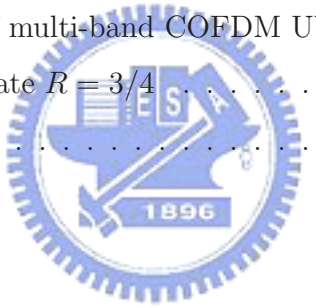
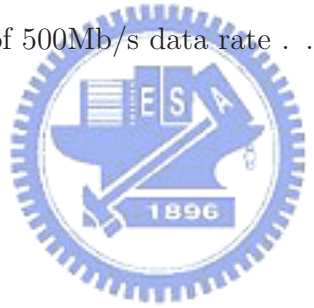
Contents

List of Figures



List of Tables



Chapter 1

Introduction

1.1 Research Motivation

The fundamental block diagram of a typical digital communication system is shown in Figure 1.1. Singnal transformation from the information source to the transmitter includes source encoding, channel encoding and modulation. The receiver will reverse the signal transformation by demodulation, channel decoding and source decoding. When a signal passes through the channel, it may be influenced by various type of noise disturbances such as channel noise, interference and fading. In order to eliminate the effects of noise disturbances, the channel encoder transforms the source codeword into the channel codeword by adding certain structural redundancy. These redundant bits can be used for detecting and correcting the errors. Theoretically, the encoding procedure provides the encoded signal with better distance properties than the un-coded one, and thus channel coding can improve the performance of the overall system.

There are two structurally different types of channel codes, the block codes and the convolutional codes. For the block code, the encoder transforms a message with k symbols into a codeword sequence with n symbols. The $n - k$ redundant symbols called parity-check depend only on the corresponding k message symbols and not on any other message symbols. Therefore the block code is memoryless. For the convolutional code,

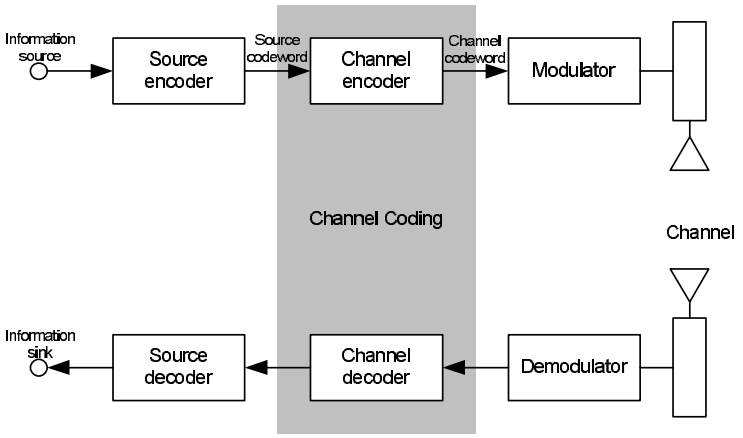


Figure 1.1: Block diagram of a typical digital communication system

the encoder contains memory units. This causes the output symbols depend not only on the current input message but also on the previous input messages. The (n, k, ν) convolutional encoder can be implemented with n-output, k-input, and ν-memory words as shown in Figure1.2.

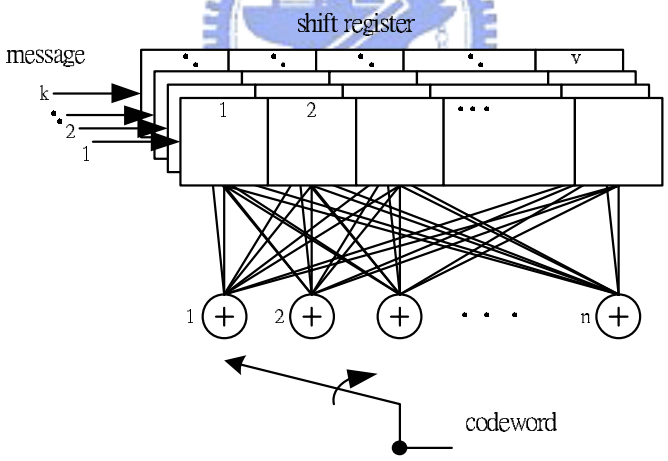


Figure 1.2: The (n, k, ν) convolutional encoder

In 1976, Viterbi [1] introduced a decoding algorithm for convolutional code. And Omura [2] showed that the Viterbi Algorithm was equivalent to a dynamic programming solution to the problem of finding the shortest path through a weighted graph. For-

ney [3] later recognized that it was in fact a maximum likelihood decoding algorithm. Until now, Viterbi algorithm is still the optimum solution for convolutional code and has been widely applied to decoding convolutional codes and signal detection in many digital communication and magnetic storage applications.

Recently, high-speed Viterbi decoder becomes more important and critical because of the ever increasing high-speed data transmission and the demanding error performance in modern digital communication systems. The huge bandwidth (3.6GHz∼10.1GHz) of these ultra-wideband(UWB) communication systems enables short-range and very high-speed data transmission. In the physical (PHY) layer proposal [4] based on multi-band orthogonal frequency-division multiplexing (MB-OFDM) technology, a convolutional code with the constraint length $K = 7(= \nu + 1)$ has been specified to support a maximum 480Mb/s data rate after puncturing to the rate (R) of 3/4. Furthermore, PHY-layer proposal employing the direct sequence UWB (DS-UWB) modulation [5] defines both $K = 6$ and $K = 4$ convolutional codes for the 500Mb/s and the over 1Gb/s data rates respectively. Accordingly, the Viterbi decoders for convolutional codes targeting to these systems have arisen great research interest [6–8].

Though the minimum distance of a convolutional code increases linearly with the constraint length $K(= \nu + 1)$, the computing complexity grows exponentially while applying the Viterbi decoding. Consequently, the very large scale integration (VLSI) implementation of Viterbi decoder for high-speed wireless applications that adopt large convolutional codes ($K \geq 7$) is still challenging as the power and cost constraints are considered. Therefore, this thesis will propose a high-speed and area-efficient solution for Viterbi decoder design.

1.2 Paper Survey

In early research of the Viterbi decoder, because of the bottleneck of the VLSI technology the key point always focused on the complexity. As the rapid development of the VLSI technology, the research interests changed to achieve the higher throughput rate.

The architectures using high-radix trellis in [7–10] achieve high speed through M steps of lookahead where the throughput of a Viterbi decoder will be enhanced by a factor of M. However, the ideal speedup is difficult to achieve due to the exponentially increasing number of branches on the high-radix trellis, limiting M to be at most two in most designs. The Viterbi decoders in [11–13] break down the critical path delay by means of bit-level pipeline and accomplish high throughput with very high clock frequencies. Furthermore, the dynamic circuit techniques are also exploited to accelerate the critical path. The four states Viterbi decoder based on sliding block approach that performs decoding concurrently in forward and backward directions is also reported in [14]. However, as the constraint length increases, the complexity grows rapidly because of the highly parallel architecture and large skew buffers. Fig. 1.3 summarizes the performance among various high-speed Viterbi decoder designs.

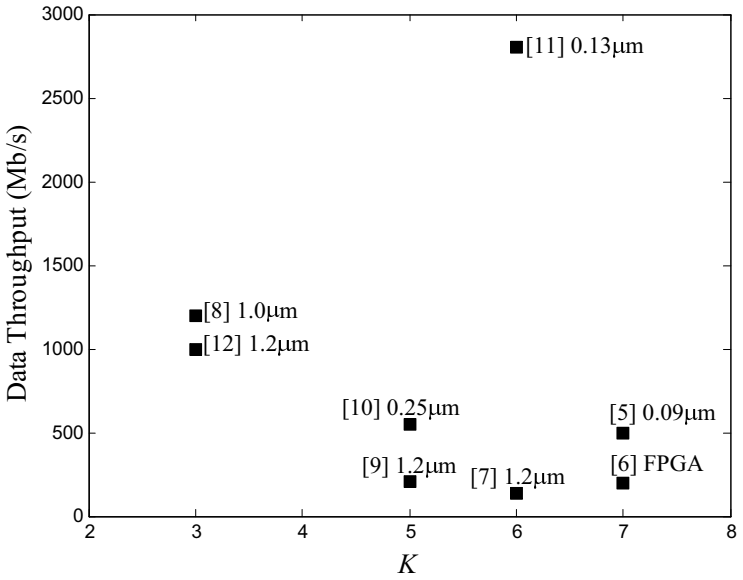


Figure 1.3: Performance of several published Viterbi decoders

1.3 Thesis Organization

The Viterbi decoder contains three main units: branch metric unit (BMU), add-compare-select unit (ACSU), and survivor memory unit (SMU). The BMU calculates the branch metrics from the input data. The ACSU recursively accumulates branch metrics (BM) as path metrics (PM), and makes decisions to select the most likely state transition sequence. Finally, the SMU traces the decisions to extract this sequence. It is the nonlinear and recursive nature of ACSU that limits the maximum achievable throughput rate.

Considering the speed of a Viterbi decoder, this thesis will focus on the improvement of most timing-critical processing unit ACSU. The speedup is accomplished by retiming techniques to parallelize the serial add, compare, and select operations based on the two-dimensional (2-D) structured ACSU. In chapter 2, the principle of convolutional code and Viterbi algorithm is reviewed. Several typical architectures for Viterbi decoder will also be discussed. chapter 3 presents a more aggressive trellis expansion that attains to the $M \geq 4$ steps lookahead and the retiming techniques for 2-D ACSU. In addition, chapter 4 reports the implementation results which target at the ultra-wideband system, including system parameters selection, post-layout simulations of the 0.13-μm and the 0.18-μm designs. Finally, the conclusion and future research plans is given in chapter 5.

Chapter 2

Convolutional Code

2.1 Convolutional Code

To describe a convolutional code, at first it is necessary to characterize the encoding process. Several methods can be used for representing the encoding process of the convolutional code. The time-domain description and trellis diagram description would be described in the following subsection.

2.1.1 Time-Domain Description

A simple convolutional encoder is shown in Figure 2.1. The figure illustrates a $(2, 1, 2)$ convolutional encoder with 2 shift-registers, and 2 modulo-2 adders.

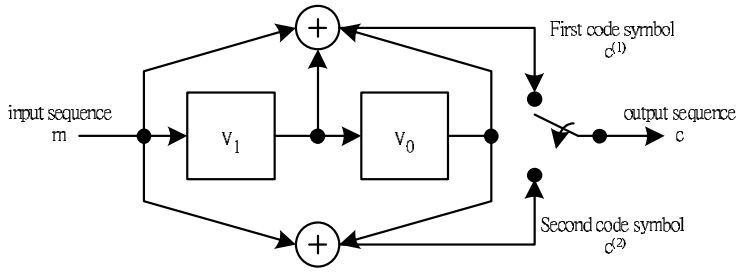


Figure 2.1: The (2, 1, 2) convolutional encoder

The input of this encoder is some binary sequence, $m = (..., m_{-1}, m_0, m_1, m_2, ...)$. The output is an interleaved sequence $c = (..., c_{-1}^{(1)}, c_{-1}^{(2)}, c_0^{(1)}, c_0^{(2)}, c_1^{(1)}, c_1^{(2)}, ...)$ of the two binary sequence $c^{(1)}$ and $c^{(2)}$. At each input bit time, the code symbol $c_i^{(1)}$ and $c_i^{(2)}$ are generated by the encoding function

$$c_i^{(1)} = m_i \oplus m_{i-1} \oplus m_{i-2} \tag{2.1}$$

$$c_i^{(2)} = m_i \oplus m_{i-2} \tag{2.2}$$

where $\oplus$ denotes modulo-2 adder (or the XOR operation). Next, the input bit is shifted into the leftmost stage and the bits in the register are shifted one position to the right. Consequently, the output sequence c depends not only the present input bit m_i, but also on the two previous input bits m_{i-1} and m_{i-2}. Evidently, the different interconnection of the encoder influences the codeword sequence. For mathematical computing convenience, these interconnections can be formulized as the generator sequence

$$g^{(1)} = (g_0^{(1)}, g_1^{(1)}, g_2^{(1)}) \tag{2.3}$$

$$g^{(2)} = (g_0^{(2)}, g_1^{(2)}, g_2^{(2)}) \tag{2.4}$$

where $g_i^{(1)}$ represents the upper and $g_i^{(2)}$ represents the lower interconnections from left to right. Then, the encoding process can now be written as

$$c^{(1)} = (..., m_{-1}, m_0, m_1, m_2, ...) * (g_0^{(1)}, g_1^{(1)}, g_2^{(1)}) \tag{2.5}$$

$$c^{(2)} = (..., m_{-1}, m_0, m_1, m_2, ...) * (g_0^{(2)}, g_1^{(2)}, g_2^{(2)}) \tag{2.6}$$

where $*$ denotes the convolution operator.

For example, Figure 2.1 can be described by

$$g^{(1)} = (111) \tag{2.7}$$

$$g^{(2)} = (101) \tag{2.8}$$

In general, the (n, k, ν) convolutional encoder is specified by a set of n generator

sequences with length $(\nu + 1)$

$$\begin{cases} g^{(1)} &= (g_0^{(1)}, g_1^{(1)}, ..., g_\nu^{(1)}) \\ g^{(2)} &= (g_0^{(2)}, g_1^{(2)}, ..., g_\nu^{(2)}) \\ &\vdots \\ g^{(n)} &= (g_0^{(n)}, g_1^{(n)}, ..., g_\nu^{(n)}) \end{cases} \tag{2.9}$$

Then the output sequences is determined by convolving the input sequence and the generator sequences

$$\begin{cases} c^{(1)} &= m * g^{(1)} \\ c^{(2)} &= m * g^{(2)} \\ &\vdots \\ c^{(n)} &= m * g^{(n)} \end{cases} \tag{2.10}$$

The encoding process can be determined in a matrix form as

$$c = mG \tag{2.11}$$

2.1.2 Trellis Diagram Description

The convolutional encoder belongs to a class of devices known as finite state machines. Thus, a convolutional encoder can be specified completely by the state diagram. In Figure 2.1, the states of the encoder is defined as the pair (m_{i-1}, m_{i-2}) of the shift-register, hence there are four possible states, 00, 01, 10, 11. At each tick of the clock, the encoder accepts an input m_i, and emits the two code symbols $c_i^{(1)}$ and $c_i^{(2)}$. Then the state transforms from a pair (m_{i-1}, m_{i-2}) to a new pair (m_i, m_{i-1}). The state diagram representation is shown in Figure 2.2. The four circles mean the four states, 00, 01, 10, 11. A transition from one state to another corresponding to an input bit of "0" is represented by a dotted line. Similarly, a transition corresponding to "1" is represented by a solid line. And the label on the line represents the code symbols $(c_i^{(1)}, c_i^{(2)})$ from one state to another.

With the help of the state diagram, it is easy to determined the output sequence of the encoder. For example, if the input sequence is (1011100...). The transition starts at state 00 and walks through the state diagram corresponding to a solid line if the input bit

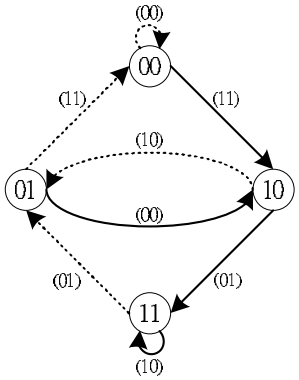


Figure 2.2: State diagram for the encoder of Figure 2.1

is "1", and a dotted line if that is "0". The track of the transition is shown in Figure 2.3. Following the track, the output sequence is $(11, 10, 00, 01, 10, 01, 11, ...)$. Consequently, that is the same output sequence in the time-domain description.

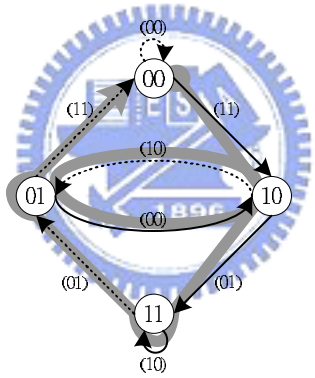


Figure 2.3: The track of the transition

When the input sequence becomes large, the track will travel the same edge many times. It is difficult to keep track of where we have been. Therefore, a representation called a trellis diagram is obtained directly from the state diagram by including the dimension of time. Once again the output sequence for the input sequence (1011100...) is represented in Figure 2.4.

In general, there are 2^{ν} states and 2^{kl} kinds of codeword corresponding to the (n, k, ν)

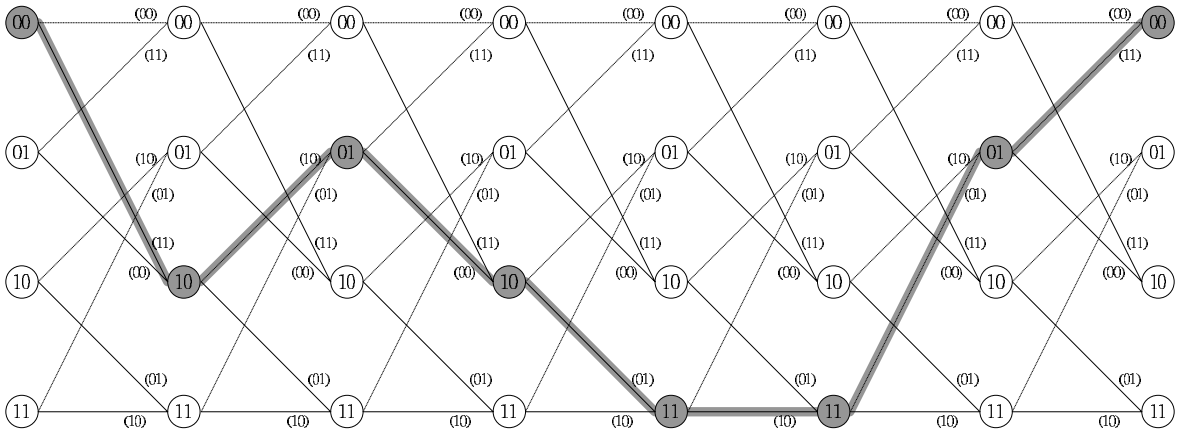


Figure 2.4: The trellis diagram

convolutional code in the trellis diagram, where the l denotes the length of input sequence.

2.2 Viterbi Algorithm

The Viterbi algorithm developed in 1967 [1] has been considered the optimal solution for decoding convolutional codes. The convolutional encoding process can be represented by trellis diagram where each node corresponds to a distinct state at a given time, and each branch is a transition between two states of different time instances. Among all possible paths in trellis diagram, a optimum solution to decode a convolutional code is equivalent to find the maximum likelihood path. Conclusively, the Viterbi decoding algorithm searches for the maximum likelihood state transition sequence according to the observed data in the noisy channel. Before launching the Viterbi algorithm, there are some basic should be introduced.

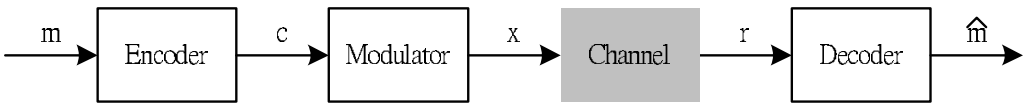


Figure 2.5: The system block of the encoder and decoder

The system block of the encoder and decoder is shown in Figure 2.5. The idea of the encoder is to transform the message sequence "m" into the codeword sequence "c" by adding certain structural redundancy. This redundancy is designed to overcome the channel impairments. Inversely, a concept of a decoder is to find the maximum likelihood sequence "$\hat{m}$" according to the structural redundancy. Mathematically, to find the maximum likelihood sequence "$\hat{m}$" is to maximize the probability $P(m|r)$, where m denotes the message sequence and r denotes the received sequence.

Using Bayes' rule

$$P(m|r) = \frac{P(m) \cdot P(r|m)}{P(r)} \tag{2.12}$$

where $P(r)$ is independent of m. Therefore, the decoder now is equivalent to maximize the probability $P(r|m)$. The probability $P(r|m)$ for the received sequence of length τ can be expressed as

$$\begin{aligned} P(r|m) &= P(r_{1\to\tau}|m_{1\to\tau}) \\ &= P(r_{1\to\tau}|x_{1\to\tau}) \\ &= \prod_{t=1}^{\tau} P(r_t|x_t) \\ &= \prod_{t=1}^{\tau}\prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r_{t,i}-x_{t,i})^2}{2\sigma^2}} \end{aligned} \tag{2.13}$$

where x denotes the modulated sequence. For the computing convenience, the log-likelihood function is used and given by

$$\log P(r|m) = \sum_{t=1}^{\tau} \log P(r_t|x_t) \tag{2.14}$$

For the AWGN channel, the log-likelihood function becomes

$$\begin{aligned} \log P(r|m) &= \sum_{t=1}^{\tau} \log \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r_{t,i}-x_{t,i})^2}{2\sigma^2}} \\ &= -\frac{n\tau}{2}\log(2\pi) - n\tau \log\sigma - \sum_{t=1}^{\tau}\sum_{i=0}^{n-1} \frac{(r_{t,i}-x_{t,i})^2}{2\sigma^2} \end{aligned} \tag{2.15}$$

This equation shows that to maximize $\log P(r|m)$ is equivalent to minimize Euclidean distance

$$\sum_{t=1}^{\tau}\sum_{i=0}^{n-1} \frac{(r_{t,i}-x_{t,i})^2}{2\sigma^2} \tag{2.16}$$

According to these computing processes, Viterbi proposed a algorithm to compute the minimum Euclidean distance as time goes on. There are two basic measure defined by the Viterbi, which are branch metric BM and path metric PM. At each time t, the branch metric and path metric is computing

$$\begin{aligned} BM(\beta_y^{t-1}) &= \sum_{i=0}^{n-1} \frac{(r_{t,i} - x_{t,i})^2}{2\sigma^2}|_{y=1,2} \\ PM(S_d^t) &= \min_{y=1,2}[PM(S_y^{t-1}) + BM(\beta_y^{t-1})] \end{aligned} \tag{2.17}$$

where S_d^t denotes the state d at time instance t, and β^t represents the branch between t and $t+1$. An equivalent radix-2 trellis diagram is shown in Figure 2.6. It is clear that the path metric $PM(S_d^t)$ is the minimum Euclidean distance for state S_d at time t. So the Viterbi algorithm can find the minimum path metric dynamically at each time instant. Then the maximum likelihood sequence can be estimated in trellis diagram along the minimum path metric.

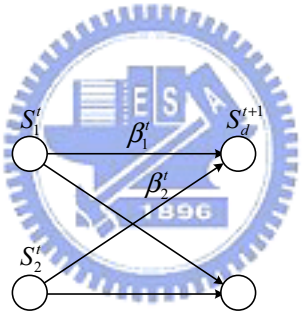


Figure 2.6: The radix-2 PM updating

The steps of the Viterbi algorithm are summarized as following.

- Initial or normalize the memory devices.
- According to the received sequence r, calculate the branch metric $BM(\beta_y^{t-1})$ from the previous state S_y to the current state S_d.
- Compute the transition $PM(S_y^{t-1}) + BM(\beta_y^{t-1})$, which is merged into the state S_d.

- Update the path metric $PM(S_d^t)$

$$PM(S_d^t) = \min_{y=1,2}[PM(S_y^{t-1}) + BM(\beta_y^{t-1})]$$

and store the survivor at the same time. The survivor here is the decision bit at time t.

- Use the survivor to decode the message sequence $\hat{m}$.

- Repeat this process.

2.3 Viterbi Decoder Architecture

The block diagram of the Viterbi algorithm is shown in Figure 2.7. There are four fundamental blocks in the Viterbi decoder. They are summarized as following

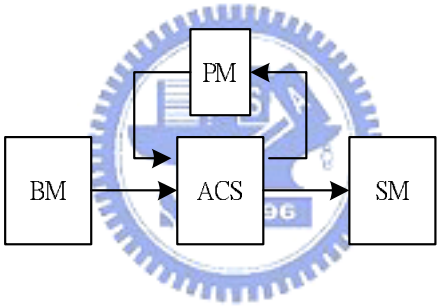


Figure 2.7: fundamental blocks of Viterbi decoder

- Branch Metric Unit (BM):
  According to received sequence r, compute the value $BM(\beta_y^{t-1})$ for different branches in trellis diagram.

- Add-Compare-Select Unit (ACS):
  Calculate the value $PM(S_d^t) = \min_{y=1,2}[PM(S_y^{t-1}) + BM(\beta_y^{t-1})]$ for each state. Output the value of $PM(S_d^t)$ and generate the survivor sequence.

- Path Metric Unit (PM):

  Store the accumulative path metric $PM(S_d^t)$ at each iteration. Detect the overflow of the accumulative path metric and normalize it.

- Survivor Path Unit (SM):

  Update the survivor from Add-Compare-Select unit. Then, using the register-exchange (RE) [15] algorithm or trace-back (TB) algorithm [16] to decode the maximum likelihood sequence.

2.3.1 Branch Metric Unit

Each time a new data is received by the decoder, the branch metric unit computes the value $BM(\beta_y^{t-1})$. Because the value of the branch metric is proportional to the logarithm of probability $\log P(m|r)$. It is clear that the numbers of quantization levels dominate the performance of Viterbi decoder. In the simplest design, the 2 levels quantization is used. However, the higher levels quantization the more coding gain can be obtained. If the quantization levels less than 2 levels, it is called the hard-decision decoding. Otherwise, it is called the soft-decision decoding.

In the hard-decision decoding, when a signal is received, a binary decision is made to determine whether the signal represents a transmitted zero or one. Therefore, the Hamming distance is used to simplify the calculation of branch metric. Using Hamming distance, the value of branch metric is described as

$$BM(\beta_y^{t-1}) = \sum_{i=1}^{n} (r_i \oplus c_i) \tag{2.18}$$

where $\oplus$ denotes the XOR operation.

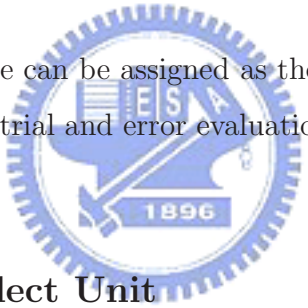
The soft-decision decoding process can provide an increase in coding gain about 2 to 3 dB over hard-decision decoding on the AWGN channel. Here, the uniform metric assignment method is introduced. If 8 levels quantization is used for a rate 1/2 code, then the entire set of branch metric are represented in Table 2.1. In this table, each pair (d_1, d_2)

denotes the Euclidean distance between codeword symbols and received sequences, and the coordinate represents the value of the branch metric.

Table 2.1: Uniform branch metric assignments

d_1d_2	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	8
2	2	3	4	5	6	7	8	9
3	3	4	5	6	7	8	9	10
4	4	5	6	7	8	9	10	11
5	5	6	7	8	9	10	11	12
6	6	7	8	9	10	11	12	13
7	7	8	9	10	11	12	13	14

In many applications, the table can be assigned as the non-uniform one. It may have the better assignments through a trial and error evaluation according to the situations of the channel.



2.3.2 Add-Compare-Select Unit

Typically, PM calculation with the recursive add-compare-select (ACS) operation is the most timing critical part that dominates the overall throughput. The path metric of state S_d^{t+1} at time instance $t+1$ can be recursively obtained by

$$\begin{aligned} PM(S_d^{t+1}) &= \min_{x=1,2}[PM(S_x^t) + BM(\beta_x^t)] \\ PM(S_x^t) &= \min_{y=1,2}[PM(S_{x,y}^{t-1}) + BM(\beta_{x,y}^{t-1})] \\ PM(S_{x,y}^{t-1}) &= \min_{z=1,2}[PM(S_{x,y,z}^{t-2}) + BM(\beta_{x,y,z}^{t-2})] \\ &\vdots \end{aligned} \tag{2.19}$$

Note that S_x^t connects to S_d^{t+1} through β_x^t, $S_{x,y}^{t-1}$ connects to S_x^t through $\beta_{x,y}^{t-1}$, and $S_{x,y,z}^{t-2}$ attaches to $S_{x,y}^{t-1}$ via $\beta_{x,y,z}^{t-2}$. The recursion in (2.19) is an ACS operation shown in Fig. 2.8

that iteratively updates the path metrics in each time instance. It is the serial operations within ACSU that causes the critical path bottleneck.

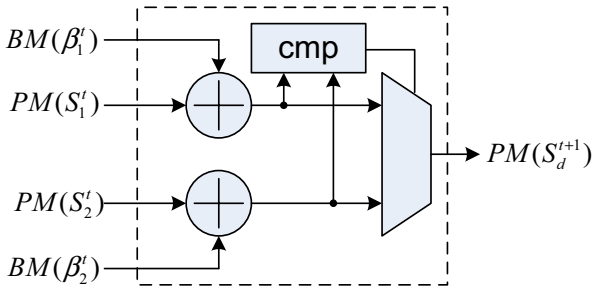


Figure 2.8: The radix-2 ACS structure

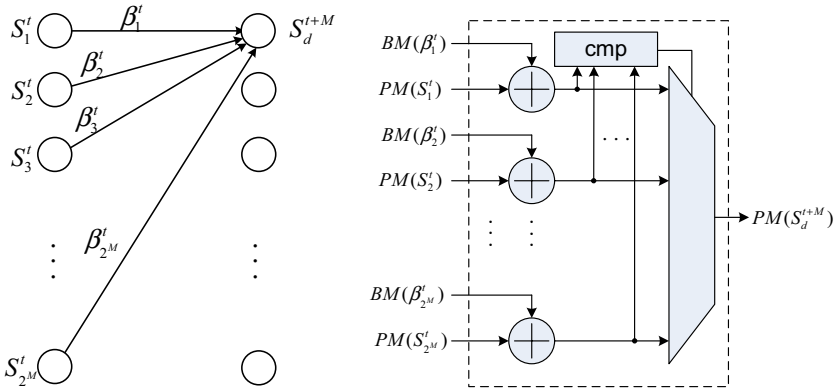
Therefore, assume $R = 1/n$ binary codes, the radix-2^M approach is proposed to enhance the decoding speed by a factor of M [9]. With M-step lookahead architecture, the trellis structure becomes radix-2^M in Fig. 2.9(a), and the $PM(S_d^{t+M})$ at time instance $t + M$ can be expressed by

$$PM(S_d^{t+M}) = \min_{x \in C}[PM(S_x^t) + BM(\beta_x^t)], \tag{2.20}$$

and $C = \{1, 2, ..., 2^M\}$ is the set of indexes indicating S_x^t connects to S_d^{t+M} through β_x^t. The equivalent radix-2^M ACS unit in Figure 2.9(b) achieves M times speedup as compared to the radix-2 ACSU in Figure 2.8. Nevertheless, the number of branches in Figure 2.9(a) will be 2^{M-1} times of that in radix-2 trellis, leading to the exponentially increasing complexity and limited M value (≤ 2). Hence, high-radix approach that accelerates Viterbi decoding may also cause large critical path delay. As shown in Figure 2.9, the adders can be proceeded simultaneously, but the speed of the comparator will be degraded as the number of branches increases. Therefore, the comparator should be optimized to acquire the corresponding enhancement contributed by high-radix trellis.

2.3.3 Survivor Path Unit

There are two well-known methods for the storage of the survivor sequences. One is the register-exchange method, and another is the trace-back method. The two methods would



(a) Radix-2^M trellis (b) Radix-2^M ACSU

Figure 2.9: The radix-2^M PM updating and the corresponding ACS structure

be introduced in the following.

- Register-exchange Method

  The register-exchange method stores the decoded sequences in shift register array. At each iteration, the decoded sequences are shifted according to the decision result of the survivor. In order to explain the operation of the register-exchange method, the convolutional code with generator polynomial $g_1(D) = 1 + D + D^2$ and $g_2(D) = 1 + D^2$ is used. And the hard-decision decoding shown in Figure 2.10 is adopted to simplify the interpretation.

  If the message sequence is represented as

  $$(1, 0, 1, 1, 1, 0, 0)$$

  From the introduction of the chapter 2, the codeword sequence would be

  $$(11, 10, 00, 01, 10, 01, 11)$$

  Figure 2.11 illustrates the procedure to find the minimum distance path. When the codeword $R = 11$ is received, the branch metrics can be obtained from the Figure

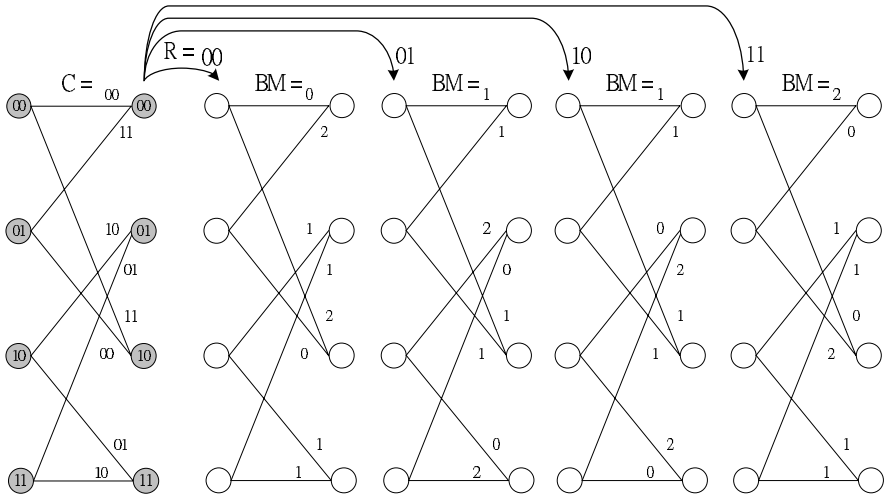


Figure 2.10: Branch metric in hard-decision decoding

2.10. Thus the transition on the dotted line from state S_{00} to state S_{00} is

$$\begin{aligned} PM_{00}(S_{00}^1) &= PM(S_{00}^0) + BM(\beta_{00}^0) \\ &= 0 + 2 \\ &= 2 \end{aligned}$$

and on the solid line from state S_{01} to state S_{00} is

$$\begin{aligned} PM_{01}(S_{00}^1) &= PM(S_{01}^0) + BM(\beta_{01}^0) \\ &= 0 + 2 \\ &= 0 \end{aligned}$$

Then the path metric at state S_{00} is

$$\begin{aligned} PM(S_{00}^1) &= \min[PM_{00}(S_{00}^0) + PM_{01}(S_{00}^0)] \\ &= \min(2, 0) \\ &= 0 \end{aligned}$$

Because the path metric on the dotted line is larger than that on the solid line, the dotted line should be deleted. On this rule, the survivor at each state would be obtained. The result is shown in Figure 2.12. At the first stage, each symbol to state S_{00} or state S_{01} would be decoded as "0", and to state S_{10} or S_{11} is "1". Therefore, the fixed value is shifted in the first stage.

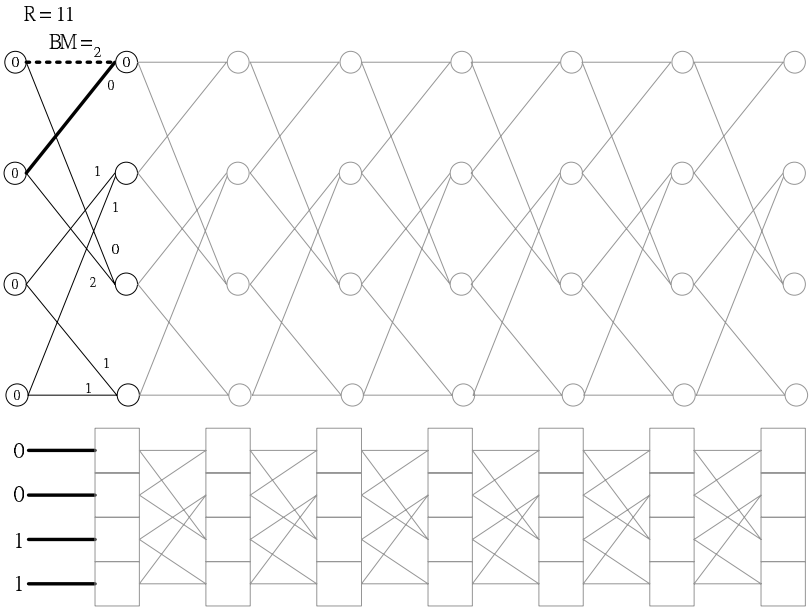


Figure 2.11: Minimum distance path

At the second iteration, the codeword $R = 10$ is received. The operations as shown in Figure 2.13 are similar to that at the first iteration, but the decoded symbols are shifted according to the decision result of the survivor.

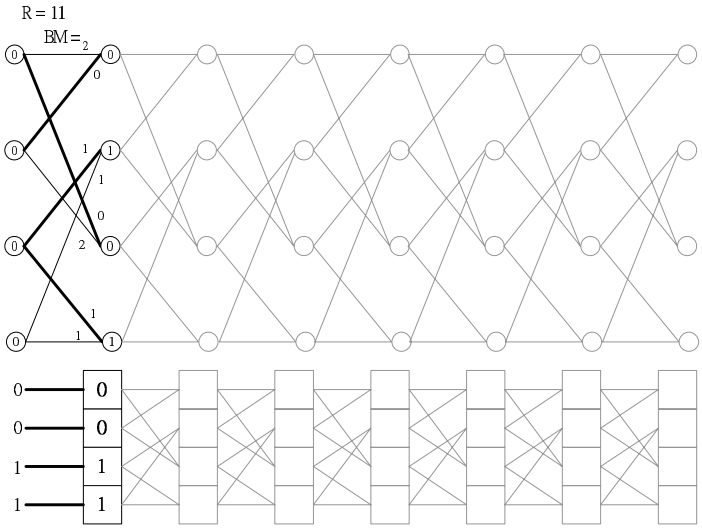


Figure 2.12: Register-exchange decoding at the first iteration

After the seventh iteration, the result is shown in Figure 2.14. It is clear that the minimum path metric is appeared at state S_{00}. Therefore, the decoded sequence is $(1, 0, 1, 1, 1, 0, 0)$.

The overall architecture of the register-exchange method can be implemented as in Figure 2.15. When the generator polynomials become complicated, the computing complexity grows up quickly. It means that high power consumption is needed. Thus the register exchange is not suitable for the decoders with the complicated generator polynomials.

- Trace-back Method

  The trace-back method stores the survivor sequence in the memory devices. In the $(2, 1, \nu)$ convolutional code, there are always two transitions merged into one state. If the upper transition arriving to this state is selected, the decision bit is set to zero. Otherwise, the decision bit is set to one. As shown in Figure 2.16, the decision bits $1, 0, 0, 0$ are set according to the lower transition to state S_{00}, and the upper transitions to other states.

  After the seventh iteration, all memory devices as shown in Figure 2.17 are filled.

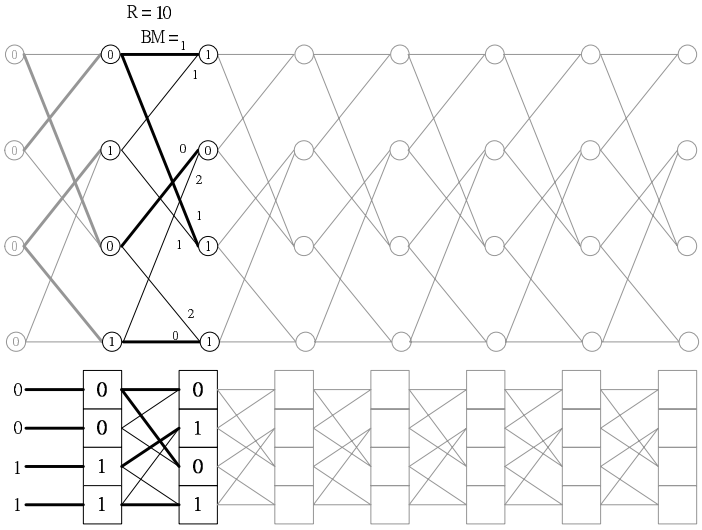


Figure 2.13: Register-exchange decoding at the second iteration

But, these decision bits define only the transitions rather than the decoded sequences. One more procedure as called trace-back must be performed. This operation starts at the state with the minimum path metric. In this example as shown in Figure 2.18, it starts from the state S_{00}, and traces backward from the seventh iteration to the first iteration. Therefore, a reverse order decoded sequence is generated.

A shift register base architecture is shown in Figure 2.19. The drawback in this architecture is that the longer critical path is performed. So it is hard to achieve the high data rate application.

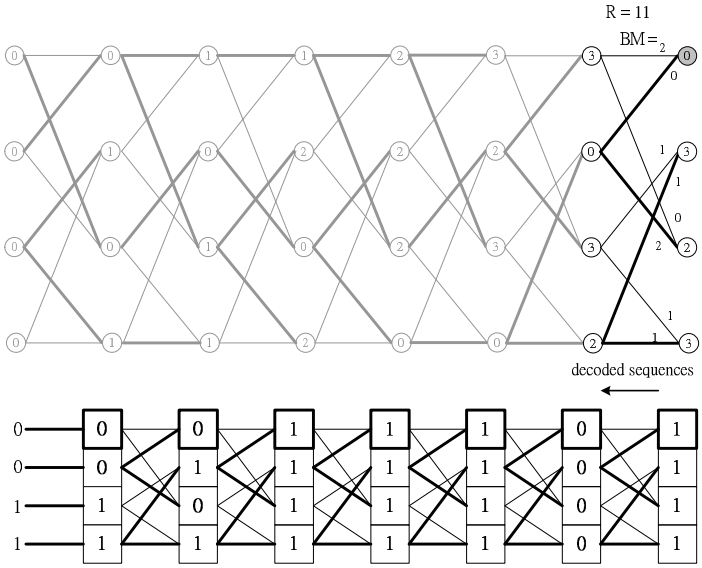


Figure 2.14: Register-exchange decoding at the seventh iteration

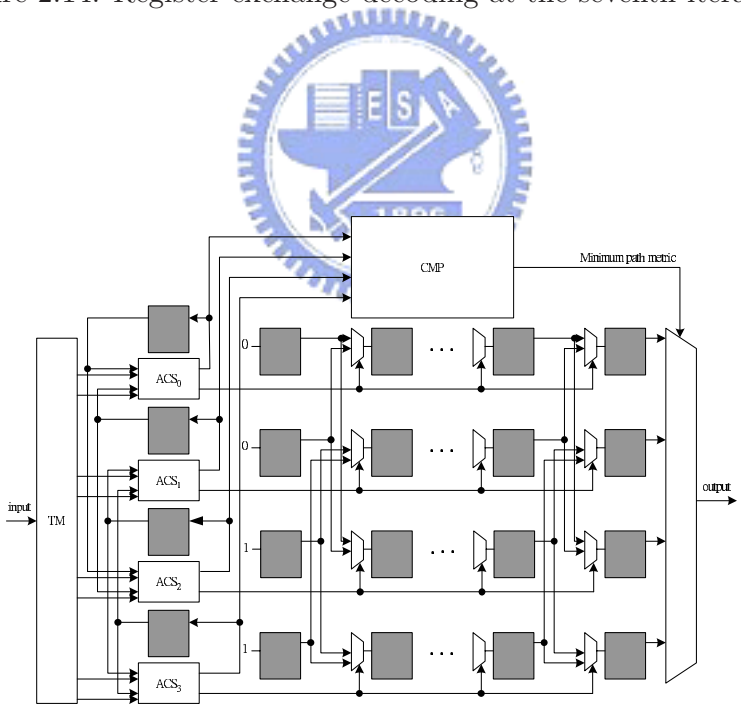


Figure 2.15: Block diagram of the register-exchange method

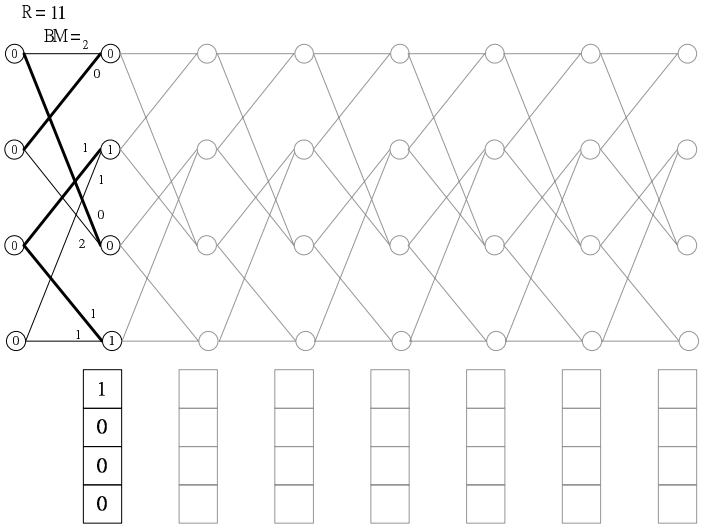


Figure 2.16: Trace-back decoding at the first iteration

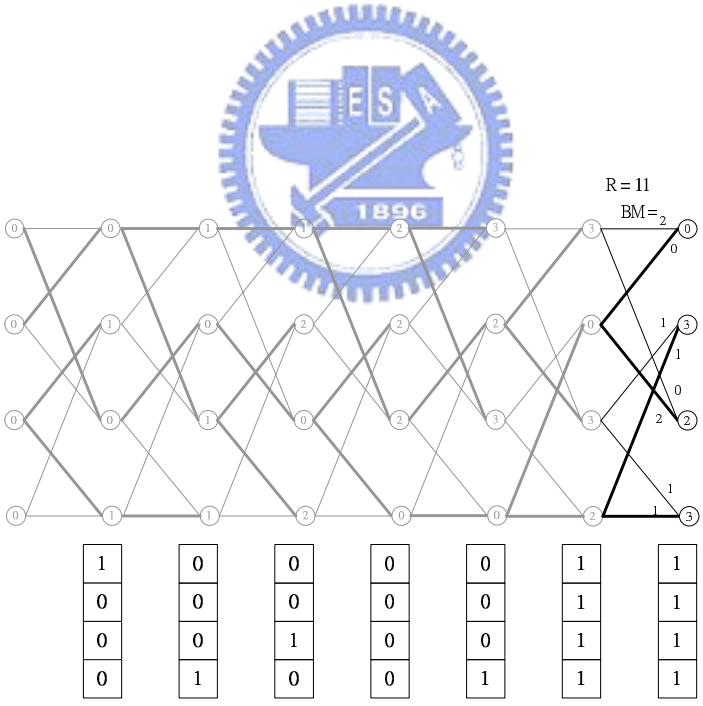


Figure 2.17: Trace-back decoding at the seventh iteration

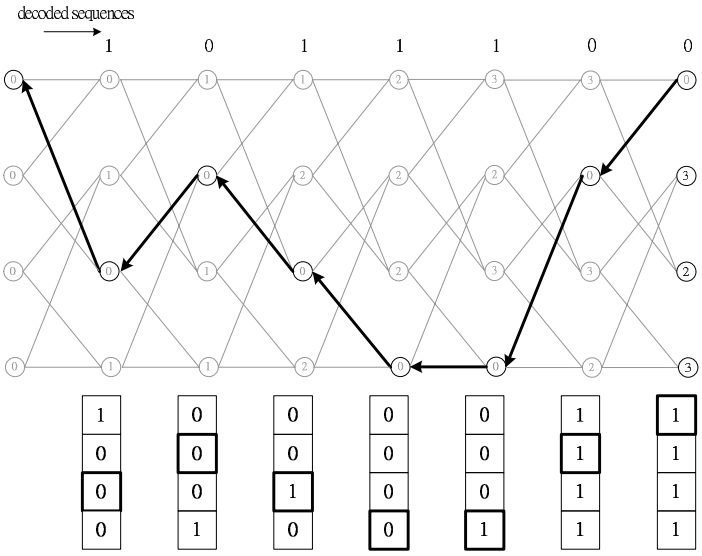


Figure 2.18: Trace-back procedure

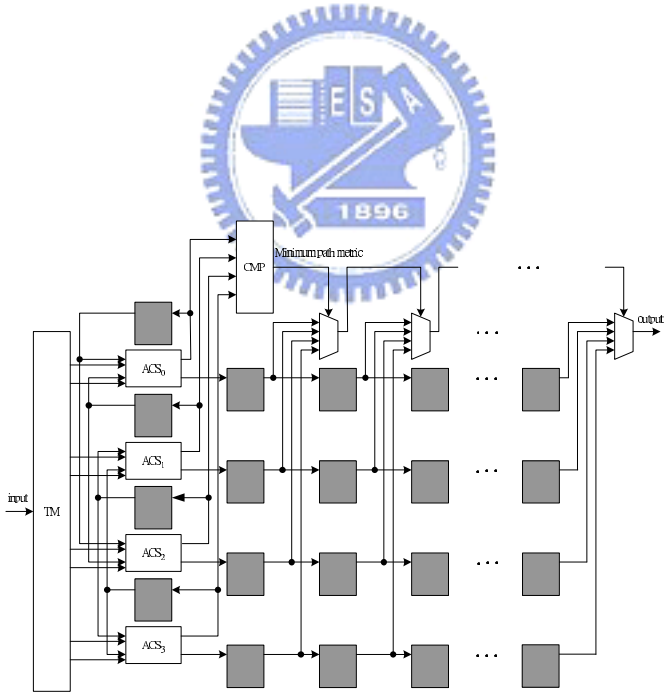


Figure 2.19: Block diagram of the trace-back method

Chapter 3

High-Speed ACS Unit Design

3.1 Introduction to Retiming Technique

The retiming approach tends to parallelize the ACSU computations. We first define another pre-path metric (pre-PM), denoted by Γ, as

$$\Gamma(\beta_x^t) \triangleq PM(S_x^t) + BM(\beta_x^t). \tag{3.1}$$

According to the recursion in (2.19), $PM(S_x^t)$ is a function of the information coming from the two branches, $\beta_{x,1}^{t-1}$ and $\beta_{x,2}^{t-1}$. Therefore, the computation in (3.1) can be extended to

$$\Gamma(\beta_x^t) \triangleq \min_{y=1,2}[\Gamma(\beta_{x,y}^{t-1})] + BM(\beta_x^t),\ \forall \beta_{x,y}^{t-1} \text{ connect to } S_x^t, \tag{3.2}$$

resulting in a recursion for Γ that contains the compare-select (CS) function for $\Gamma(\beta_{x,y}^{t-1})$ and the addition with $BM(\beta_x^t)$. Figure 3.1 illustrates the operation in trellis when $x = 1$. Note that the final addition has no impact on the compare function; therefore, the addition and the comparison can be performed concurrently, leading to the parallel architecture in Figure 3.2(b). Since the recursion has been changed from $PM(S_d^{t+1})$ to $\Gamma(\beta_1^t)$ and $\Gamma(\beta_2^t)$, the number of adders and multiplexers is doubled in contrast to the original ACSU in Figure 2.8.

The process is a retiming of PM registers and adders among different time instances, and Fig. 3.2(a) demonstrates the retiming procedure with pre-PMs instead of PMs being

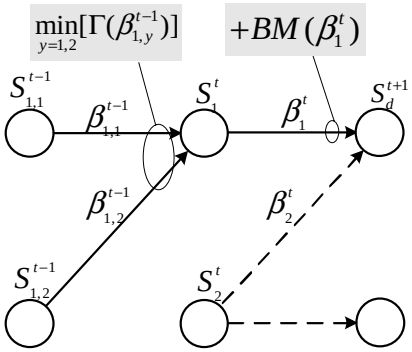
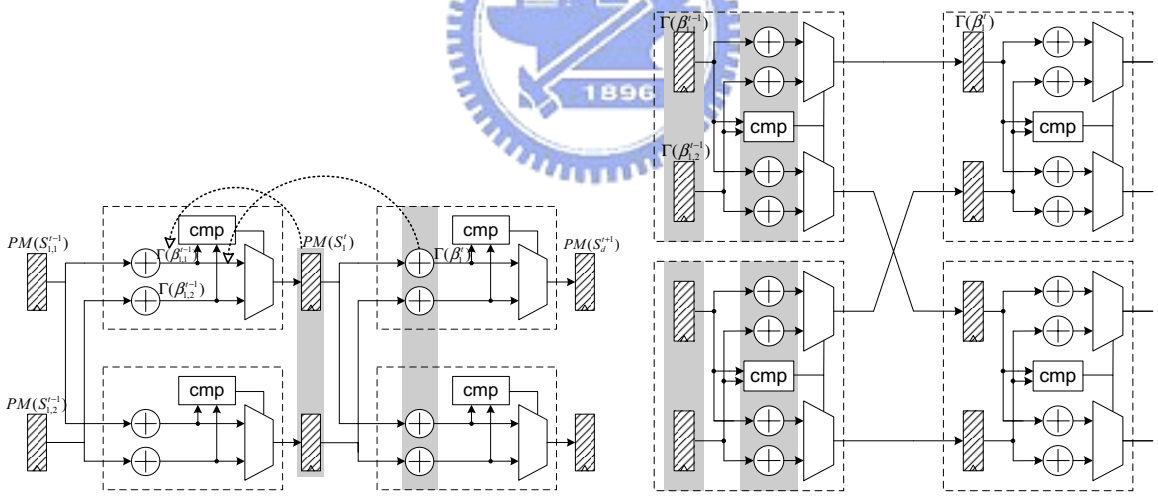


Figure 3.1: The trellis diagram after retiming

stored in registers. The PM registers at time instance t are moved to the branches between t and $t-1$ to keep the pre-PM metric Γ, making the number of registers twofold. Furthermore, the adders are also relocated to be parallel with the compare operations. The result after retiming is presented in Fig. 3.2(b) in which the number of registers, adders, and multiplexers is double as many as the structure before retiming. Actually, the architecture in Fig. 3.2(b) is identical to the double state approach presented in [17].



(a) Retiming of radix-2 ACSU (b) After retiming

Figure 3.2: The retiming procedure among different time instances

The performance of high-radix approaches is dominated by the large critical path due to exponentially increasing branches. Therefore, the present method improves the speed

through parallel processing in radix-2^M ACS units. The other area-efficient solution based on two-dimensional (2-D) radix-$2^m \times 2^n$ trellis structure is also proposed in following sections.

3.2 Retiming of radix-2^M

The critical part of a radix-2^M ACS unit is to search the minimum $PM + BM$ value among 2^M candidates. One of the solutions to simplify the searching algorithm is the decomposition of the candidates that need to be compared. The ACS operation in (2.20) can be re-written as

$$PM(S_d^{t+M}) = \min_{x \in \mathrm{X}_n} [\min_{y \in \mathrm{Y}_m} [PM(S_{x,y}^t) + BM(\beta_{x,y}^t)]], \tag{3.3}$$

where $\mathrm{X}_n = \{1, 2, \ldots, 2^n\}$, $\mathrm{Y}_m = \{1, 2, \ldots, 2^m\}$, and $M = m + n$. The minimum function is decomposed into two levels, and the 2^M candidates are partitioned into 2^n subsets. The first level is 2^m-way CS (CS-2^m) operations that finds the minimum within each subset containing 2^m candidates. Similarly, with a 2^n-way CS (CS-2^n) function, the outputs from the first level are compared consecutively to produce the final result. Fig. 3.3 demonstrates the architecture of a radix-2^M ACS unit. The critical path in Fig. 3.3 will be the adder plus two levels of comparator and multiplexer.

In order to further improve the speed, the retiming method as mentioned above is applied to the radix-2^M ACS unit. The variable $\Gamma(\beta_x^t)$ is defined as the result of the first level comparison (see Fig.3.3), and

$$\Gamma(\beta_x^t) = \min_{y \in \mathrm{Y}_m} [PM(S_{x,y}^t) + BM(\beta_{x,y}^t)] \tag{3.4}$$

Therefore, the original ACS recursion in (3.3) can then be converted to

$$PM(S_d^{t+M}) = \min_{x \in \mathrm{X}_n} [\Gamma(\beta_x^t)]. \tag{3.5}$$

If we substitute $\Gamma(\beta_{x,y,z}^{t-M})$ for $PM(S_{x,y}^t)$ in (3.4), the recursion of $\Gamma(\beta_x^t)$ can be deduced,

$$\Gamma(\beta_x^t) \quad = \quad \min_{y \in \mathrm{Y}_m} [\min_{z \in \mathrm{X}_n} [\Gamma(\beta_{x,y,z}^{t-M})] + BM(\beta_{x,y}^t)], \tag{3.6}$$

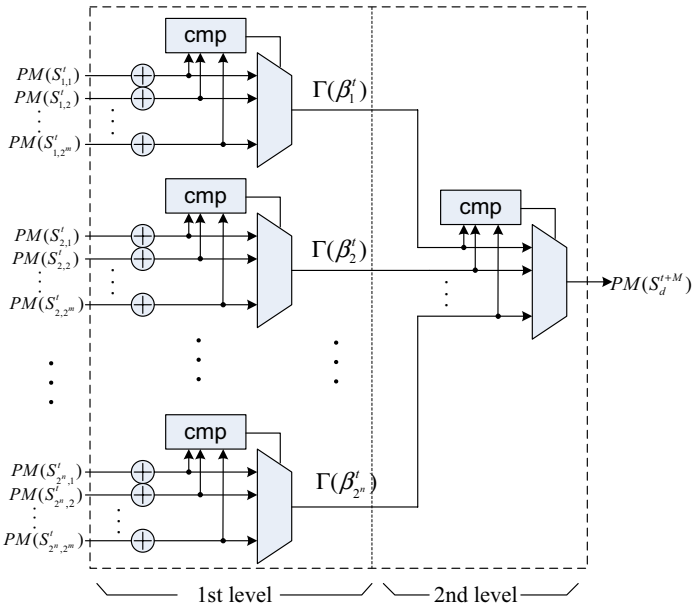


Figure 3.3: Radix-2^M ACS unit with two levels of CS functions

where $\beta^{t-M}_{x,y,z}$ is the z-th incoming branch of $S^t_{x,y}$.

Fig. 3.4 illustrates the related radix-2^M trellis diagram for (3.6). With two levels of computations, the first CS-2^n operation performs $\min_z[\Gamma(\beta^{t-M}_{x,y,z})]$ and the second ACS-2^m operation completes the remaining calculations in (3.6). Since $BM(\beta^t_{x,y})$ is constant for all z, the first CS-2^n operation and the additions in ACS-2^m can proceed simultaneously, achieving less datapath delay.

Fig. 3.5(a) shows the retiming process (RT-1) of radix-2^M ACS unit according to (3.6) and Fig. 3.4. The registers keeping $PM(S^t_{x,y})$ for $y \in \mathrm{Y}_m$ are moved to the branch $\beta_{x,y,z}$ to store $\Gamma(\beta^{t-M}_{x,y,z})$ for $z \in \mathrm{X}_n$; therefore, the number of registers becomes 2^n times. Furthermore, the adders are changed to the inputs of 2^n-to-1 multiplexer, and their amount also increases $2^n - 1$ times. The number of multiplexers in the second level operation should be 2^M times because each state has 2^M leaving branches. Fig. 3.5(b) shows the structure of the retimed radix-2^M ACS unit where the comparisons in the first level coincide with the additions.

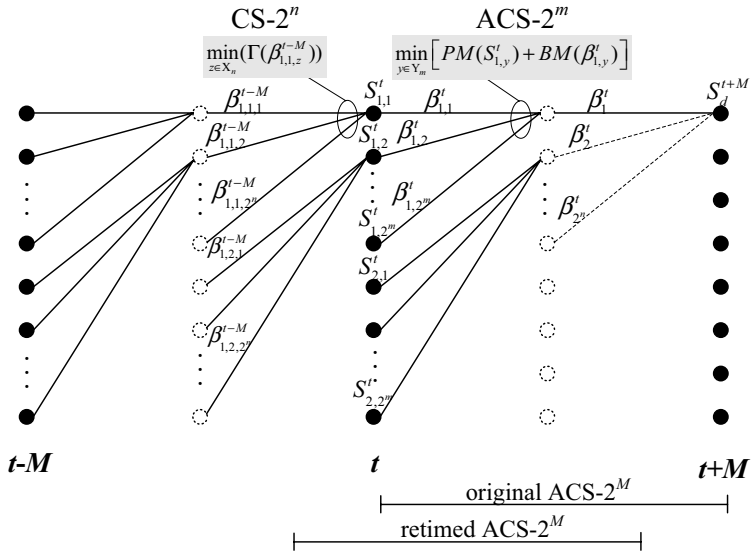


Figure 3.4: Radix-2^M trellis diagram

3.3 Introduction to Two-Dimensional ACS Unit

The exponentially increasing complexity of high-radix Viterbi decoder is the major concern during VLSI implementation. Moreover, the number of BMs generated by BMU also increases exponentially. For a code with $K \geq 7$, large number of ACS units required to achieve high-speed parallel processing will need more area for signal routing. Therefore, a radix-$2^m \times 2^n$ structure is introduced to achieve the throughput equivalent to radix-2^M approach where $M = m + n$. As shown in Fig. 3.6, the radix-$2^m \times 2^n$ ACS unit consists of two levels of consecutive radix-2^m and radix-2^n ACS units. In the second level, the new PMs at time instance $t + m$ is obtained in advance and directly passed to compute $PM(S_d^{t+M})$, leading to an equivalent radix-2^M ACS operation. Thus,

$$\begin{aligned} PM(S_d^{t+M}) &= \min_{x \in \mathrm{X}_n}[PM(S_x^{t+m}) + BM_2(\beta_x^{t+m})] \\ &= \min_{x \in \mathrm{X}_n}[\min_{y \in \mathrm{Y}_m}[PM(S_{x,y}^{t}) + BM_1(\beta_{x,y}^{t})] + BM_2(\beta_x^{t+m})] \end{aligned} \tag{3.7}$$

where BM_1 and BM_2 correspond to the BM value in the 1-st and the 2-nd level.

The radix-$2^m \times 2^n$ ACS unit in Fig. 3.7, referred to the two-dimensional (2-D) structure, is similar to the radix-2^M ACS unit, except that only smaller radix-2^m ACS (ACS-2^m)

units and radix-2^n ACS (ACS-2^n) units are required. Since the exponentially increasing hardware cost of a high-radix ACS, the complexity of a Viterbi decoder based on radix-$2^m \times 2^n$ architecture is much smaller than that based on radix-2^M architecture. However, the critical path of the 2-D structure is through two levels of ACS unit, inducing one more adder delay as compared with the radix-2^M ACS unit in Fig. 3.3.

3.4 Retiming of 2-D ACS Unit

The 2-D ACS unit can be further improved to achieve higher speed with acceptable cost through retiming approach. Moreover, the experimental results show that radix-$2^m \times 2^n$ structure with retiming method is more area efficient than the radix-2^M architecture. Two possible retiming schemes based on (3.7) will be presented in the following. The first scheme comes from the observation that $BM_2(\beta_x^{t+m})$ is independent of the function $\min_{y \in \mathrm{Y}_m}[PM(S_{x,y}^t) + BM_1(\beta_{x,y}^t)]$, and can be moved to the inputs of multiplexers in the first level of Fig. 3.7. This retiming procedure (RT-2) as shown in Fig. 3.9(a) results in a ACS unit in Fig. 3.9(b). Note that the critical path is almost the same as the radix-2^M ACS unit in Fig. 3.3. The overhead to attain this acceleration is $2^m - 1$ times more adders and multiplexers in the first level.

The other retiming scheme can be proceeded by setting $\Gamma(\beta_x^{t+m})$ to be

$$\Gamma(\beta_x^{t+m}) = \min_{y \in \mathrm{Y}_m}[PM(S_{x,y}^t) + BM_1(\beta_{x,y}^t)] + BM_2(\beta_x^{t+m}), \tag{3.8}$$

and (3.7) will become

$$PM(S_d^{t+M}) = \min_{x \in \mathrm{X}_n}[\Gamma(\beta_x^{t+m})]. \tag{3.9}$$

Similarly, $PM(S_{x,y}^t)$ can be extended to be a function of $\Gamma(\beta_{x,y,z}^{t-n})$ where $\beta_{x,y,z}^{t-n}$ is the incoming branch of state $S_{x,y}^t$. The extension of (3.8) should be

$$\Gamma(\beta_x^{t+m}) = \min_{y \in \mathrm{Y}_m}[\min_{z \in \mathrm{X}_n}[\Gamma(\beta_{x,y,z}^{t-n})] + BM_1(\beta_{x,y}^t)] + BM_2(\beta_x^{t+m}) \tag{3.10}$$

Fig. 3.8 illustrates the corresponding operation on the radix-$2^m \times 2^n$ trellis for (3.10)

with $x = 1$. The computation contains CS-2^n operations, ACS-2^m calculations, and a final addition (Add). Note that the additions can also be retimed for less datapath delay.

The retiming procedure (RT-3) is demonstrated in Fig. 3.9(a) where both registers and adders are moved. Fig. 3.9(c) shows the final result after retiming. Keeping the results $\Gamma(\beta_x^{t+m})$ on branches, the registers increase to 2^n times as many as the original one. Furthermore, the numbers of adders and multiplexers also become 2^n times due to the movement of additions.

3.5 Comparison

The optimization methods as mentioned above tend to break the critical path through parallelizing the serial add, compare, and select operations. Fig. 3.10 compares datapath delays of different ACS configurations. The delay times of CS-2^m, CS-2^n, ACS-2^m, and ACS-2^n functions are defined to be T_{CS-2^m}, T_{CS-2^n}, T_{ACS-2^m}, and T_{ACS-2^n}. We also assume that T_{CS-2^m} and T_{CS-2^n} are larger than the delay time of additions. The major enhancement is the elimination of datapath delay contributed by additions. Note that both ACS-2^M with RT-1 and ACS-$2^m \times 2^n$ with RT-3 can achieve the lowest delay time $T_{CS-2^m}+T_{CS-2^n}$ because of the parallel additions and comparisons. Furthermore, Fig. 3.10 also shows that ACS-$2^m \times 2^n$ with RT-2 can acquire a comparable performance to the ACS-2^M structure.

Table 3.1 summarizes the complexity of the ACS architectures presented in this paper. The cost of ACS-$2^m \times 2^n$ is smaller than that of ACS-2^M because $2^M \geq 2^m + 2^n$, and the minimum adder requirement of ACS-$2^m \times 2^n$ can be achieved when $m = \lceil \frac{M}{2} \rceil$ and $n = M - \lceil \frac{M}{2} \rceil$. Considering ACS-$2^M$ with RT1 and ACS-$2^m \times 2^n$ with RT-3, the number of adders in the former is larger than that in the latter while $n > 1$. Moreover, ACS-2^M with RT-1 has 2^n times as many 2^m-way comparators as ACS-$2^m \times 2^n$ with RT-3. The original ACS-2^M structure has the delay time similar to ACS-$2^m \times 2^n$ with RT-2, but has $(2^n - 1)$ times more 2^m-way comparators, which are considerably more complex than adders. According the the summary in Table 3.1, the 2-D ACS-$2^m \times 2^n$ structure can

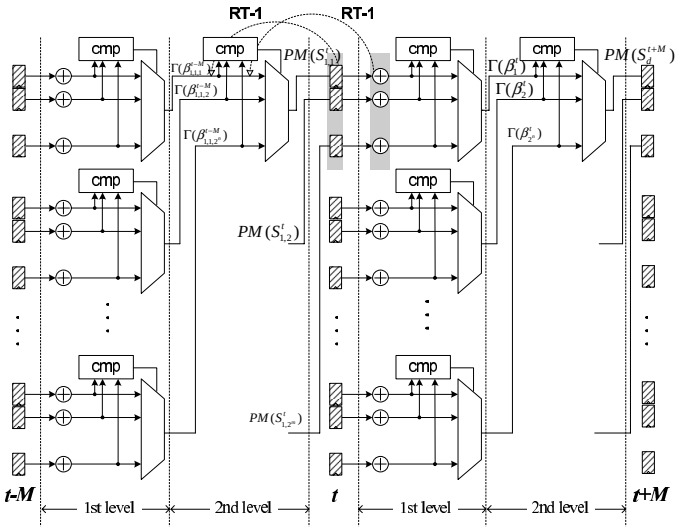
Table 3.1: Comparison of complexity among different ACS configurations

	registers	adders	2^m-way comparator	2^n-way comparator	2^m to 1 multiplexer	2^n to 1 multiplexer
ACS-2^M	N	$2^M \cdot N$	$2^n \cdot N$	N	$2^n \cdot N$	N
ACS-2^M (RT-1)	$2^n \cdot N$	$2^n \cdot 2^M \cdot N$	$2^n \cdot N$	N	$2^n \cdot N$	$2^M \cdot N$
ACS-$2^m \cdot 2^n$	N	$(2^m + 2^n) \cdot N$	N	N	N	N
ACS-$2^m \times 2^n$ (RT-2)	N	$(2^m + 2^M) \cdot N$	N	N	$2^n \cdot N$	N
ACS-$2^m \times 2^n$ (RT-3)	$2^n \cdot N$	$(2^M + 2^M) \cdot N$	N	N	$2^n \cdot N$	$2^m \cdot N$

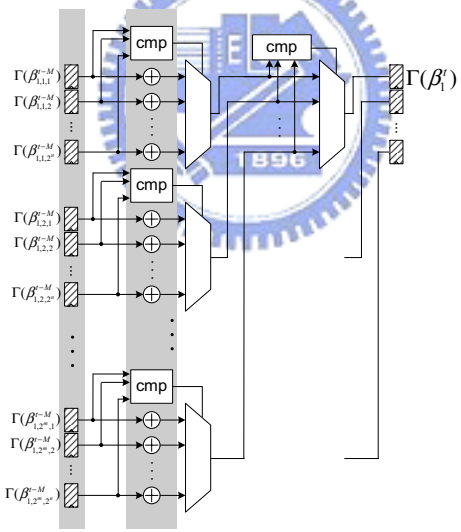
[1] The number of states is $N = 2^{K-1}$.

[2] $M = m + n$.

accomplish more cost efficient solutions with retiming for the high-speed requirement.



(a) Initial radix-2^M ACS structure



(b) Retimed radix-2^M ACS structure

Figure 3.5: Retiming of the radix-2^M ACS unit among different time instances

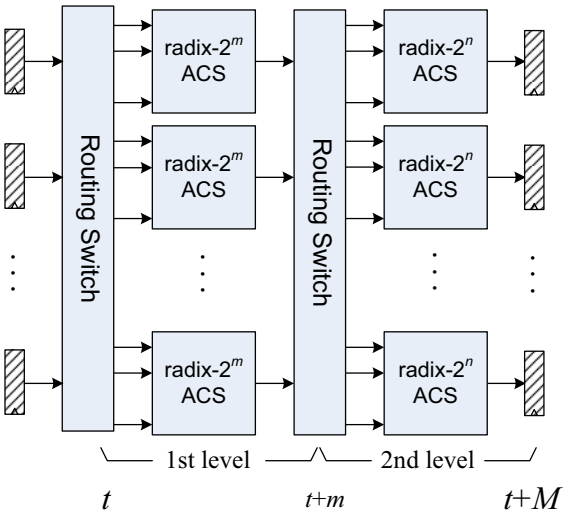


Figure 3.6: The structure of ACS-$2^m \times 2^n$ unit

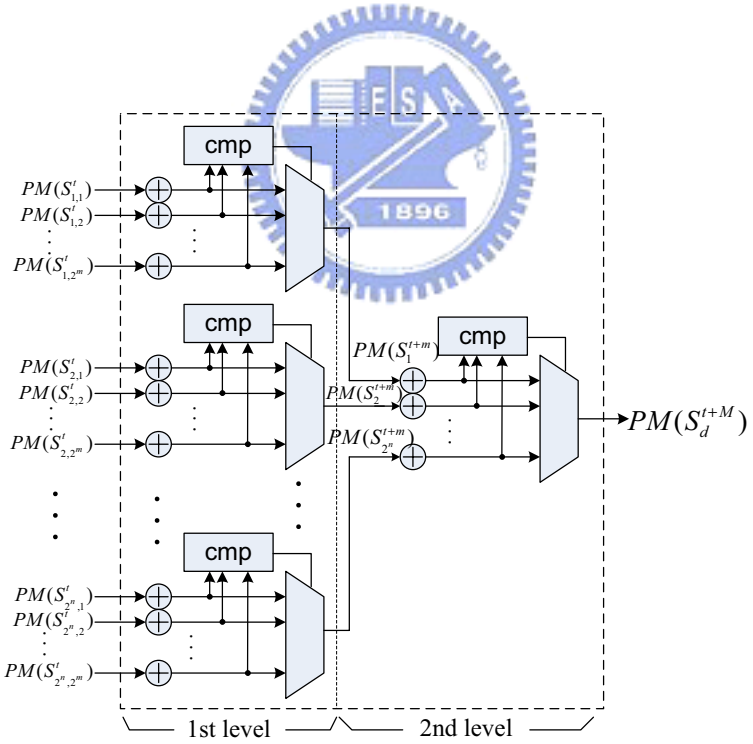


Figure 3.7: The ACS-$2^m \times 2^n$ unit

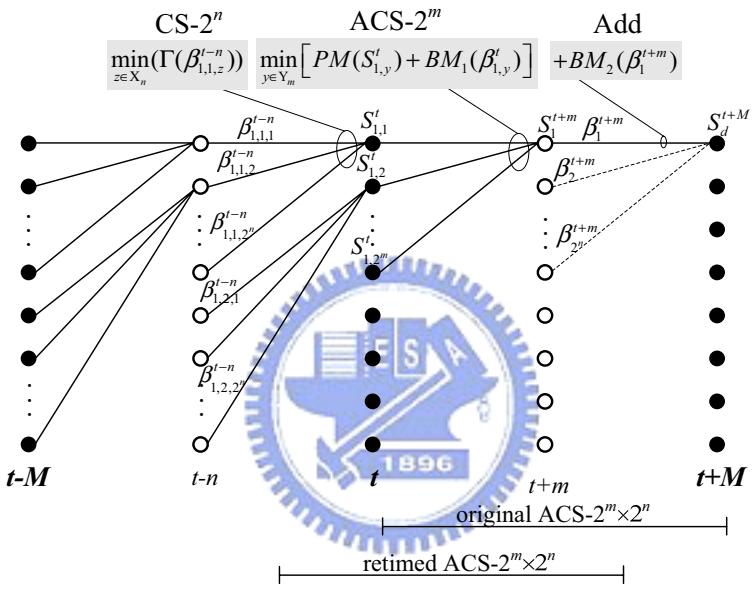
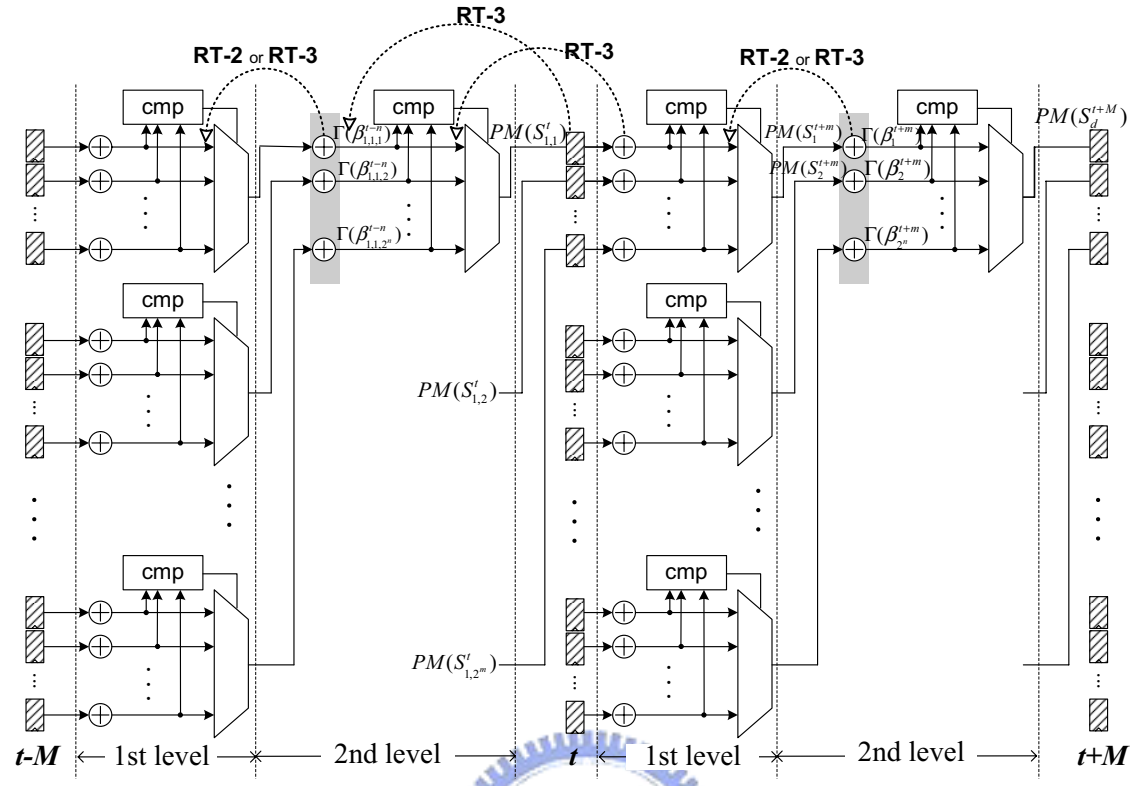
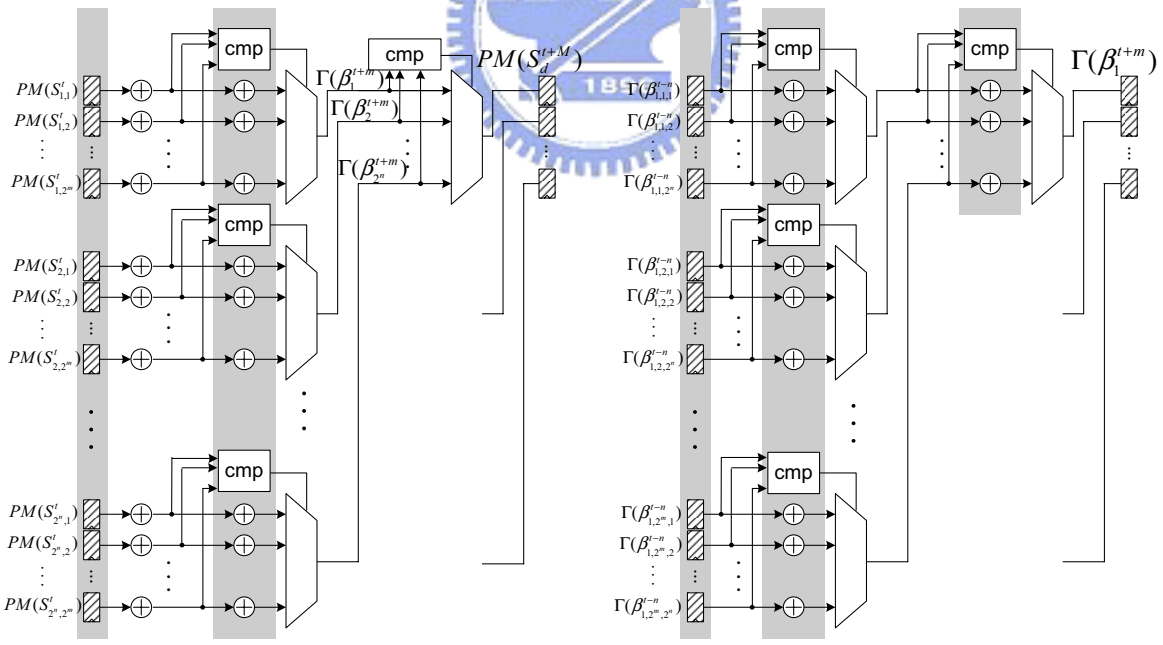


Figure 3.8: Radix-$2^m \times 2^n$ trellis



(a) Initial radix-$2^m \times 2^n$ ACS structure



(b) Result of RT-2

(c) Result of RT-3

Figure 3.9: Retiming of the radix-$2^m \times 2^n$ ACS unit

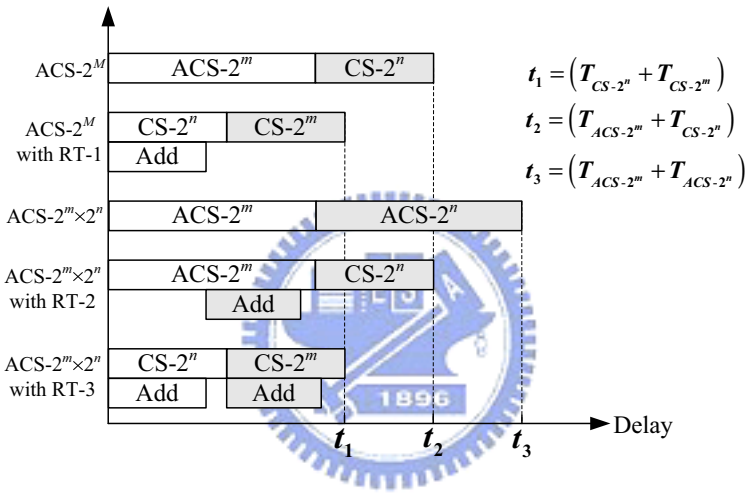


Figure 3.10: Comparison of critical path delay for original and retimed ACS units

Chapter 4

Simulation and Implementation

4.1 Introduction to Ultra-Wide Band System

Ultra-wideband (UWB) is an emerging wireless physical(PHY)-layer technologythat uses a very large bandwidth [18, 19]. By its rule-making proposal in 2002, the Federal Communications Commission (FCC) unleashed 3.1GHz to 10.6GHz RF band for increasing high-speed data transmission. The mulit-band OFDM PHY-layer proposal indicates the coded OFDM (COFDM)-based baseband solution can provide up to 480Mb/s with 2m desired range for 528MHz UWB systems [4]. To enhance overall system performance, the 64-state convolutional codes and interleaving techniques are used in the forward error correction (FEC) mechanism, whose block diagram is shown in Fig. 4.1.

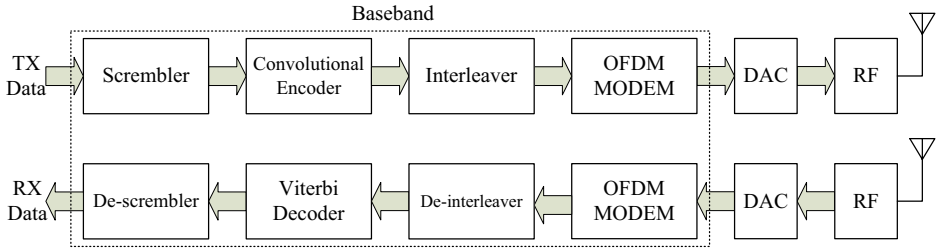


Figure 4.1: Block diagram of multi-band COFDM UWB systems.

In the MB-OFDM UWB systems, the maximum 480Mb/s data rate with a bandwidth of 528MHz is specified. The punctured convolutional code with either frequency or time

domain spreading is used to change the data rate for different channel state information. The encoding function uses the punctured convolutional encoder with the base rate (R) 1/3 and the generator polynomials $g_0 = 133_8$, $g_1 = 165_8$, and $g_2 = 171_8$ as shown in Figure4.2. Higher coding rates are derived by puncturing. Puncturing is a procedure for omitting some of the encoded bits in the transmitter and inserting a dummy zero metric into the decoder on the receive side in place of the omitted bits. The puncturing patterns are illustrated in Figure4.3.

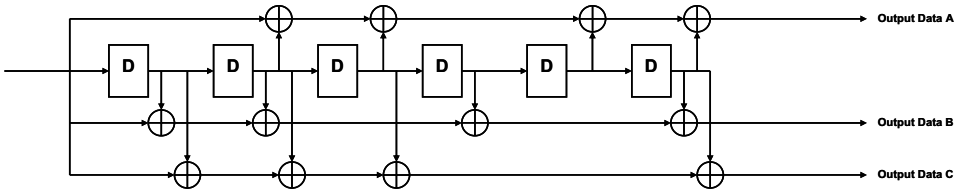


Figure 4.2: Convolutional encoder of multi-band COFDM UWB systems.

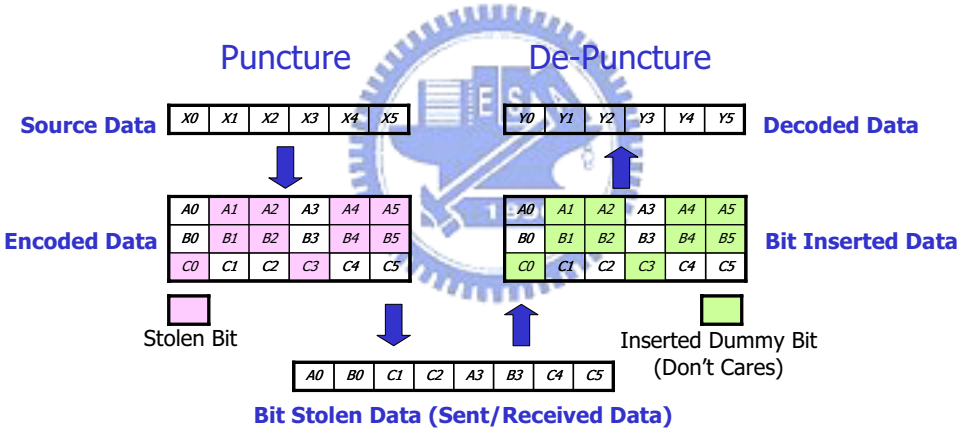


Figure 4.3: Punctured coding with rate $R = 3/4$

4.2 Simulation Results

The Viterbi decoder was designed to target the MB-OFDM physical layer proposal for the IEEE 802.15.3a standard. In order to determine appropriate design parameters such as the bit widths of the path metric, branch metric, and the input symbol, the performance

evaluation through simulations are necessary. The bit error rate (BER) curves of the floating point and the fixed point decoders are presented in Figure4.4. Note that the $R = 11/32$ case is excluded because its performance is very close to the $R = 1/3$ one.

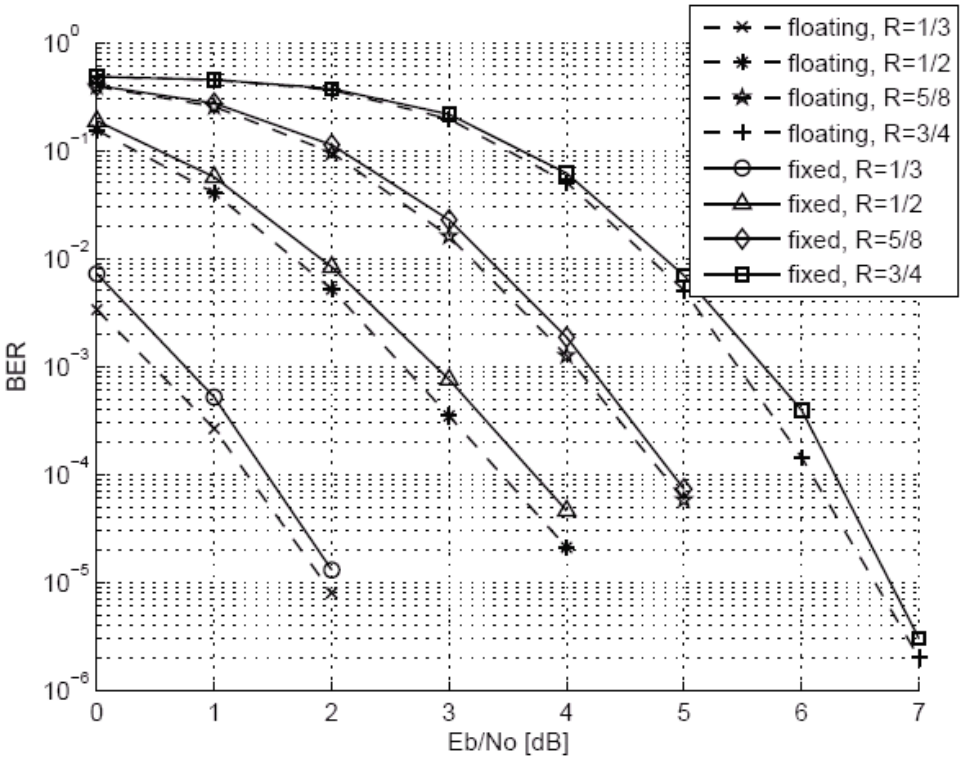


Figure 4.4: The BER curves

Note that the truncation length of the survivor memory varies with the rate R in order to reduce the power dissipation under limited performance loss. The input symbols are quantized to eight levels with the step size $\triangle = 0.25$. For both I and Q inputs, the range of $-1 \sim +1$ is divided into 8 parts corresponding to $0 \sim 7$.

4.3 Implementation Results

The circuit implementation of Viterbi decoders are completed based on the proposed high-radix and 2-D ACS structures. The punctured convolutional code specified in [4] is selected for implementation. Furthermore, the resolutions of path metric and branch

Table 4.1: Parameters of the Viterbi decoder

State number	64
Coding rate	$\frac{1}{3}, \frac{11}{32}, \frac{1}{2}, \frac{5}{8}, \frac{3}{4}$
PM width	9 bits
BM width	5 bits
Truncation length	96 (max.)
ACS structure	radix-16 $m = n = 2$

metric as well as the traceback length of SMU are determined according to the system of [4]. The parameters of the Viterbi decoder in our experiment are listed in Table 4.1.

Considering the high throughput requirement, the RE approach is applied to SMU with quite different structures in radix-2^M and radix-$2^m \times 2^n$ designs. The number of branches between time instances t and $t+M$ in radix-2^M trellis is $2^M \cdot N$; however, this number reduces to $(2^m + 2^n) \cdot N$ in radix-$2^m \times 2^n$ trellis, resulting in less multiplexers in SMU. In this implementation with $M = 4$ and $m = n = 2$, if we select the 4-to-1 multiplexer as the basic unit and assume the 16-to-1 multiplexer consists of five units, the multiplexers in the radix-2^4 SMU is $\frac{5 \cdot N}{(1+1) \cdot N} = 2.5$ times as many as that in the radix-$2^2 \times 2^2$ SMU.

Based on the proposed architectures, the Viterbi decoders have been implemented by using 1.8V 0.18-μm 1P6M CMOS technology and 1.2V 0.13-μm 1P8M CMOS technology. We estimate the data throughput with static timing analysis (STA) while considering 1.62V supply for the 0.18-μm design, 1.08V supply for the 0.13-μm design, the worst speed corner, and the coupling noise due to crosstalk effect on signal wires. The results with tight timing constraints are reported in Table 4.2 and Table 4.3. The gate count(N_G) is calculated based on the extracted gate level netlist from the layout, and ΔN_G indicates the gate count increase during the physical implementation. Note that ΔN_G can reflect the signal routing complexity in the layout implementation. Larger ΔN_G indicates more capacitance caused by signal connections should be buffered. The density is an area

Table 4.2: Implementation results with timing critical constraints

0.18-μm 1P6M	Data rate (Mb/s)	Area (mm^2)	N_G	ΔN_G	Density
ACS-16	513	9.30	740.0k	151.1k	0.79
ACS-16(RT-1)	623	15.21	1129.9k	190.4k	0.74
ACS-4 $\times$ 4[1]	427	4.41	310.4k	76.4k	0.77
ACS-4 $\times$ 4(RT-2)	553	5.29	398.6k	36.6k	0.75
ACS-4 $\times$ 4(RT-3)	731	6.76	533.1k	114.6k	0.79

[1] This chip was fabricated, and the results showed the 500Mb/s data rate is achieved under 1.8V supply.

utilization measure for standard cells within the core region and also dominated by the routing complexity.

4.4 Discussion

In Table 4.2, both ACS-16 and ACS-4$\times$4 with RT-2 can sufficiently satisfy the data rates of the UWB system in [4]. Nevertheless, the ACS-4 $\times$ 4 based decoder has only 57% area of the ACS-16 based one. With RT-1, the speed of the ACS-16 decoder can be risen from 513Mb/s to 623Mb/s. Furthermore, the data throughput can be improved by 71% and 30% when RT-3 and RT-2 are applied to the ACS-4 $\times$ 4 architectures.

The results in Table 4.3 indicate much higher speed and density can be accomplished due to the improvement of technology and two additional metal layers. Similar to the results of 0.18-μm designs, ACS-16 with RT-1 and ACS-4 $\times$ 4 with RT-3 are shown to achieve the highest data rates which are over 1Gb/s; however, the ACS-4$\times$4 based design is much smaller than the ACS-16 based one. The less computational units and the simple signal routing result in not only the smaller N_G and ΔN_G, but also the higher chip density. Consequently, the implementation shows that ACS-4 $\times$ 4 based decoders are with much small area.

Table 4.3: Implementation results with timing critical constraints

0.13-μm 1P8M	Data rate (Mb/s)	Area (mm^2)	N_G	ΔN_G	Density
ACS-16	933	3.61	647.3k	84.7k	0.92
ACS-16(RT-1)	1,038	5.36	945.7k	212.3k	0.90
ACS-4 $\times$ 4	923	1.28	239.2k	29.1k	0.96
ACS-4 $\times$ 4(RT-2)	986	1.85	349.9k	45.6k	0.97
ACS-4 $\times$ 4(RT-3)	1,105	1.96	358.0k	43.9k	0.94

Table 4.4: Implementation results of 500Mb/s data rate

	Area (mm^2)	N_G	ΔN_G	Density	Power(mW)[a]
ACS-16	2.66	491.7k	66.1k	0.94	344
ACS-16(RT-1)	3.84	685.5k	82.0k	0.92	533
ACS-4 $\times$ 4	0.90	165.5k	5.2k	0.94	119
ACS-4 $\times$ 4(RT-2)	1.38	247.7k	8.4k	0.94	169
ACS-4 $\times$ 4(RT-3)	1.44	263.2k	9.9k	0.94	195

[a] 1.2V supply and 500Mb/s data rate

Table 4.4 also lists the results when the timing constraint of 500Mb/s throughput is applied to all designs. The reports of ACS-4 $\times$ 4 based Viterbi decoders also present much smaller area and ΔN_G than the ACS-16 based decoders. In this table, the power consumption evaluated with 1.2V power supply reveals the same trends as N_G.

In conclusion, these reports confirm the analysis in section 3.5 that ACS-4 $\times$ 4 based architectures are more cost efficient for high-speed Viterbi decoders. Furthermore, the retiming techniques can improve the throughput especially for the timing critical cases. In the 0.13-μm technology, the ACS-4 $\times$ 4 based design can completely meet the UWB system in [4], and over 1Gb/s data rates are available with the retiming process.

Chapter 5

Conclusion

The 2-D ACS structure and the retiming mechanism are presented in this paper. The 2-D architecture provides more area efficient solutions for the high-radix trellis decoding, and the retiming techniques reduce the critical path delay of ACS units to facilitate high-speed applications. The Viterbi decoder for the UWB system [4] is also realized with the proposed approaches. The experimental results report a significant area reduction for the designs with 2-D ACS unit and a considerable improvement in throughput when the retiming process is employed. The 0.18-μm chip design shows RT-3 can improve the speed of ACS-4 $\times$ 4 by about 71%. In addition, built in the 0.13-μm technology, both the ACS-16 decoder and the ACS-4 $\times$ 4 decoder with retiming can accomplish the 1Gb/s data rate, but the later occupies only 37% area in contrast to the former.

Bibliography

[1] A. J. Viterbi, "Error bounds for convolutional codes and asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, no. 2, pp. 260–269, Apr. 1967.

[2] J. Omura, "On the viterbi decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 177–179, Jan. 1969.

[3] J. G. D. Forney, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, no. 3, pp. 268–278, Mar. 1973.

[4] A. Batra *et al.*, "Multi-band OFDM physical layer proposal for IEEE 802.15 task group 3a," submitted to IEEE P802.15 working group for WPANs, Sept. 2004.

[5] R. Fisher *et al.*, "DS-UWB physical layer submission to 802.15 task group 3a," submitted to IEEE P802.15 working group for WPANs, Mar. 2004.

[6] J. Tang and K. K. Parhi, "Viterbi decoder for high-speed ultra-wideband communication systems," in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, 2005, pp. 37–40.

[7] M. Anders, S. Mathew, R. Krishnamurthy, and S. Borkar, "A 64-state 2GHz 500Mbps 40mW Viterbi accelerator in 90nm CMOS," in *Symop. VLSI Circuits Dig. Tech. Papers*, 2004, pp. 174–175.

[8] S. W. Choi and S. S. Choi, "200Mbps Viterbi decoder for UWB," in *Int. Conf. Advanced Commun. Tech.*, vol. 2, 2005, pp. 904–907.

[9] P. J. Black and T. H. Meng, "A 140-Mb/s, 32-state, radix-4, Viterbi decoder," *IEEE J. Solid-State Circuits*, vol. 27, no. 12, pp. 1877–1885, Dec. 1992.

[10] H. Dawid, G. Fettweis, and H. Meyr, "A CMOS IC for Gb/s Viterbi decoding: system design and VLSI implementation," *IEEE Trans. VLSI Syst.*, vol. 4, no. 1, pp. 17–31, Mar. 1996.

[11] A. K. Yeung and J. M. Rabaey, "A 210Mb/s radix-4 bit-level pipelined Viterbi decoder," in *IEEE Int. Solid-State Circuit Conf. (ISSCC) Dig. Tech. Papers*, Feb. 1995, pp. 88–89.

[12] V. S. Gierenz, O. Weiss, T. G. Noll, I. Carew, J. Ashley, and R. Karabed, "A 550 Mb/s radix-4 bit-level pipelined 16-state 0.25-μm CMOS Viterbi decoder," in *Int. Conf. Application-Specific Syst., Architectures, and Processors*, 2000, pp. 195–201.

[13] N. Bruels, E. Sicheneder, M. Loew, J. Gliese, and C. Sauer, "A 2.8 Gb/s, 32-state, radix-4 Viterbi decoder add-compare-select unit," in *Symop. VLSI Circuits Dig. Tech. Papers*, 2004, pp. 170–173.

[14] P. J. Black and T. H. Meng, "A 1 Gb/s, four-state, sliding block Viterbi decoder," *IEEE J. Solid-State Circuits*, vol. 32, no. 6, pp. 797–805, June 1997.

[15] C. M. Rader, "Memory management in a Viterbi decoder," *IEEE Trans. Commun.*, vol. 29, pp. 1399–1401, Sept. 1981.

[16] G. Feygin and P. Gulak, "Architectural tradeoffs for survivor sequence memory management in Viterbi decoders," *IEEE Trans. Commun.*, vol. 41, no. 3, pp. 425–429, Mar. 1993.

[17] I. Lee and J. L. Sonntag, "A new architecture for the fast Viterbi algorithm," *IEEE Trans. Commun.*, vol. 51, no. 10, pp. 1624–1628, Oct. 2003.

[18] A. Batra, J. Balakrishnan, G. R. A. J. R. Foerster, and A. Dakbak, "Design of a multiband OFDM system for realistic UWB channel environments," *IEEE Trans. Microwave Theory Tech.*, vol. 52, no. 9, pp. 2123–2138, Sept. 2004.

[19] L. Yang and G. Giannakis, "Ultra-wideband communications," *IEEE Signal Processing Mag.*, pp. 26–54, Nov. 2004.