8 and 4.9, respectively. The results are presented as dynamic power dissipation versus the average number of surviving states per stage. In both equalizers, the 8PSK consumes more power than the larger alphabet modulation types. It is because in the full trellis structure, the 8PSK has much smaller

number of total states (64) than 16QAM (256) and 32QAM (1024). For the same level of average surviving states, the surviving states distribute more closely in 8PSK trellis structure, and therefore the data path are enabled more frequently when processing 8PSK burst than 16QAM and 32QAM burst.

The results in table 4.8 and 4.9 are plotted in the figure 4.2. From the figures, it can be seen that the SD_I equalizer consumes less power than the SD_II equalizer in general.

	$P_{dynamic}$(mW) / Surv. State Per Stage		
GMSK 16-state	4.36 / 1	4.69 / 2	4.84 / 3
QPSK 64-state	4.07 / 1	4.77 / 4	5.73 / 13
8PSK 64-state	5.12 / 1	5.71 / 8	6.83 / 23
16QAM 256-state	3.48 / 1	5.07 / 17	5.43 / 25

Table 4.8: The dynamic power consumption of processing a burst by SD_I equalizers(mW)

	$P_{dynamic}$ (mW) / Surv. State Per Stage			
GMSK 16-state	6.6 / 1	7.1 / 2	7.3 / 3	-
QPSK 64-state	7.52 / 1	8.42 / 4	9.48 / 13	-
8PSK 64-state	7.9 / 1	9.51/ 8	10.3 / 23	-
16QAM 256-state	7.5 / 1	9.7 / 17	10 / 25	-
32QAM 1024-state	7.2 / 1	9.15 / 14	9.5 / 20	9.9 / 33

Table 4.9: The dynamic power consumption of processing a burst by SD_II equalizers(mW)

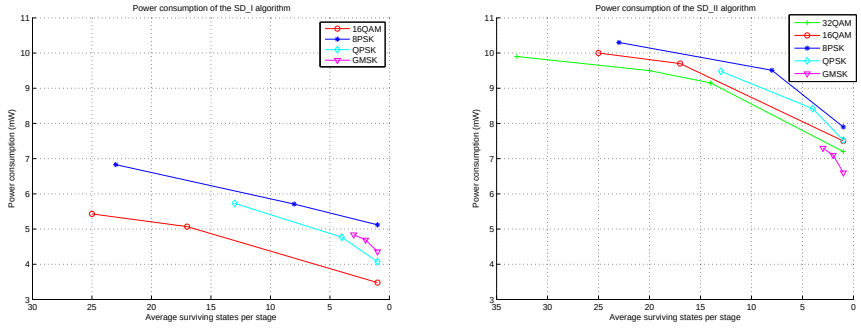

Figure 4.2: The dynamic power consumption of SD_I and SD_II equalizer when processing different types of burst

	$P_{dynamic}$ (mW)/ Surv. State Per Stage				
GMSK 16-state	3.76 / 1	4.08 / 2	4.22 / 3	-	-
QPSK 64-state	3.64 / 1	4.31 / 4	5.31 / 13	-	-
8PSK 64-state	3.71 / 1	5.35/ 14	5.87/ 23	6.53 / 38	-
16QAM 64-state	3.74 / 1	4.77 / 17	6.12 / 22	6.59 / 36	-
32QAM 64-state	4.19 / 1	5.49 / 14	6.25 / 22	6.78 / 32	7.77/ 38

Table 4.10: The dynamic power consumption of processing a burst by RSSE_T equalizers(mW)

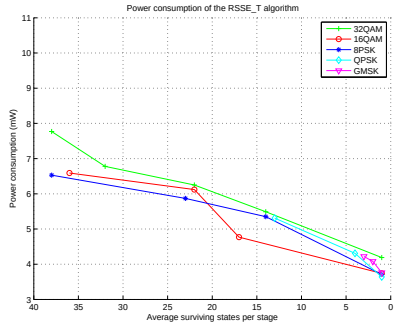

Figure 4.3: The power consumption of the RSSE_T equalizer

The dynamic power dissipation of the RSSE_T equalizer is also examined, and is shown in table 4.10. As the total states of the trellis structure are the same for the 32QAM, 16QAM, 8PSK and QPSK, the power consumption of these bursts are very close. The results in table 4.10 are plotted in figure 4.3.

To compare the power consumption of the different equalizers, the results of the same modulation types are compared. In figure 4.4, 4.5, and 4.6, the power consumption of the same modulation types burst are plotted. The red dashed line is the power consumption of the RSSE equalizer, which is independent of the number of surviving states. The magenta line is the SD_I equalizer, the blue line is the SD_II equalizer, and the green line is the RSSE_T equalizer. Here the difference of the power consumption of each equalizer can be seen. The power consumption of the SD_II equalizer varies from 20% over that of the RSSE equalizer to about 10% under that of the RSSE equalizer. The power consumption of the SD_I and the RSSE_T equalizer is close, and in the worst case is about the same as the RSSE equalizer. And when the number of surviving states per stage decreases to the minimum value of 1, the power consumption of these two equalizer is about half of the power consumption of the RSSE equalizer.

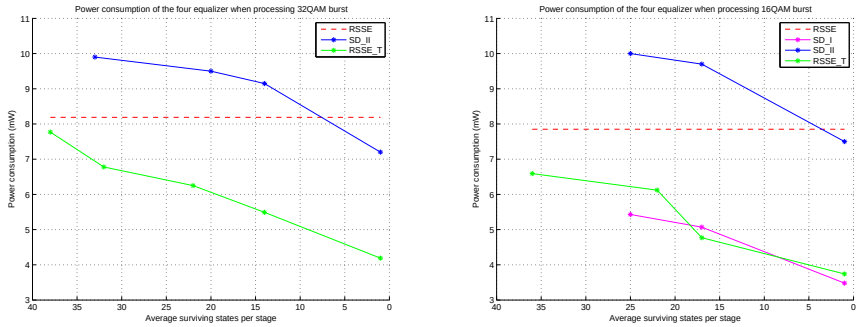

Figure 4.4: The dynamic power consumption of the four equalizers when processing 32QAM and 16QAM bursts

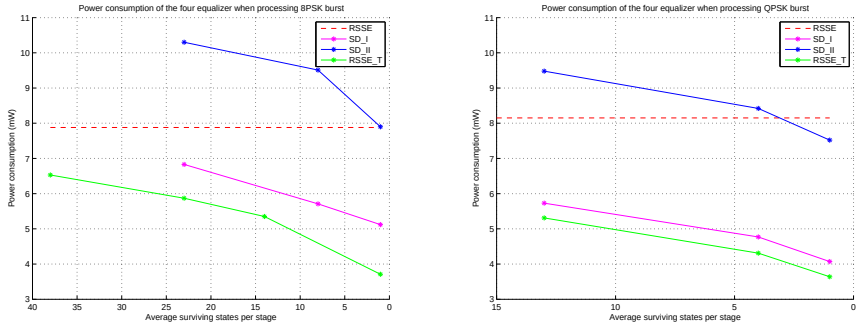

Figure 4.5: The dynamic power consumption of the four equalizers when processing 8PSK and QPSK bursts

Figure 4.7 combines all the power simulation results and plot them into one figure. In this figure, the different colors represent bursts of different modulation types. The dashed lines are the results of the RSSE equalizer. The dot-dash lines are the results of the SD_I equalizer. The solid thin lines are the results of the SD_II equalizer. And the solid thick lines are the results of the RSSE_T equalizer.

Finally, the power dissipation for each module in the equalizer core is examined. Here the test burst used is the 16QAM burst with surviving state per stage equal to 17. The result is shown in table 4.11. It is clear that most power is consumed by the Error Calculation module and the Controller unit. The equalizers which have the threshold constraint consume less power in the data path, since the data path is not toggled when processing the pruned state, and therefore it does not consume power when processing these states.

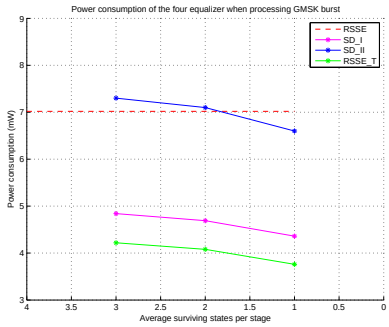

Figure 4.6: The dynamic power consumption of the four equalizers when processing GMSK bursts

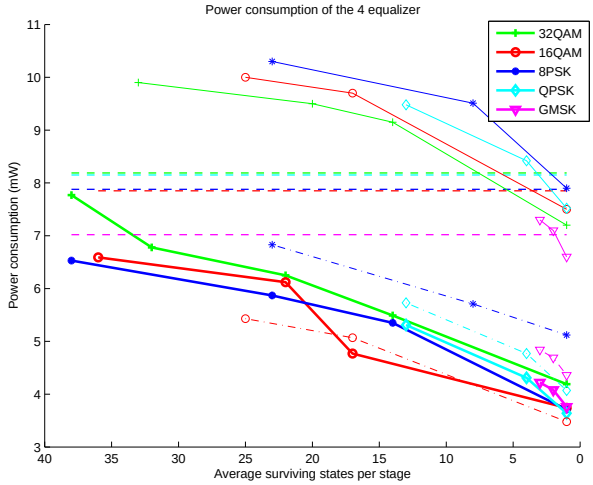

Figure 4.7: The dynamic power consumption of the four equalizer

Overall, without concerning BER, both SD_I and RSSE_T have lower power consumption than the others. However, the SD_I is not able to meet the timing requirements for 32QAM modulation scheme. Thus the RSSE_T is preferred over other algorithms in terms of power.

	P_{dyn_RSSE} (μW)	$P_{dyn\_SD\_I\_17}$ (μW)	$P_{dyn\_SD\_II\_17}$ (μW)	$P_{dyn\_RSSE\_T\_17}$ (μW)
Equalizer	7850	5070	9700	4770
ChannelSymbol LUT	101	99.6	89.9	97.7
Error Calculation	3950	2010	2860	2170
Merge Path	32.8	45.3	91.9	8.68
Soft Decision	66	5.46	18	4.51
Parallel Compare	426	-	-	275
8-input Compare	35.9	-	-	1.68
32-input Compare	-	85.9	225	-
10-input Compare	-	2.44	0.869	0.629
Compute Shifter	-	-	26.8	-
Prune Shifter	-	-	23.6	-
Controller unit	3238.3	2821.3	6363.9	2211.8

Table 4.11: The dynamic power dissipation of each module(μW), in 16QAM modulation type, with average surviving states of 17 in each stage

4.6 Energy consumption analysis

For mobile devices, the energy consumption is a crucial performance factor. The energy consumption is generally the sum of the power dissipation over time. The energy for processing a TDMA frame can be described as:

$$E = P_{dynamic} * t_{computation} + P_{static} * t_{idle} \tag{4.1}$$

Where E represents energy in Joule, P represents power in Watt, and t represents time in Second.

Two cases of energy consumption are calculated for evaluating the algorithms. The first case assumes that only one slot of a eight-slot frame carries data burst. The second case assumes that five slots in a frame carry data burst. In the first case, the computation time is only about 1/16 of the entire frame, since the burst is normally transmitted within 1/2 slot time, while the computation time in the second case is about 5/16 of the entire frame. Therefore the static power can have a relatively bigger contribution for the total energy consumption in the first case than in the second case.

As in the computation time measurement, three different levels of surviving states are calculated for the SD_I, the SD_II and the RSSE_T equalizers:
Minimum (Min): very few surviving states(1 state per stage).
Typical (Typ): medium number of surviving states.
Maximum (Max): near-maximal number of surviving states.

The calculated results of 1 slot per frame are shown in table 4.12 and figure 4.8. The calculated results of 5 slots per frame are shown in table 4.13 and figure 4.9.

	E_{RSSE} (μJ)	$E_{SD\_I}(\mu J)$			$E_{SD\_II}(\mu J)$			$E_{RSSE\_T}(\mu J)$		
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max
GMSK	0.47	0.52	0.55	0.56	0.75	0.81	0.83	0.53	0.56	0.57
QPSK	1.27	0.93	1.06	1.28	0.89	1.15	1.71	0.93	1.07	1.30
8PSK	1.08	0.78	0.88	1.08	0.81	1.16	1.65	0.84	1.27	1.46
16QAM	1.12	1.23	1.76	1.90	0.85	1.54	1.77	0.69	1.07	1.17
32QAM	1.16	-	-	-	0.93	1.56	2.12	0.75	1.08	1.49

Table 4.12: The energy consumption of processing 1 TDMA frame by four different equalizers(μJ), with duty cycle of 1 slot per frame

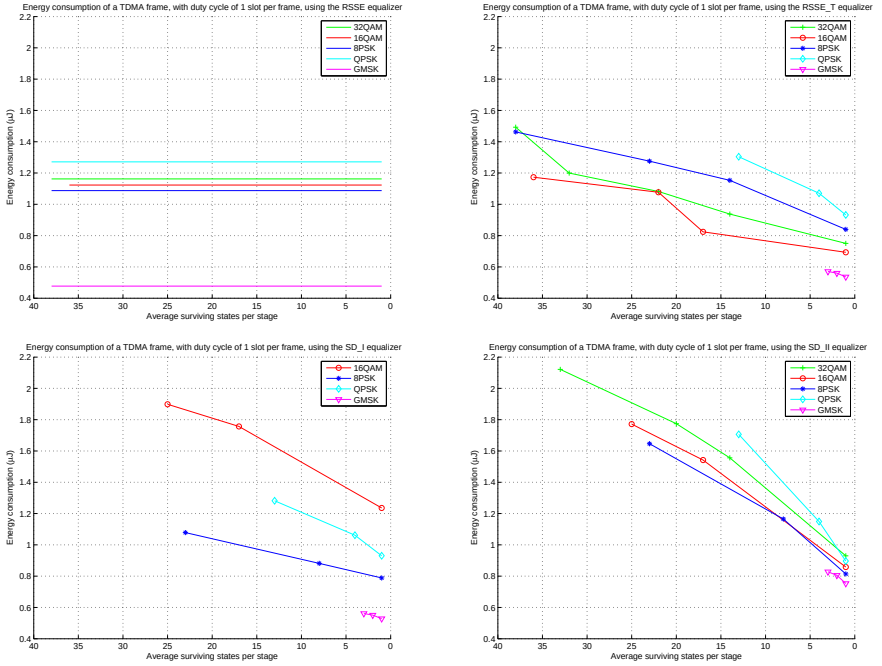

Figure 4.8: The energy consumption of processing 1 TDMA frame by four equalizers, with duty cycle of 1 slot per frame. Top left: RSSE. Top right: RSSE_T. Bottom left: SD_I. Bottom right: SD_II.

	E_{RSSE} (μJ)	$E_{SD\_I}(\mu J)$			$E_{SD\_II}(\mu J)$			$E_{RSSE\_T}(\mu J)$		
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max
GMSK	1.48	1.71	1.83	1.88	1.66	1.92	2.03	1.65	1.77	1.82
QPSK	5.45	3.72	4.38	4.47	2.38	3.64	6.43	3.63	4.32	5.48
8PSK	4.53	3.02	3.48	4.47	1.97	3.72	6.13	3.16	5.34	6.28
16QAM	4.71	5.25	7.86	8.57	2.19	5.60	6.75	2.43	4.35	4.83
32QAM	4.91	-	-	-	2.54	5.67	8.50	2.72	4.38	6.43

Table 4.13: The energy consumption of processing 1 TDMA frame by four different equalizers(μJ),with duty cycle of 5 slot per frame

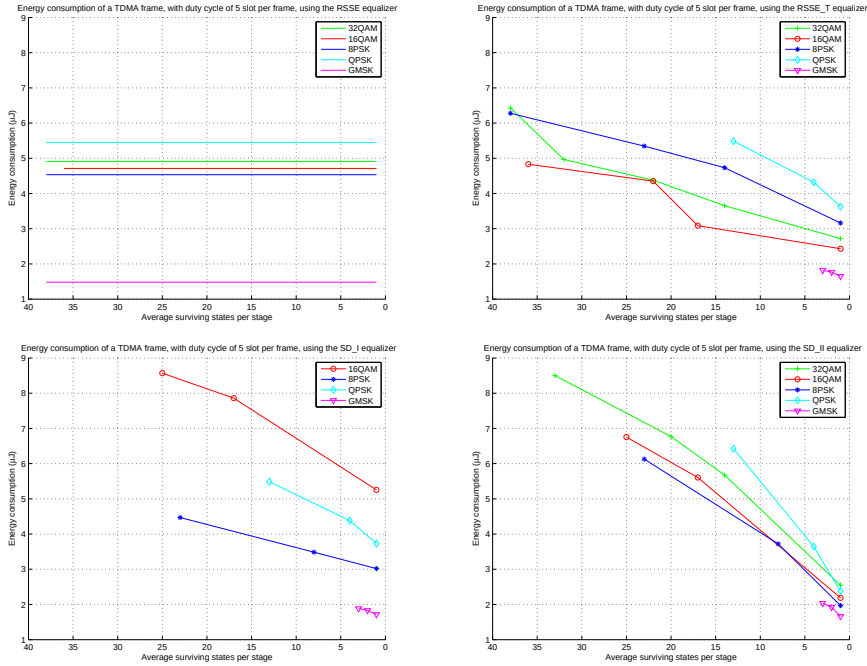

Figure 4.9: The energy consumption of processing 1 TDMA frame by four equalizers, with duty cycle of 5 slots per frame. Top left: RSSE. Top right: RSSE_T. Bottom left: SD_I. Bottom right: SD_II.

As shown in the figures of the energy consumption, the energy of the second case of 5 slot per frame has bigger variation range than that of the first case. This is because the static power consumption has less contribution to the second case.

For the RSSE_T equalizer, the minimum energy consumption is about 50%

of the maximum energy consumption, and the typical energy consumption is almost the same as the energy consumption of the RSSE equalizer. For the SD equalizers, the energy consumption varies significantly with the number of the surviving states. Especially for the SD_II equalizer in the case of 5 slots per frame, the minimum energy consumption is only about 30% of the maximum energy consumption.

Comparing the energy consumption of the SD_II and the RSSE_T equalizers, it can be seen that the SD_II equalizer has better energy consumption when the number of the surviving states per stage is under 5. However, when the number of the surviving states increases, the energy consumption of the SD_II equalizer quickly exceeds that of the RSSE_T equalizer, and can reach as much as 150% of that of the RSSE_T equalizer. Since the optimal number of surviving states per stage of the equalizer varies typically from 5 to 20, the RSSE_T equalizer is a preferable choice in the terms of energy consumption.

Chapter 5

Conclusion

This thesis concentrates its effort on finding a power/area optimal algorithm to implement an equalizer which supports the GSM standard release 7. Two algorithms: RSSE and SD, are studied, and a hybrid algorithm: RSSE_T is proposed. These algorithms are expected to achieve the performance of the MLSE solution in a reduced computation complexity. Four hardware equalizer models are implemented and compared: one for the RSSE algorithm, two different models for SD algorithm, and one for the RSSE_T algorithm.

The RSSE algorithm reduced the complexity of MLSE by exploiting a reduced-state trellis structure, which groups the states into subsets by Ungerboeck's set-partitioning method. The simulation results show that the total cell area of the RSSE equalizer is 62102 μm^2. The RSSE equalizer has a maximal power consumption of 8.2 mW when processing 32QAM burst, and a minimum power consumption of 7 mW, when processing GMSK burst. The computation time of processing a burst is 23000 clock cycles in the worst case of the test.

The SD algorithm solves the problem by using a threshold constraint in the trellis search. It only calculates the states in a MLSE trellis structure of which the state metric is beneath the threshold. Two different approaches are used when implementing the SD algorithm. The first approach, SD_I equalizer, checks the status of each state in the trellis and performs calculation on the surviving states. The second approach, SD_II, searches for valid states, and does not

spend any clock cycle on the pruned states.

From the simulation, the SD_I equalizer is more optimal in the area cost, which is 94050 μm^2 compared to 182965 μm^2 for the SD_II equalizer. For the power consumption, both have decreasing power consumption when the average surviving states per stage decreases. For the SD_I equalizer, the worst case power consumption is 6.8 mW for an average surviving state of 23 per stage. And in the situation of only one surviving state per stage, the power consumption can be reduced to about 3.5 mW. For the SD_II equalizer, the worst case is 10.3mW for the same amount of the surviving state as in SD_I, and can be reduced to about 7.5 mW in the situation of only one surviving path. The result shows that the SD_I equalizer is better than the SD_II equalizer in both area and power consumption. The disadvantage of the SD_I equalizer is that the computation time of processing a burst is longer. In the worst case, the SD_I equalizer takes 62000 clock cycles to finish a 16QAM burst, while the SD_II equalizer only needs 25200 clock cycles to process the same burst. Besides, the calculation showed that design of the SD_I equalizer cannot process a 32QAM burst in a required 1/2 GSM slot time of 288 μs.

Since it is feasible to combine the complexity-reducing methods of the RSSE algorithm and the SD algorithm, a hybrid algorithm, RSSE_T is proposed and implemented in this work. This algorithm both exploits the reduced trellis structure in the RSSE, and constrains the trellis search with a threshold. The simulation results show that the RSSE_T equalizer has an area cost of 78317μm^2, which is about 25% more expensive than the RSSE equalizer. But it is only 80% the area cost of SD_I and 42% the area cost of the SD_II equalizer. The power consumption of the RSSE_T also decreases when the number of the surviving states decreases. The worst case in the test is 7.7 mW for 38 surviving states, and for the most optimal case of 1 surviving path is about 3.6 mW, which is below 50% of the power consumption of the RSSE equalizer. Compared to the SD_I equalizer, the RSSE_T equalizer has very close power consumption to the SD_I equalizer, but the computation time of processing a burst is shorter. It is able to process a burst of any required modulation type within the 1/2 GSM slot time.

A side-by-side comparison shows that the energy consumption of the SD_I, the SD_II and the RSSE_T equalizers decrease when the number of surviving states decreases. The RSSE_T equalizer has the advantage of having an overall lower energy consumption when the number of surviving states is within an optimal operation range. Even though in extreme cases the SD_II equalizer has lowest energy consumption, the number of surviving states in those extreme cases is too low to guarantee the path extension can continue with the presence of noise.

To summarize the performance of the four equalizer:

- All four equalizers have achieved the functional requirements described in section 3.1 of chapter 3.
- The RSSE equalizer has the minimum area cost among the four equalizers, while the SD_II equalizer has the maximum area cost.
- The SD_II equalizer has the biggest average power consumption. In the most optimal case, its power consumption is of the same level of that of the RSSE equalizer. In the worse case, it is 25% over that of the RSSE equalizer.
- The SD_I and the RSSE_T equalizers have very close power consumption, and in the worst case close to that of the RSSE equalizer. In the most optimal case, the power consumption is about 50% of that of the RSSE equalizer.
- The power consumption of the RSSE_T equalizer is constantly 30% lower than the SD_II equalizer.
- The RSSE, the SD_II and the RSSE_T equalizers meet the timing requirements, while the SD_I equalizer only does in some of the modulation types.
- The energy consumption of the RSSE_T equalizer is lower than the others under realistic operating conditions.

Based on the simulation result, it can be concluded that the hybrid design: RSSE_T equalizer combined the beneficial factors of RSSE and SD algorithm. It has slightly higher cost in area than the RSSE algorithm, but in return it achieved the best performance in the power/energy consumption. Therefore the RSSE_T is an efficient hardware implementation of the equalizer that is able to perform equalization for the data of GMSK, QPSK, 8PSK, 16QAM and 32QAM modulation types.

5.1 Future work

To further optimize the design described in this thesis, following tasks could be carried out:

- Register balancing. Most of the pipeline registers are set between two modules, so the organizing of the pipeline stages is not very optimal. The synthesis result shows that latency of some pipeline stages are much longer

than the others. The pipeline should therefore be balanced according to the synthesis result.

- Deeper pipelining. Both the area and the power cost suggest that the Error Calculation module contributes a lot to the final results. The Error Calculation module contains several multipliers, and it is placed in a single pipeline stage. More pipeline stages are suggested for this module, so as to reduce the delay.

- In the SD_II design, a normal 32-bit shifter is used to detect the first non-zero element. Some area-optimal shifter algorithms can be used instead. For example, a 32-bit Barrel shifter which costs 160 multiplexer is more optimal, comparing to a normal shifter which costs 512 multiplexer.

- The shift algorithm in the SD_II equalizer can be applied in RSSE_T equalizer to further reduce the computation time. The change in the energy consumption is interesting to study.

- As described in chapter 3, the memory allocation of the SD equalizers is straightforward, but not efficient in the case of low number of surviving states per stage. It is possible to design an advanced memory indexing mechanism and store the data in the consecutive sequence. Thus the memory will become smaller. However, it is difficult to estimate which approach has the better performance, since it is a complexity-area trade-off.

Appendix A

Tail bits definition

Modulation type	Tail bits 0	Tail bits 1	Tail bits 2	Tail bits 3
GMSK	0	0	0	-
8PSK	1,1,1	1,1,1	1,1,1	-
16QAM normal rate	0,0,0,1	0,1,1,0	0,1,1,0	-
32QAM normal rate	1,1,1,1,0	0,1,1,1,0	0,1,1,1,0	-
QPSK	0,0	0,1	1,1	1,0
16QAM higher rate	0,0,0,1	0,1,1,0	0,1,1,0	1,1,0,1
32QAM higher rate	1,1,1,1,0	1,1,1,1,0	0,1,1,1,0	0,1,1,1,0

Table A.1: Tail bits of each modulation type

Appendix B

Diagrams of the equalizer

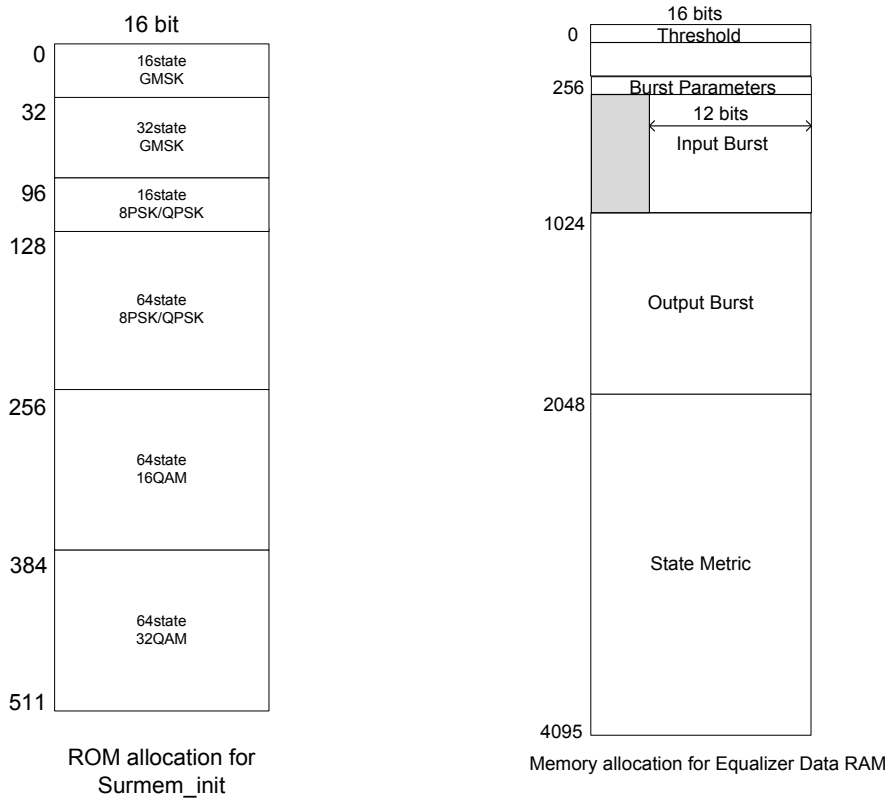

Figure B.1: Left: The ROM module of the RSSE equalizer Right: The data RAM module of the four equalizers

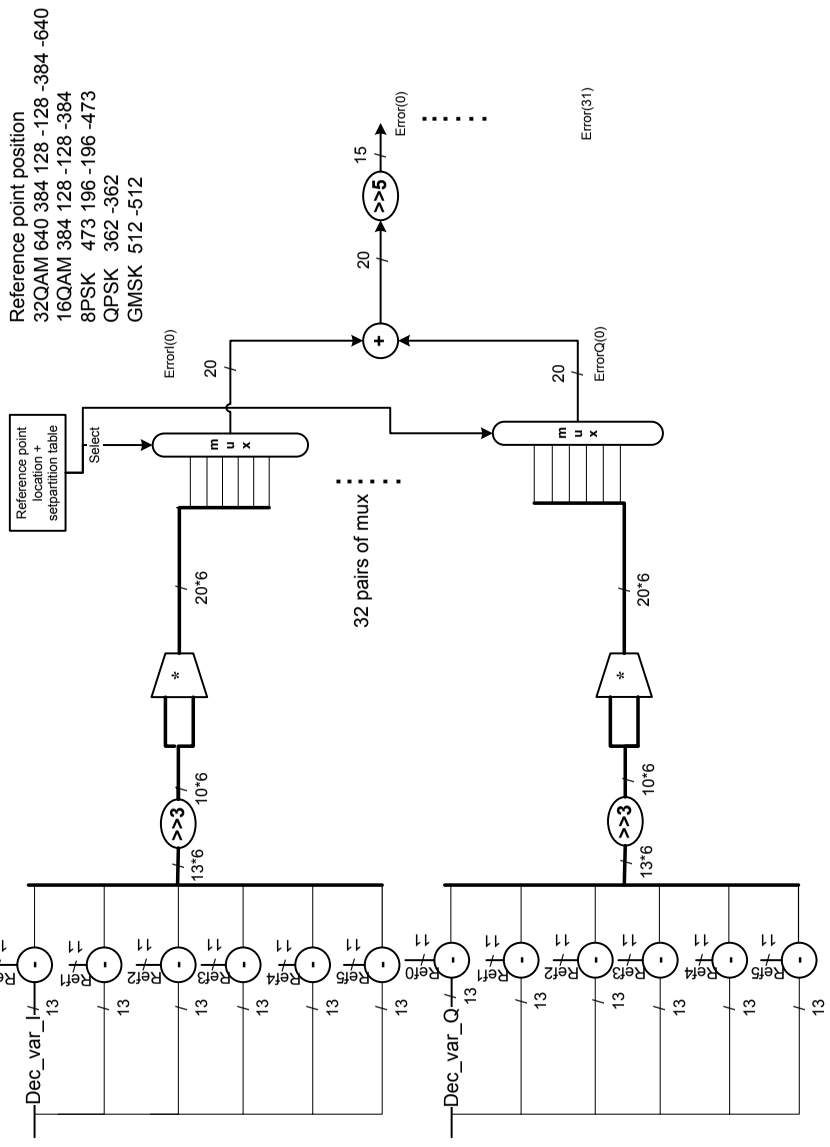

Figure B.2: The Error Calculation module of four equalizers

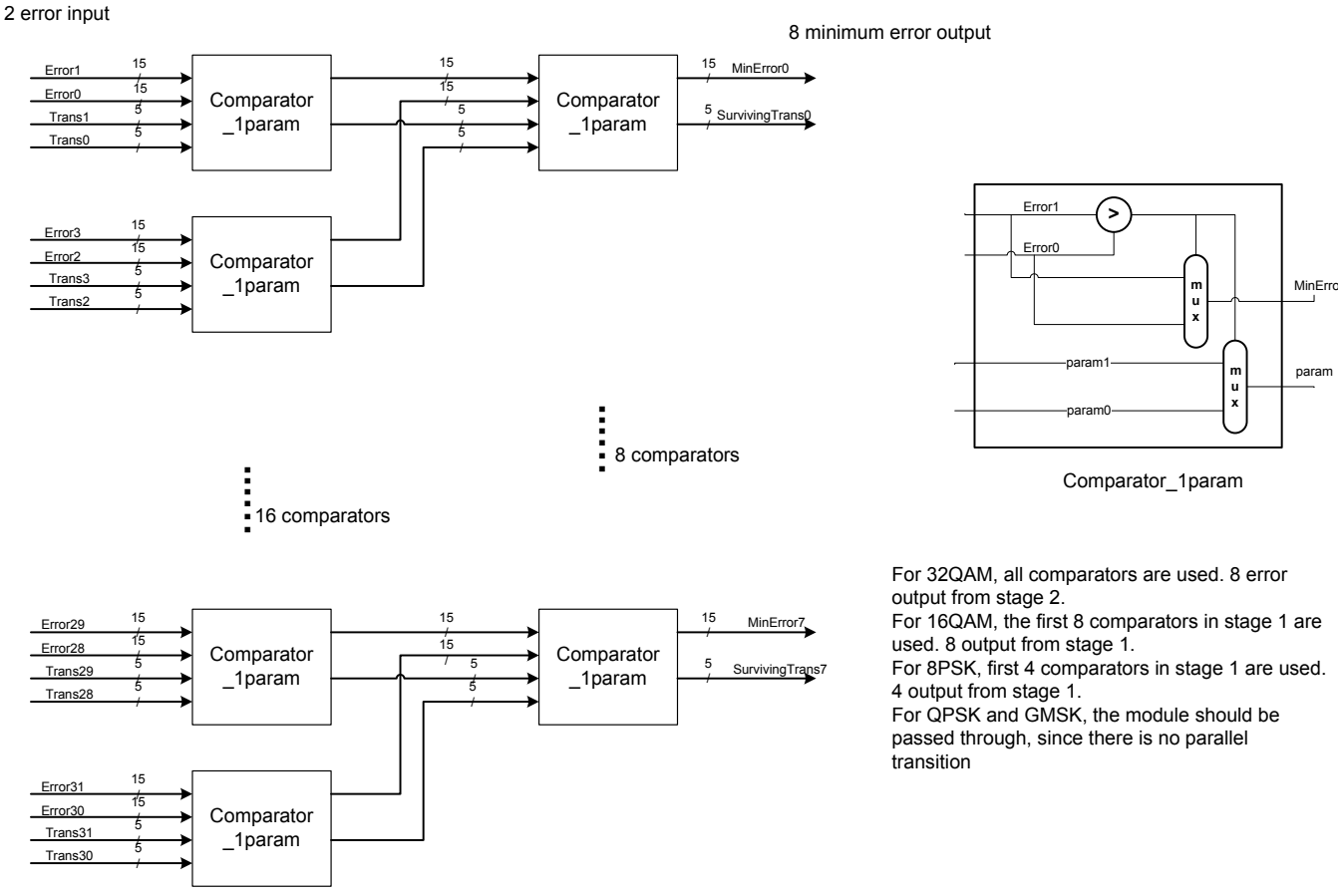

Figure B.3: The Parallel Compare module of the RSSE equalizer

Bibliography

[1] J.G.Proakis, "Digital Communications". McGraw Hill, 4th edition, 2001

[2] G.D.Forney.Jr.,"Maximum-likelihood sequence estimation of digital sequence in the presence of intersymbol interference". IEEE Transactions on Information Theory,VOL.IT-18,pp.363-378,May 1972

[3] A.D.Viterbi, "Error Bounds for Convolutional Codes and Asymptotically Optimum Decoding Algorithms". IEEE Transactions on Information Theory,pp.260-269,April 1967.

[4] M.V.Eyuboglu, S.U.H.Qureshi, "Reduced-State Sequence Estimation with Set Partitioning and Decision Feedback". IEEE Transactions on communications, VOL.36, No.1, January 1988

[5] B.Hassibi and H.Vikalo, "On the Sphere-Decoding Algorithm. I.Expected Complexity". IEEE Transactions on Signal Processing, pp. 2806-2818, August 2005.

[6] B.Hassibi and H.Vikalo, "On the Sphere-Decoding Algorithm. II. Generalizations, Second-Order Statistics, and Application to Communications". IEEE Transactions on Signal Processing, pp. 2819-2834, August 2005.

[7] 3rd Generation Partnership Project; Technical Specification Group GSM/EDGE Radio Access Network; Modulation(Release 7) 3GPPTS45.004 v7.2.0(2008-02)

[8] 3rd Generation Partnership Project; Technical Specification Group GSM/EDGE Radio Access Network; Multiplexing and multiple access on the radio path (Release 7) 3GPPTS45.002 v7.6.0(2007-11)

[9] G Ungerboeck, “Trellis-Coded Modulation with redundant Signal Sets Part I: Introduction”. IEEE communications Magazine February 1987-Vol.25,no 2

[10] G Ungerboeck, “Trellis-Coded Modulation with redundant Signal Sets Part II: State of the Art”. IEEE communications Magazine February 1987-Vol.25,no 2

[11] M Hansen, “Comparison of Optimal and Near-Optimal Detection in GSM/EDGE”. Master thesis, Infomatics and Mathematical Modelling, DTU, 2006

[12] K Wu, “Optimal Algorithms for GSM Viterbi Modules”. Master thesis, Infomatics and Mathematical Modelling, DTU, 2003

[13] “Open Core Protocol Specification 2.1” OCP-IP Association 2005

[14] C Langton “Intuitive Guide to Principles of Communications”. Available on the internet at http://www.complextoreal.com

[15] R. Booth “A note on th application of the Laurent Decomposition to the GSM GMSK signaling waveform”. Tropian, Inc Link: http://www.tropian.com/wps/wselaurent1.pdf

[16] P. J. Ashenden. “The designer’s guide to VHDL”. Morgan Kaufmann, 2nd edition, 2002.

[17] J. F. Wakerly. “Digital design principles and practices”. Prentice Hall, 3rd edition, 2000.

[18] N. H. E. Weste and K. Eshraghian. “Principles of CMOS VLSI design”. Addison Wesley, 2nd edition, 1993.

[19] B. Sklar. “Digital Communications - Fundamentals and Applications”. Prentice Hall ISBN 0-13-211939-0, 2nd edition, 1988.

[20] S. G. Wilson. “Digital Modulation and Coding”. Prentice Hall, 1st edition, 10, 1998.