NASA-CR-200625

Progress Report for the NASA Project Entitled
Architecture and Implementations of a High-Speed Viterbi Decoder for a Reed-Muller Subcode
Grant Number: NAG 5-2938

SUMMARY OF RESEARCH PERFORMED FOR
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Electrical Engineering Division
Goddard Space Flight Center

CIRCUIT DESIGN APPROACHES FOR IMPLEMENTATION OF A SUBTRELLIS IC FOR A REED-MULLER SUBCODE
(A Progress Report)

Shu Lin, Gregory T. Uehara, Eric B. Nakamura, and Cecilia W. P. Chu

Principal Investigator: Shu Lin, Professor
e-mail: slin@spectra.eng.hawaii.edu

Co-Principal Investigator: Gregory T. Uehara, Assistant Professor
e-mail: uehara@spectra.eng.hawaii.edu

Both are with the Department of Electrical Engineering
University of Hawaii at Manoa
2540 Dole Street
483 Holmes Hall
Honolulu, HI 96822

February 20, 1996

INTRODUCTION

In his research, we have proposed the (64, 40, 8) subcode of the third-order Reed-Muller (RM) code to NASA for high-speed satellite communications. This RM subcode can be used either alone or as an inner code of a concatenated coding system with the NASA standard (255, 233, 33) Reed-Solomon (RS) code as the outer code to achieve high performance (or low bit-error rate) with reduced decoding complexity. It can also be used as a component code in a multilevel bandwidth efficient coded modulation system to achieve reliable bandwidth efficient data transmission.

This report will summarize the key progress we have made toward achieving our eventual goal of implementing a decoder system based upon this code.

In the first phase of study, we investigated the complexities of various sectionalized trellis diagrams for the proposed (64, 40, 8) RM subcode. We found a specific 8-trellis diagram for this code which requires the least decoding complexity with a high possibility of achieving a decoding speed of 600 M bits per second (Mbps). The combination of a large number of states and a high data rate will be made possible due to the utilization of a high degree of parallelism throughout the architecture. This trellis diagram will be presented and briefly described. In the second phase of study which was carried out through the past year, we investigated circuit architectures to determine the feasibility of VLSI implementation of a high-speed Viterbi decoder based on this 8-section trellis diagram. We began to examine specific design and implementation approaches to implement a fully custom integrated circuit (IC) which will be a key building block for a decoder system implementation. The key results will be presented in this report.

This report will be divided into three primary sections. First, we will briefly describe the system block diagram in which the proposed decoder is assumed to be operating and present some of the key architectural approaches being used to implement the system at high speed. Second, we will describe details of the 8-trellis diagram we found to best meet the trade-offs between chip and overall system complexity. The chosen approach implements the trellis for the (64, 40, 8) RM subcode with 32 independent sub-trellises. And third, we will describe results of our feasibility study on the implementation of such an IC chip in CMOS technology to implement one of these subtrellises.

1. Background and Implementation Considerations

We will begin this section with a brief discussion of the system block diagram in which the proposed decoder is assumed to be operating. Next, we will examine advantages of the proposed architectures for implementation of the Viterbi decoder along with design considerations which result. Following this we will present the architecture we have chosen for implementation of the decoder system.

System Block Diagram

A simplified block diagram of a receiver in which the proposed decoder may be used is shown in Fig. 1. The signal enters the receiver via an antenna and is first amplified by a low noise amplifier (LNA) before being passed to the 2-PSK demodulator. We assume the functions of carrier and timing acquisition and gain control are properly performed in the demodulator. The output of the demodulator is sampled at the correct phase at the symbol rate of 960 MHz. The output of the sampler is converted to the digital domain by the 3-bit analog-to-digital converter (ADC) for decoding by the Viterbi Decoder block which follows. Our discussion will focus exclusively on the implementation of the Viterbi Decoder.

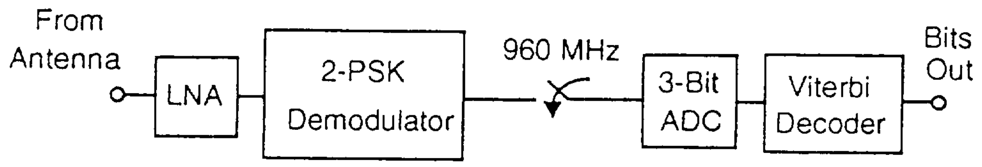


Figure 1 Block diagram of a high speed satellite receiver employing 2-PSK signalling and a Viterbi Decoder.

Summary of System Level Architectural Considerations

In our earlier report [1], we describe in detail the different ways in which parallelism can be utilized to decode the (64, 40) RM code. We will briefly present a summary of that description in this section.

There are many diverse issues at different levels of the design requiring consideration for implementation of the (64, 40) RM code at a rate of 600 Mbits/sec. Fig. 2 illustrates the different layers of hierarchy associated with the proposed implementation. First, there are N parallel decoders with each operating on a different independent block of 64 symbols. Given a decoder which can decode a 64-symbol block at a certain rate, using N decoders and having them each operate on a different block of 64 symbols allows a throughput N times greater.

Second, each decoder is implemented with K parallel isomorphic subtrellises. As described in [6], the trellis for an RM code can be decomposed into parallel isomorphic subtrellises that are connected at only the inputs and outputs as shown conceptually in Fig. 2 with K parallel subtrellises. This has a tremendous advantage for IC implementation because it minimizes the amount of routing required within the trellis which would otherwise be unrealizable at high speed for applications requiring large numbers of states. This is the key which makes an implementation using CMOS IC's at such a high rate and complexity possible.

And third, there are a number of parameters associated with the implementation of each of the K subtrellises. The first is the number of sections in the subtrellis denoted as L. Next, is the number of states at the end of each section i ($i = 1, 2, \ldots, L$) denoted as $|S_i|$ which will generally not be the same. Finally, there is the radix of each section denoted as R_i for radix R in section i. As the number of sections L decreases, the complexity of each section and the number of parallel branches per section increases. These trade-offs are discussed in detail in [1].

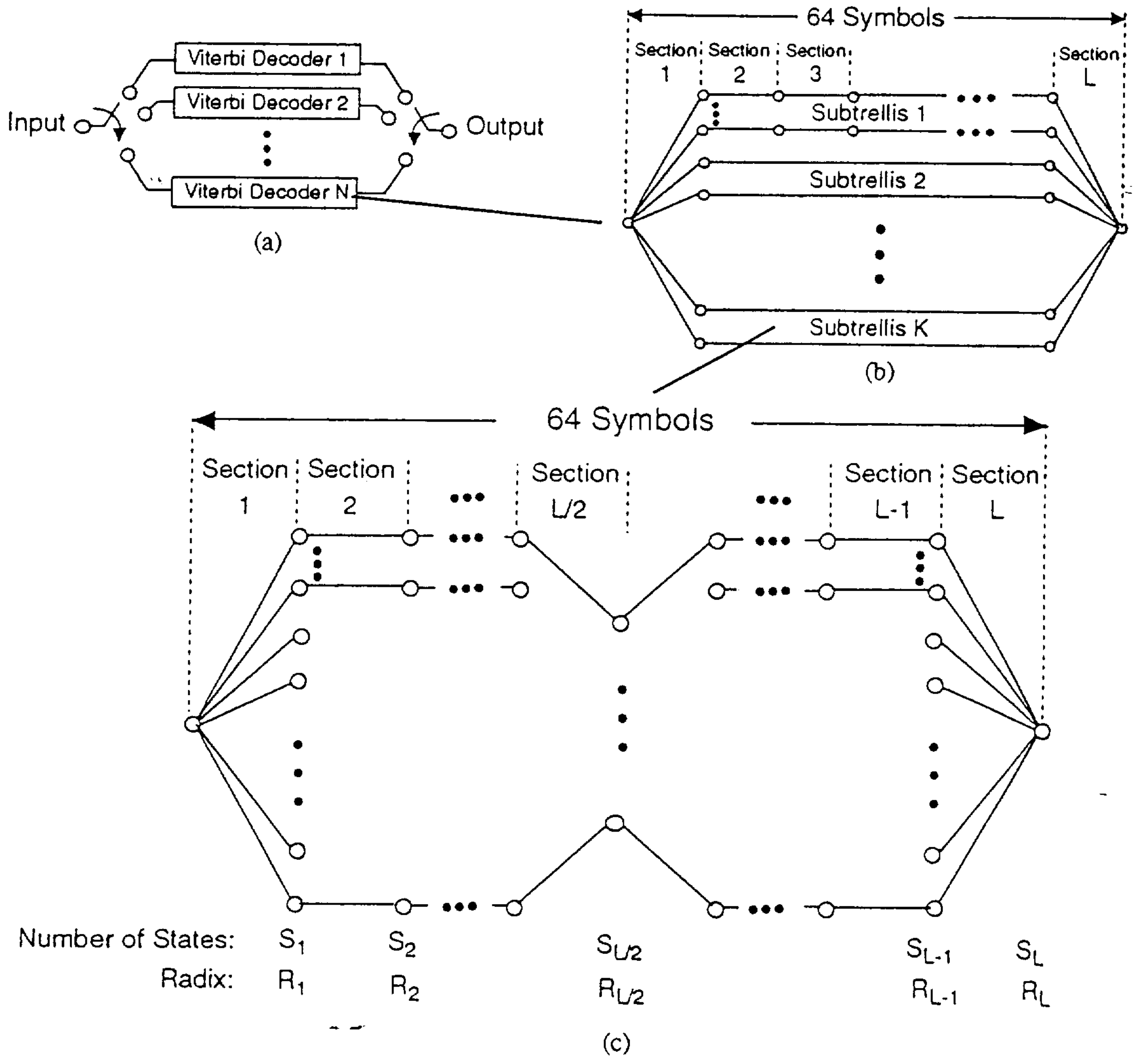


Figure 2 Levels of hierarchy in the proposed Viterbi decoder implementation. (a) Parallel Viterbi decoders operating on different blocks of data. (b) Implementation with K parallel isomorphic subtrellises. (c) Subtrellis implementation.

2. Architecture Chosen for Implementation

In this section, we will present the architecture we chose (over two other candidates) to investigate for implementation of the decoder and present some of the approaches we have developed for implementation of this architecture.

In Fig. 3 is the 8-section trellis which we are investigating for implementation of the decoder. It illustrates the form of two of the parallel isomorphic subtrellis for this chosen architecture. Atop the trellis is the number of subtrellises required to implement the decoder. The numbers inside the subtrellises indicate the number of states in that particular section of the trellis. Below the trellis is the radix at each stage of the trellis.

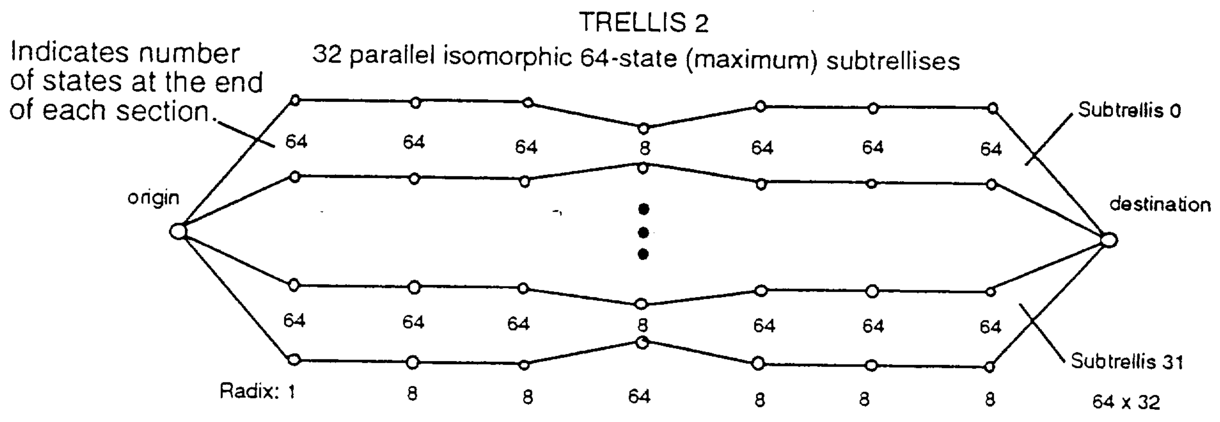

TRELLIS 2
32 parallel isomorphic 64-state (maximum) subtrellises
Indicates number of states at the end of each section.
origin
destination
Subtrellis 0
Subtrellis 31
64 x 32
64
8
Radix: 1

Figure 3 The 8-section architecture we are investigating for implementation of the 600 Mb/sec Viterbi decoder for the (64,40,8) RM subcode.

Implementing one of the 32 subtrellises on a single chip at such a high speed will not be trivial and will require full custom circuit design. From a yield/cost standpoint, the die size of an IC should be kept on the order of 10 mm on each side (100 mm^2). This and other factors were considered in choosing Trellis 2 for further investigation.

The detailed structure of one of the subtrellises for Trellis 2 is shown in Fig. 4. As can be seen in the figure, the 8-way ACS is a critical building block for implementation of this subtrellis. As described in [1], the approach we are examining is based upon a customized 8-way ACS block which is used with comparators to implement the radix-64 section in Section 4 of the subtrellis.

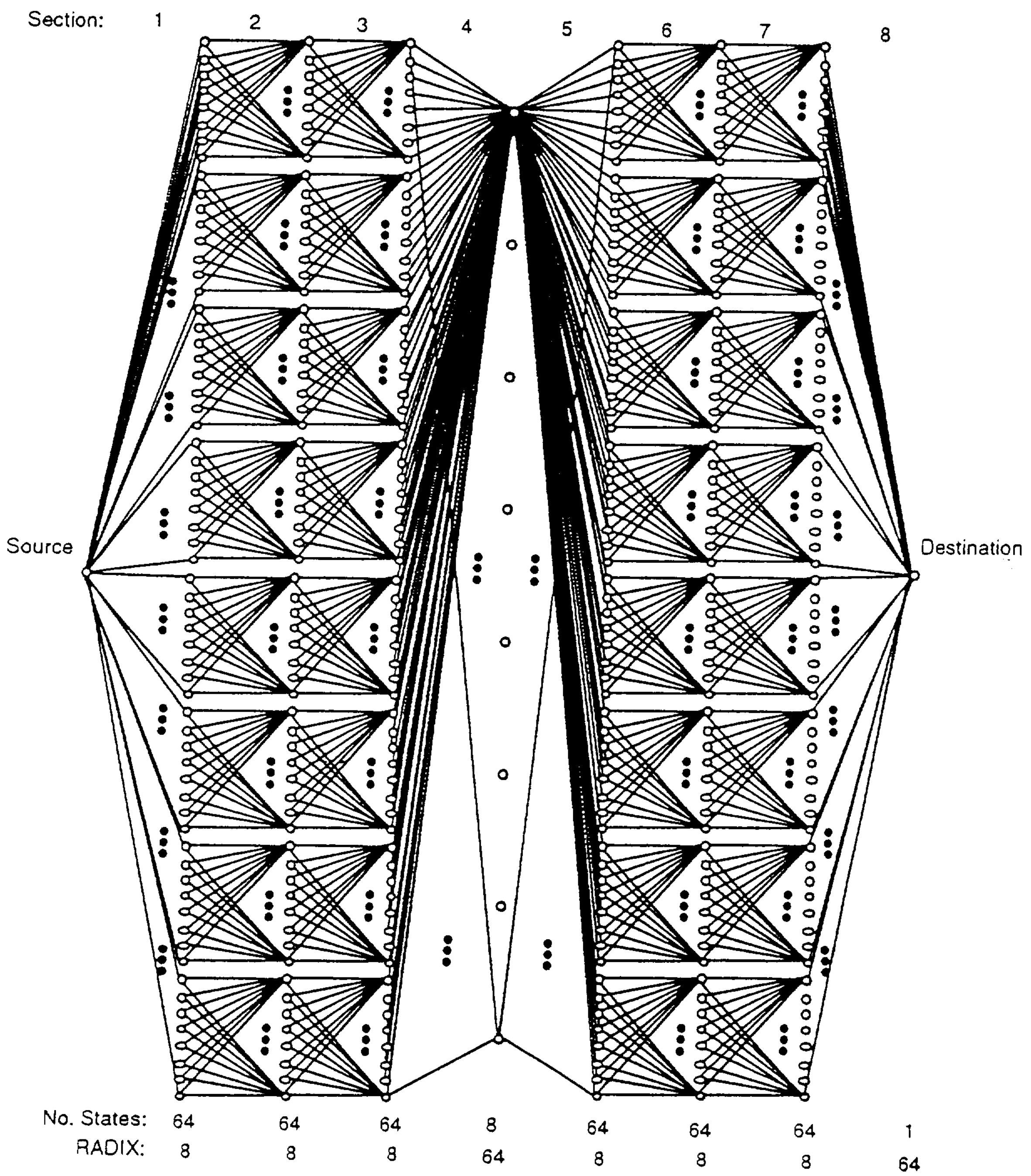


Figure 4 Detailed subtrellis structure for Trellis 2.

3. Chip Plan and Key Results from the Past Year

The key to the implementation of a (64, 40) RM decoder will be the successful implementation of an IC implementing the subtrellis described in the previous section. In this section, we will present some of the key results from the feasibility study of the past year in which we examined the issues associated with such an implementation.

The key objectives of the subtrellis IC implementation are to:

1. Maximize the efficiency as measured by maximizing the utilization of the hardware (in other words, attempt to minimize the time the majority of the hardware is not being used).
2. Use a chip plan which minimizes the area used for routing (routing area is simply an overhead which should be minimized).
3. In whatever the available technology, attempt to approach the speed of 600 Mbits/sec with the minimum number of parallel decoders (in other words, attempt to attain the highest possible speed in a given technology subject to the constraints in the next objective).
4. Consider reliability and robustness issues. In particular, use the lowest speed system clock possible which allows high speed operation in order to reduce the number of issues which can limit the performance (which in this case would be clock skew between chips or race conditions both within and between the different ICs.
5. Consider the board design and the numbers of inputs and outputs to each chip to facilitate implementation of the final decoder system.
6. Keep the size of the IC on the order of 10 mm per side to facilitate its implementation and yield for testing.
7. Utilize the most aggressive IC technology available to our design team at the time of the design.

In this section, we will examine 4 key aspects of the design including the sequence to be used to decode the 8 sections of the subtrellis, the overall chip plan, and some of the details associated with the design of the 8-way ACS and the decoder.

Decoding Sequence

Due to the inherent nature of block codes, they can be decoded either sequentially or out of order as shown in Fig. 6. The arrow in Fig. 6a indicates how a trellis is typically decoded sequentially, starting with Section 1 and on through to Section 8. In Fig. 6b is another approach where, first, Sections 1 through 4 are decoded sequentially and path information corresponding to the most likely paths into the center 8 states which are the destination states in Section 4 are stored. Next, Sections 5 through 8 are decoded starting from Section 8 and moving back through to Section 5. The path metrics corresponding to the most likely paths into the 8 destination states at the end of Section 5 (moving right to left) are then added to those which were found into those states from the first 4 sections. The two paths (entering the center 8 states) with the largest path metric sum comprise the most likely path through the trellis.

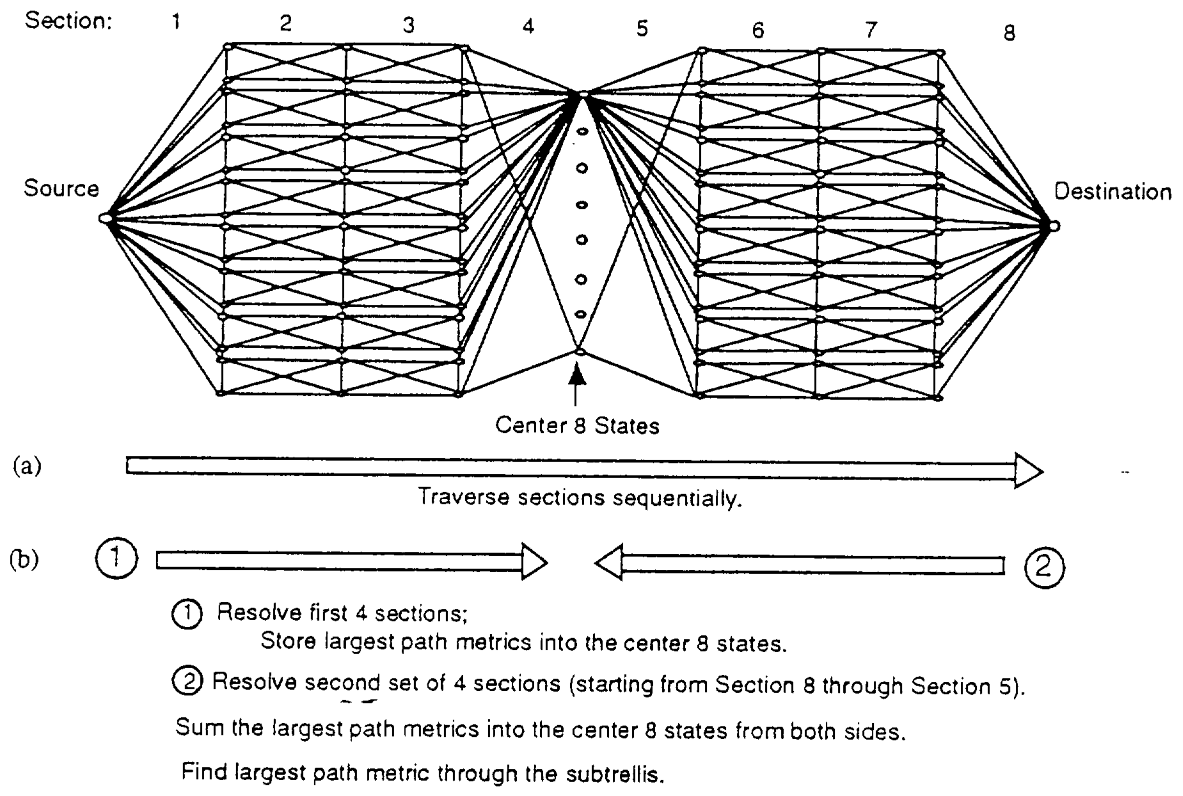
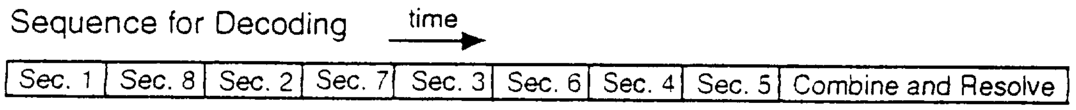

Section: 1 2 3 4 5 6 7 8
Source
Destination
Center 8 States
(a)
Traverse sections sequentially.
(b) ① ②
① Resolve first 4 sections;
Store largest path metrics into the center 8 states.
② Resolve second set of 4 sections (starting from Section 8 through Section 5).
Sum the largest path metrics into the center 8 states from both sides.
Find largest path metric through the subtrellis.

Figure 5 Two possible decode paths for the subtrellis. (a) Traverse all sections sequentially. (b) Traverse in two sections.

The approach we have adopted is a third approach which we call the *modified concurrent bi-directional execution sequence*. This approach exploits the use of pipelining in the ACS implementation and the mirror symmetry of the subtrellis about the center axis (the 8 center states) and results in potential advantages in terms of both speed and structural regularity. Sections are decoding starting from Section 1 and then Section 8, Section 2 and then Section 7, and on down the line until the center is reached and the entire path is resolved as in approach (b) illustrated in Fig. 5.



Sec. 1	Sec. 8	Sec. 2	Sec. 7	Sec. 3	Sec. 6	Sec. 4	Sec. 5	Combine and Resolve

Figure 6 Sequence for decoding using the modified concurrent bi-directional execution sequence.

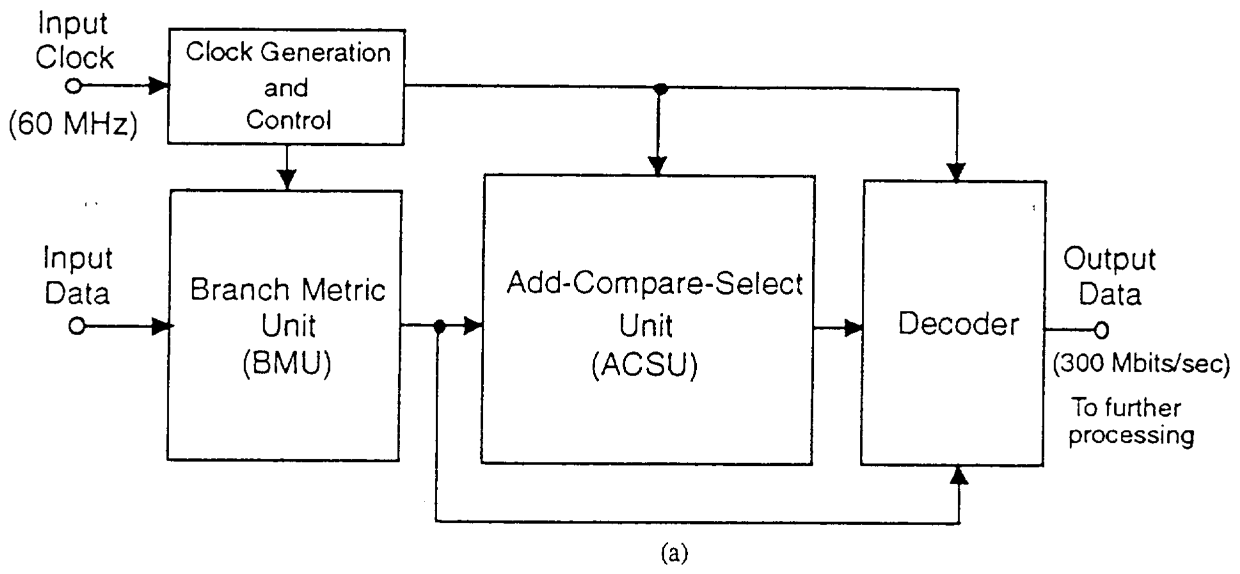
Chip Plan -- Block Level Overview

An outline of the overall chip plan illustrating the major blocks is shown in Fig. 7a. The *Clock Generation and Control* block will generate the necessary clock phases to clock the chip. Input data will enter the *Branch Metric Unit* (*BMU*) which will generate the branch metrics for the *Add-Compare-Select Unit* (*ACSU*). The outputs of the ACS Unit include the winning path metrics and the winning branch labels. These are input to the *Decoder* which determines the most likely path through the subtrellis for the 64-symbol block.

Pipelining is used extensively within the BMU, ACSU, and the Decoder. Preliminary circuit design suggests that to achieve a 600 Mbits/sec decode rate in a 0.6 μm CMOS process, 2 decoders operating in an interleaved manner will be required. As a result, each will be required to operate at a 300 Mbits/sec rate. The symbols will enter the chip at a 300 Mbits/sec x (64/40) = 480 Msymbols/sec rate. The incoming symbols will be separated into groups of 8 3-bit symbols and enter the chip at a 480 M/8 = 60 MHz rate. We currently plan to have the input clock to the chip clock at this 60 MHz rate.

A tentative design for the BMU employs pipelining and takes 3 cycles of the input clock to generate the branch metrics for one section of the trellis. This is indicated in the timing diagram in Fig. 7b with a 3 clock cycle delay from the instant that input data is latched to the time at which branch metrics for a section are output. Each of the stages are shown with the movement of data corresponding to Section 1 indicated with a darkened timing bubble. The outputs of the BMU are input to the ACSU which after 3 cycles of the clock generates outputs for the first section which are passed to the decoder. With each subsequent clock, the ACSU outputs path metrics and branch labels in the order presented in Fig. 6. After the outputs for Section 5 are generated, the decoder then has all the information it needs to determine the most likely path through the subtrellis. Extensive simulations were performed examining different circuit and architectural approaches for implementation of the ACSU. Since this block is potentially the bottleneck to high speed performance and will consume the majority of chip area, much time was spent investigating various permutations of pipelining and parallelism and algorithmic approaches until settling on one which we believe to best meet the various design considerations.

The final decode function is not a trivial one due to the size and amount of data output from the ACSU. During its operation, the ACSU finds the most likely paths from the start of the subtrellis to each of the 8 states at the end of Section 4 and the end of the trellis traversing back through Sections 8-5 to the same location. The decoder must then combine these most likely paths and determine the most likely path from the start to the end of the subtrellis. It must do so while keeping track of the winning branch labels of the partial paths in order to output this information along with the winning path metric to the off-chip post processing which follows. The off-chip processing then determines the path most likely among the most likely from each of the 32 subtrellis. The functions which comprise the decode function are also pipelined although this is not indicated explicitly in the figure.



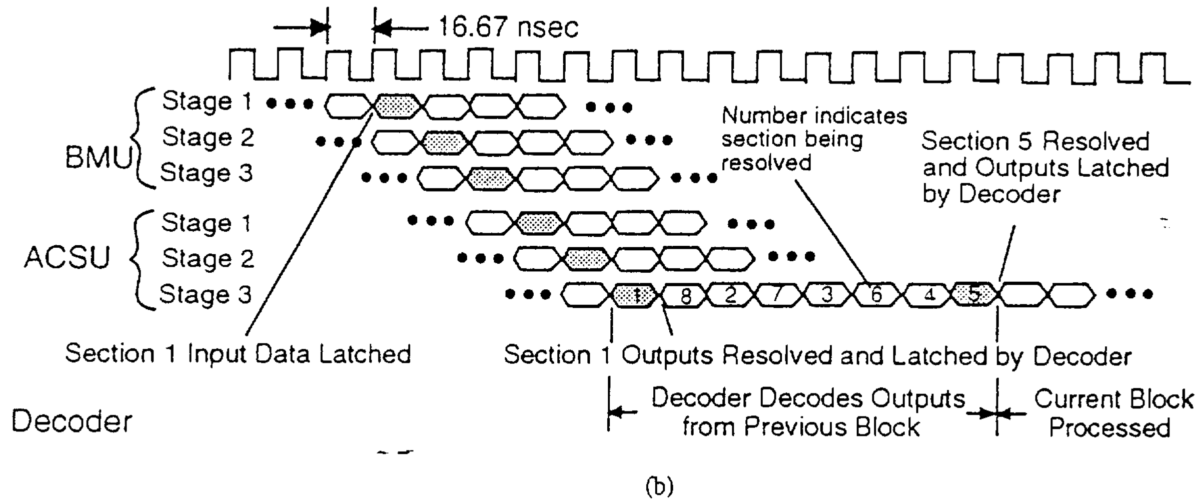

16.67 nsec
BMU
Stage 1
Stage 2
Stage 3
ACSU
Stage 1
Stage 2
Stage 3
Number indicates section being resolved
Section 5 Resolved and Outputs Latched by Decoder
1 8 2 7 3 6 4 5
Section 1 Input Data Latched
Section 1 Outputs Resolved and Latched by Decoder
Decoder
Decoder Decodes Outputs from Previous Block
Current Block Processed
(b)

Figure 7 (a) Block diagram of the IC being developed to implement a subtrellis. (b) Basic high level timing diagram.

Let us now briefly examine some of the details associated with the implementation approaches developed for implementation of the subtrellis IC. We will focus on our developments for implementation of the 3 major blocks: BMU, ACSU, and Decoder.

Branch Metric Unit (BMU)

The 64 symbol sequence is broken into 8 sections with 8 symbols per section. Thus, the BMU generates branch metrics with 8 symbols input at a time. Each input symbol is a 3-bit word and 8 of these are summed in a variety of ways to generate 64 6-bit sums which are the branch metrics as shown in Fig. 8. These sums are conveniently generated using the pipelined 3-stage approach shown in the figure. The use of pipelining facilitates the implementation to reduce the speed requirements of any one stage. Thus, this will be easily implementable at the desired speed.

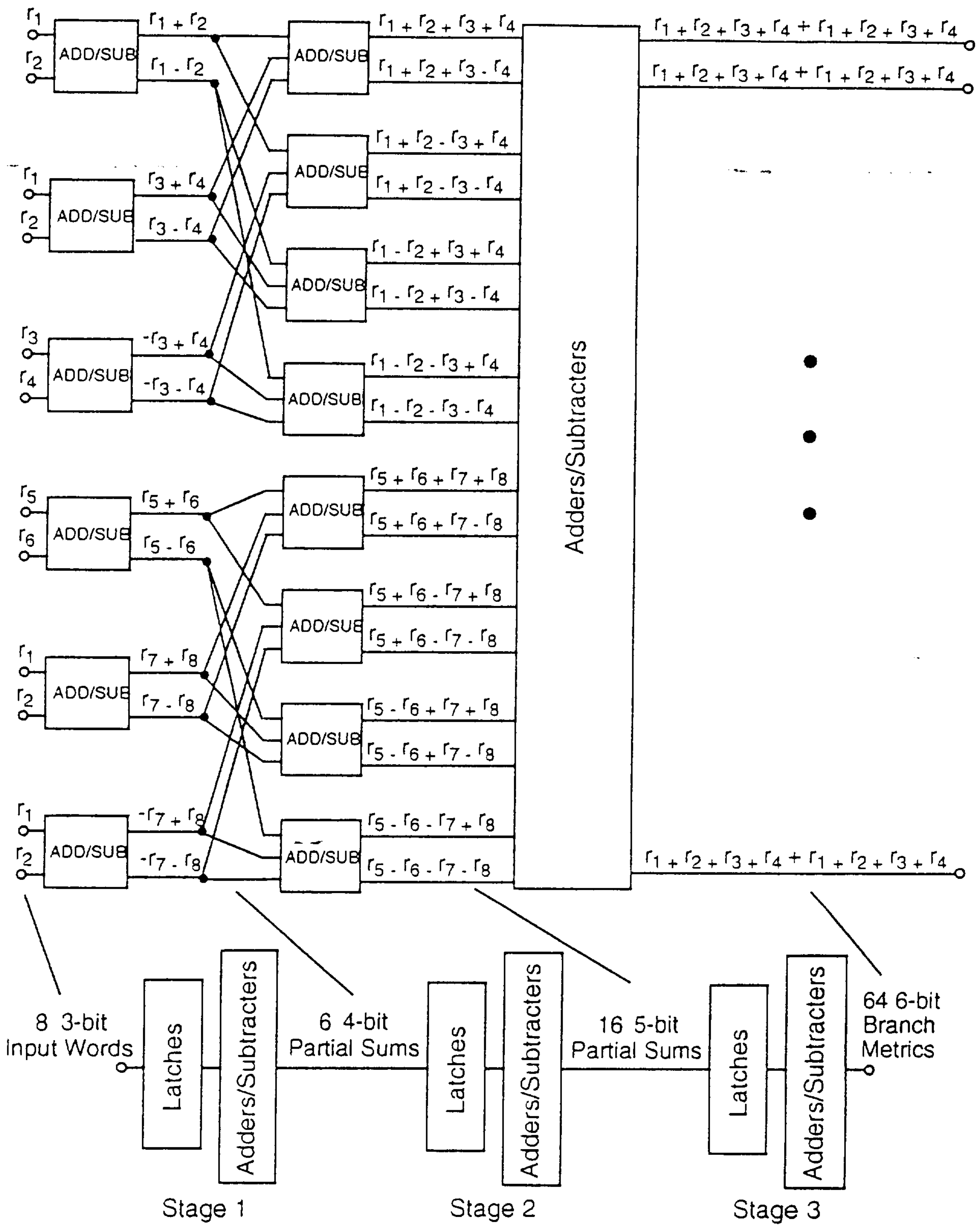


Figure 8 Block diagram of the Branch Metric Unit with a conceptual schematic of the implementation showing pipeline latches below.

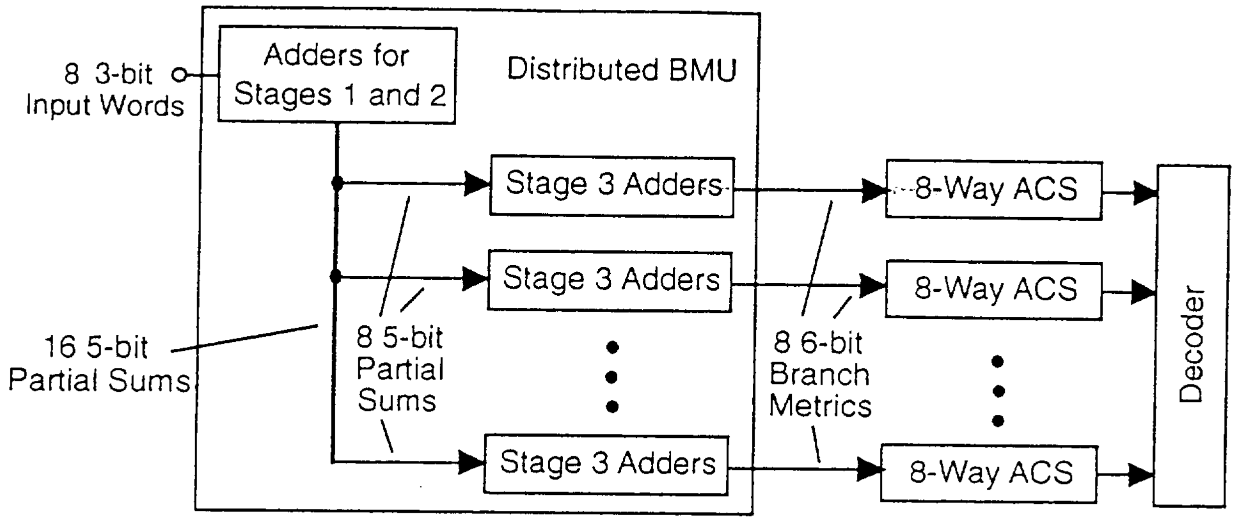


Figure 9 Conceptual layout of the Distributed Branch Metric Unit indicating how Stage 3 of the pipeline is performed locally near each 8-Way ACS to reduce the number of routing lines.

Recall that there are a total of 8 8-Way ACSs on a subtrellis IC. Each 8-Way ACS requires only 8 of the 64 total branch metrics. Therefore, instead of generating all 64 on one end of the IC and routing all 64 6-bit branch metrics along the side of the IC for distribution, Stage 3 of the BMU is separated into custom units which locally generate the 6-bit branch metrics for the ACS units as shown in Fig. 9. This greatly reduces routing and the parasitic capacitance which would otherwise result on the 64 word lines if all branch metrics were generated in one location and passed throughout the IC.

8-Way ACS Unit

The 8-way ACS is the key block for implementation of the sub-trellis decoder. The ACSs will dominate the chip area and will limit the attainable throughput. Therefore, it must be optimized to trade-off speed and chip area.

The ACS is comprised of the well-known add-compare-select function as required in a Viterbi decoder and is shown schematically in Fig. 10 for one section of a radix-8 section. Each of the 8 destination states have a path from each of the 8 source states which makes it an 8-state radix-8 ACS section. A branch metric corresponding to each path into a destination state is added to the accumulated path metrics at each of the source states. These sums are then compared, and the largest sum is the winning path metric for the destination state. This sum is then fed back to become the accumulated path metric for the corresponding state for the next section of the trellis.

There are two outputs from the ACS units: the winning path metric into each state and the path labels corresponding to the winning paths.

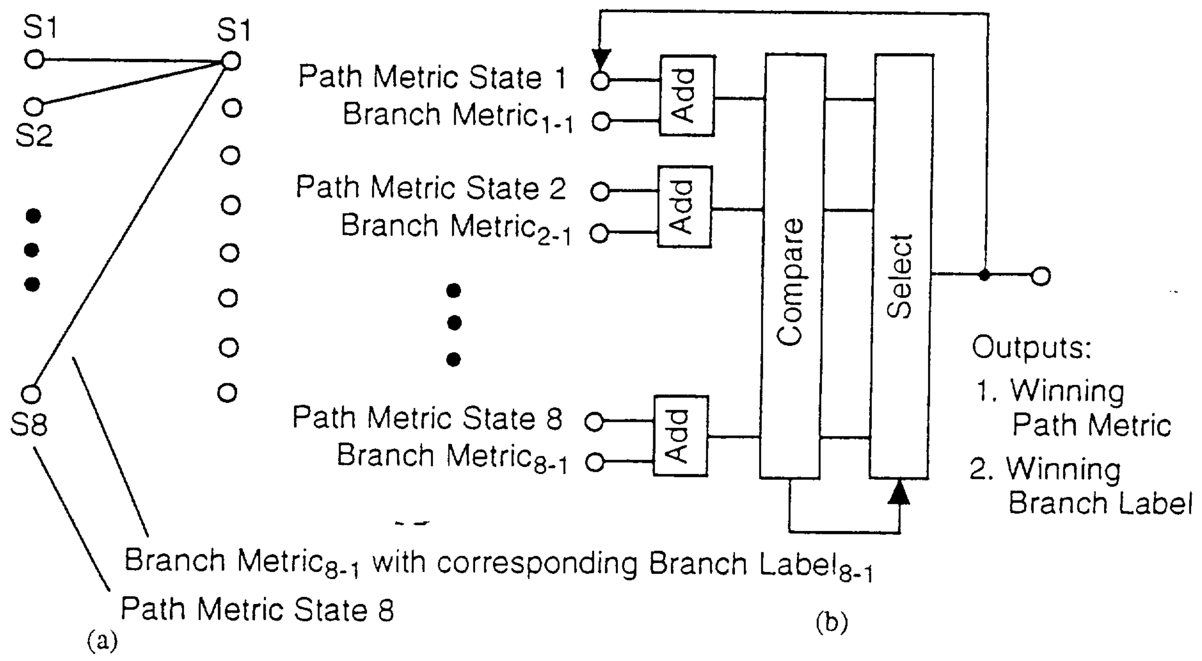


Figure 10 (a) One section of an 8-Way ACS. From each of the 8 states emanates branches to each of the 8 states which follow. (b) Block diagram of the add, compare, and select functions.

Each path between states in the trellis thus far has been shown as a single line. In this decoder, each path is actually comprised of parallel branches. Branch metrics are calculated by adding and subtracting signed numbers. As a result, the branch metrics can be both positive and negative in value. The branch metric for each of the parallel branches between two states is calculated using the same equation and the winning branch is the one with the positive sign. Thus, the winner of the two parallel branches is resolved simply by taking the absolute value of a branch metric as shown in Fig. 11 and sending one bit signifying whether or not a sign change was needed to the circuitry which generates the winning branch label.

The approach shown in Fig. 11 is a straight forward implementation of add, compare, and select and can be mapped directly to hardware. However, it is far from optimum, especially with regard to speed. Let us now focus on implementation of the ACS function.

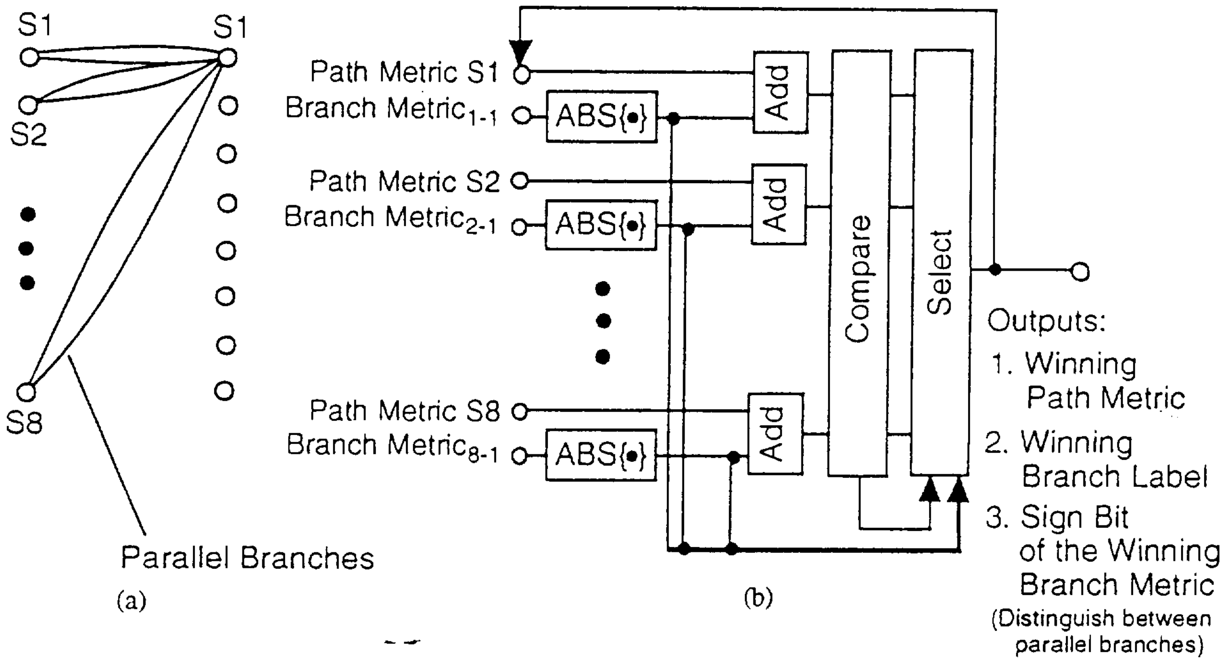


Figure 11 (a) One section of an 8-Way ACS indicating parallel paths. The magnitude of the branch metrics from the parallel paths is the same but the signs are different. Our implementation always uses the metric with the positive sign. (b) Block diagram with the absolute value function added which operates on the branch metric. Information whether or not the sign is changed is used by the decode circuitry.

Different methods have been proposed for implementation of the ACS function. The key trade-offs include speed, silicon area, and required input clock frequency for a given throughput and is a strong function of the radix of the stage as described in [1]. One approach implements this block using word level pipelining for radix-4 ACS units [3,4]. This method determines the largest of 4 numbers using 6 parallel comparators followed by some post processing logic. Each of the 6 possible pair combinations are input to the 6 comparators and the post logic determines the largest of the four. This is attractive for a radix-4 implementation but not for a radix-8 implementation due to speed and area requirements.

Another approach [2] implements the entire ACS using bit-level pipelining and a contention-based approach for implementation of the compare-select functions as shown in Fig. 12. In this approach, there are two paths for the *N* adder outputs. One is through a block labelled Output Largest Sum which uses a contention-based approach for determining the largest sum. It begins by comparing all the MSBs in sequence down to the LSBs. The comparison begins with all *N* inputs in contention. Losers disqualify themselves until only one of the *N* inputs is left. This output is the largest of the sums and becomes one of the *N* path metrics for the section which follows. The second path is through a circuit which determines the label associated with the largest sum. This general approach was selected for the prototype implementation. The approach described in [2] utilizes bit level pipelining. As a result, a relatively high clock speed is required and there is the need for a large number of pipeline latches. We investigated pipelining at bit and word levels as well as some combinations of the two, and settled on the approach which best suits the trade-offs outlined earlier for implementation of the 8-Way ACS.

As described earlier, the clock to the system is a 60 MHz clock. Each of the functions including add, output largest sum, and determine the winning label requires one clock cycle. Thus, the combination of the add and output largest sum functions require two clock cycles. This would normally create a recursive bottleneck because the winning path metrics in a section must be known before computing the following section. As a result, we developed the concurrent bi-directional approach described earlier where we resolve sections of the trellis starting at Section 1 then Section 8, followed by Section 2 then Section 7, etc... Using this approach, while the largest path metrics in Section 1 are being computed for use in the Section 2 calculations, the adders can be used (due to the use of pipelining) to begin processing Section 8.

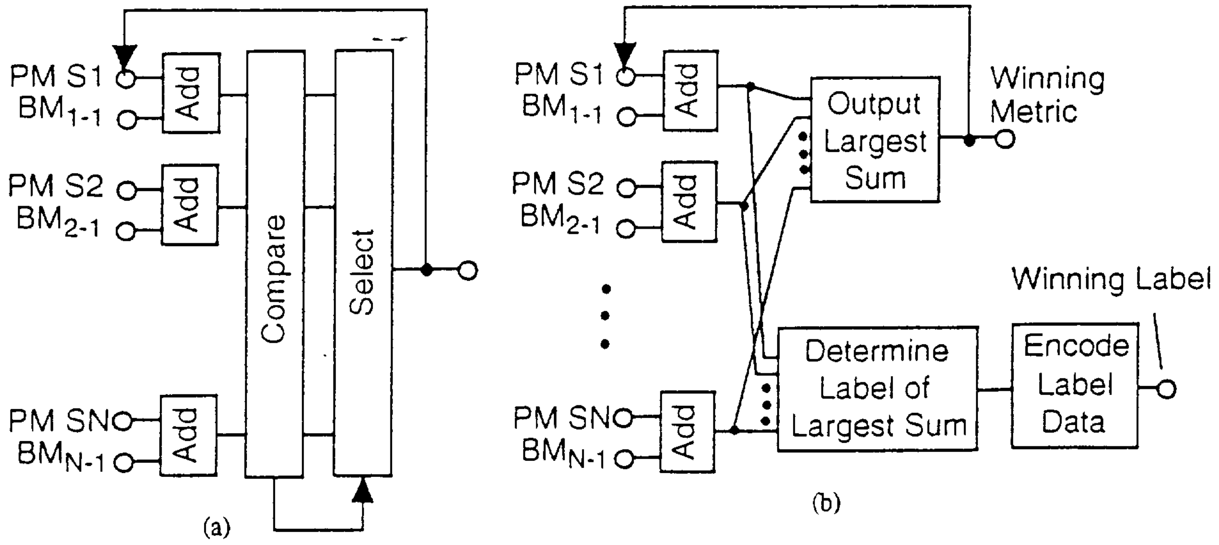


Figure 12 (a) Conventional ACS block diagram. (b) Block diagram with independent determination of the largest metric and the corresponding label and the encoding of this label using the contention approach [2].

The structure of one of the ACS blocks we have developed for implementation of the 8-way ACS is shown in Fig. 13. The implementation we propose is entirely bit sliced except for the encoder which determines the winning label. The key difference between the approach we propose and [2] is that latches are introduced at the outputs of the major blocks so that pipelining occurs at the word level as opposed to the bit level. While this results in a small reduction in attainable speed, it has two important advantages. First, is reduced area because of the need for far fewer intermediate latches. This is important due to the large number of ACSs required on a subtrellis IC. Second, the required clock speed is much lower which should result in more robust overall implementation.

Each 8-Way ACS is comprised of 8 parallel radix-8 sections. Each of the 8 states have paths into 8 destination states. The winning path metrics into the destination states recursively become the starting path metrics for the section which follows as shown in Fig. 13b. In this way, each of the sections of the trellis is resolved using the same 8-Way ACS section as described in our earlier report. We refer to a section which performs the radix-8 operation for a particular state as a *state slice*. One 8-Way ACS is comprised of 8 parallel states slices as shown schematically in Fig. 13c. The 8 winning path metrics are input to each of the state slices. Thus, the state slices are laid out in such a way that these winning metrics are passed through each of the slices for convenient access.

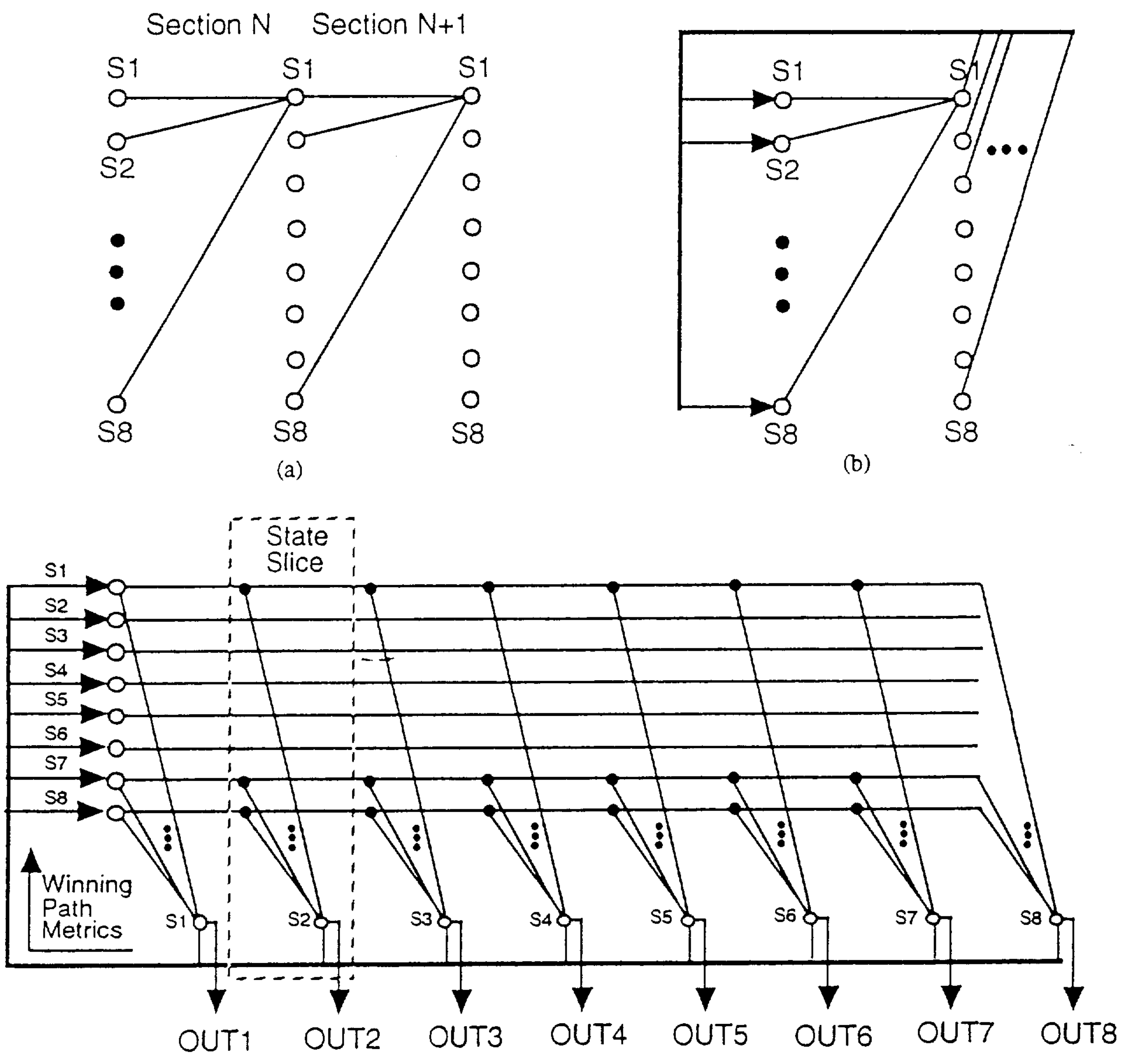


Figure 13 (a) Cascade of radix-8 sections as present in the subtrellis. (b) Block diagram of a recursive ACS unit used to resolve each radix-8 section. (c) Conceptual layout of a full 8-Way ACS indicating the definition of a State Slice which implements one radix-8 section.

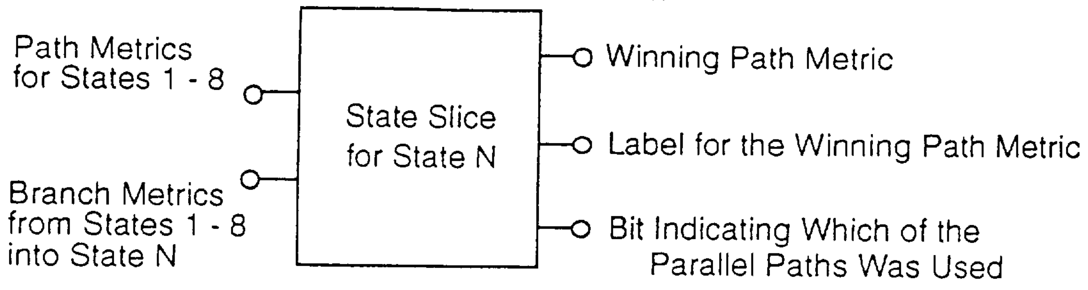

Path Metrics for States 1 - 8
Branch Metrics from States 1 - 8 into State N
State Slice for State N
Winning Path Metric
Label for the Winning Path Metric
Bit Indicating Which of the Parallel Paths Was Used

Figure 14 Inputs and outputs of a State Slice for State N.

The inputs and outputs of a state slice are shown in Fig. 14. Information of the winning labels and a sign bit indicating which of the parallel branches is the winner are output for every section. However, the accumulated path metrics are only needed after Sections 1 - 4 and Sections 8 - 5 have been resolved. These are then summed by circuitry which follows to determine the most likely path through a subtrellis.

Each state slice is implemented using a bit-slice architecture as shown conceptually in Fig. 15. Each slice operates on a different bit location of the input words and implements the entire ACS function for that particular bit. For example, one slice operates on the MSBs of each of the path metrics and the branch metrics and outputs the MSB of the winning path metric along with data that is encoded into the winning path label. Information such as carry bits are passed between the bit-slices as needed. The overall result of this approach is a very regular and compact layout. A plot of the layout of a bit slice is shown in Fig. 16.

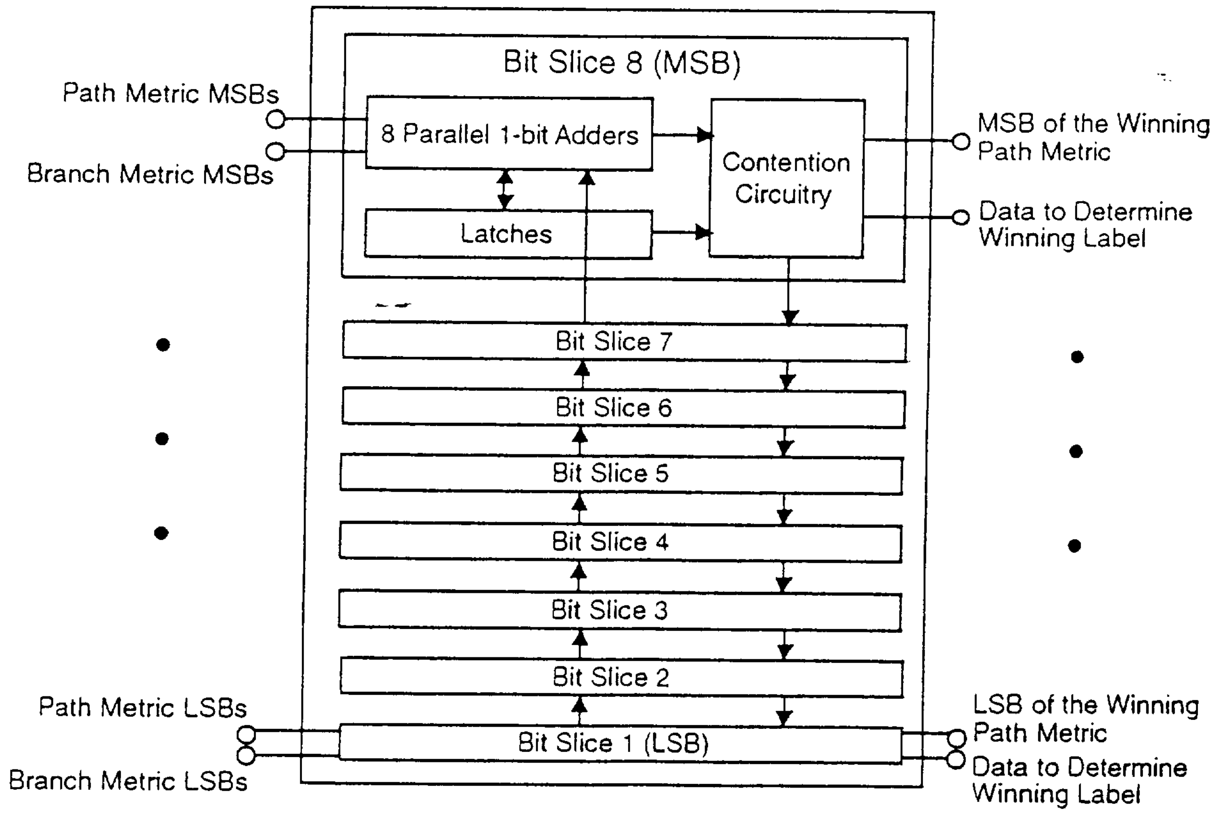


Figure 15 Conceptual block diagram of a State Slice using a bit-slice implementation. In the front-end, 8 additions are performed in parallel with the MSBs of all 16 words input to Bit Slice 8 and other bits input similarly in groups to the LSBs to Bit Slice 1.

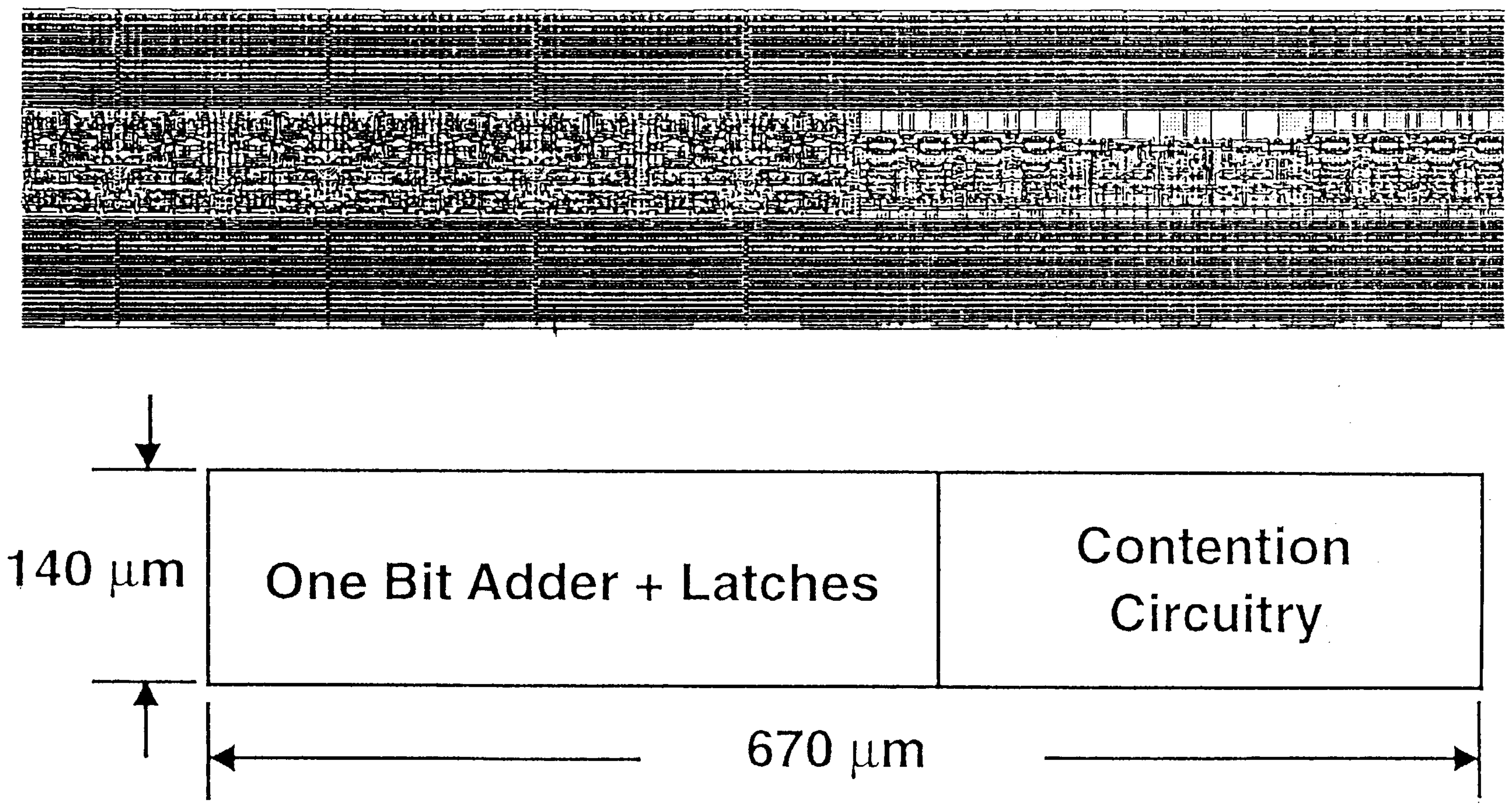


Figure 16 Cifplot of the layout of a bit slice for implementation of the ACS blocks.

Eight of the bit slices of Fig. 16 are stacked to implement a state slice. Eight state slices are placed in parallel to implement an 8-Way ACS. And finally, eight 8-Way ACSs are stacked in order to implement the ACS block for one subtrellis IC. The ACS block output feeds the Decoder block which stores the winning path label information and combines and compares the accumulated path metrics in order to determine the most likely path through a subtrellis. This information is then output from the IC. The outputs of the ACS blocks to the Decoder block are shown as they appear in time in Fig. 17.

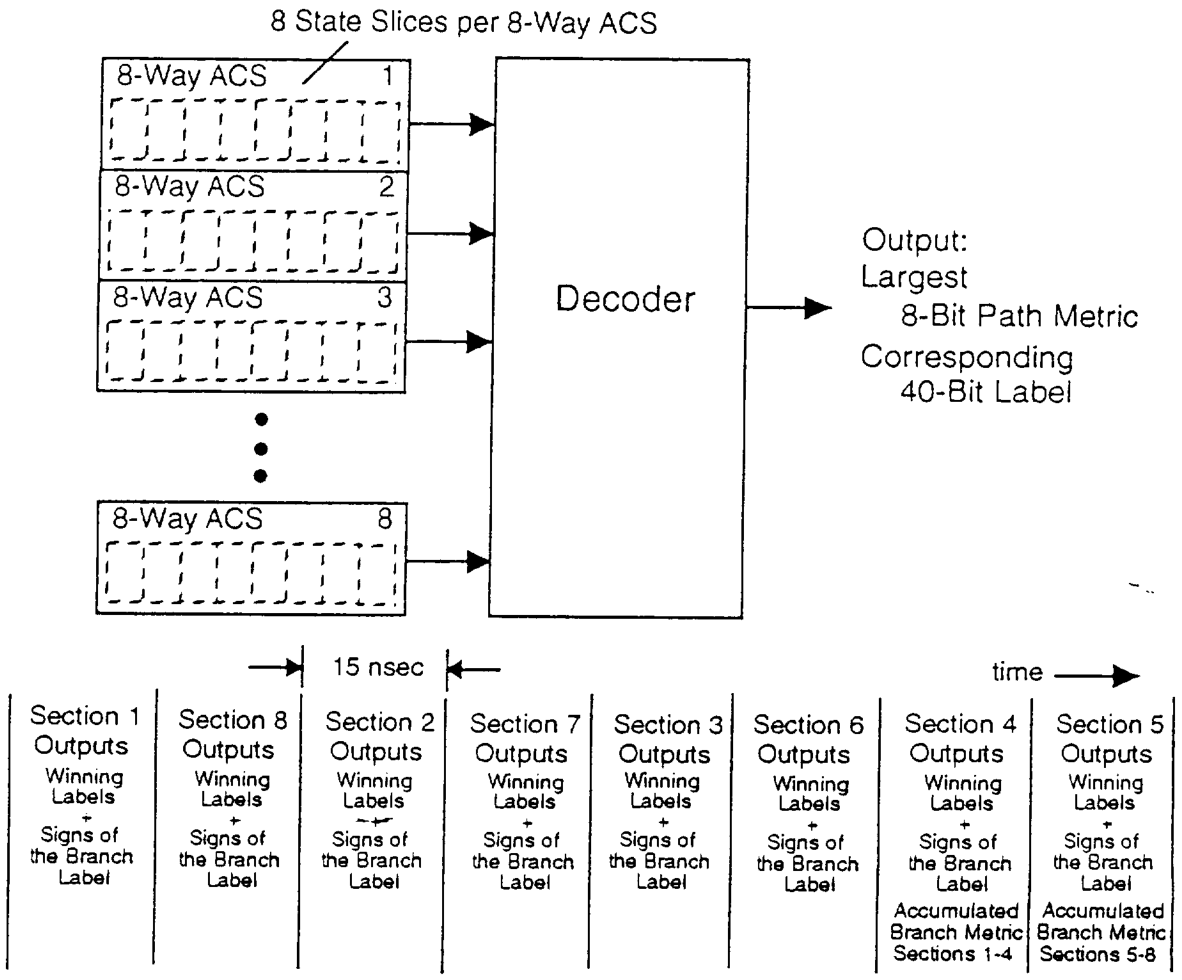

8 State Slices per 8-Way ACS
8-Way ACS 1
8-Way ACS 2
8-Way ACS 3
8-Way ACS 8
Decoder
Output:
Largest
8-Bit Path Metric
Corresponding
40-Bit Label
15 nsec
time
Section 1 Outputs Winning Labels + Signs of the Branch Label
Section 8 Outputs Winning Labels + Signs of the Branch Label
Section 2 Outputs Winning Labels + Signs of the Branch Label
Section 7 Outputs Winning Labels + Signs of the Branch Label
Section 3 Outputs Winning Labels + Signs of the Branch Label
Section 6 Outputs Winning Labels + Signs of the Branch Label
Section 4 Outputs Winning Labels + Signs of the Branch Label Accumulated Branch Metric Sections 1-4
Section 5 Outputs Winning Labels + Signs of the Branch Label Accumulated Branch Metric Sections 5-8

Figure 17 State slices in parallel make up an 8-Way ACS. Eight 8-Way ACSs are stacked to achieve the number required for a sub-trellis implementation. The outputs of the ACSs feed the decoder block. The outputs of the ACSs over a 8 clock sequence are shown.

Decoder

The first function required of the decoder block is to de-interleave the data from the ACS blocks for decoding as shown conceptually in Fig. 18. This is implemented simply as a multiplexor or switch.The remaining functions are:

1. Complete the radix-64 operations needed to complete Sections 4 and 5.
2. Combine the winning metrics from both directions (Sections 1 - 4 and 8 -5) into the center 8 states.
3. Compare these 8 metrics to find the largest through the subtrellis.
4. Determine the corresponding path from the label information.

The first 3 of the functions listed above can be implemented with multiplexors and comparators and will not be described further in this report. The forth is akin to the trace-back problem in a trellis decoder and the general approach we developed for this application will be briefly described.

One way this might be accomplished is to store all the label information, determine the most likely path, then perform the trace-back function. With the large amount of data output from the ACSs, this would be costly and complicated in terms of the required routing.

The approach we have developed instead sorts the output data in a way that the complete paths into the states most recently resolved are always organized and available. In this way, the path corresponding to the winning metric can be obtained simply by multiplexing it out of latches.

The general principle behind this approach will now be described. For simplicity, we will describe a decoder for a 4-state radix-4 system.

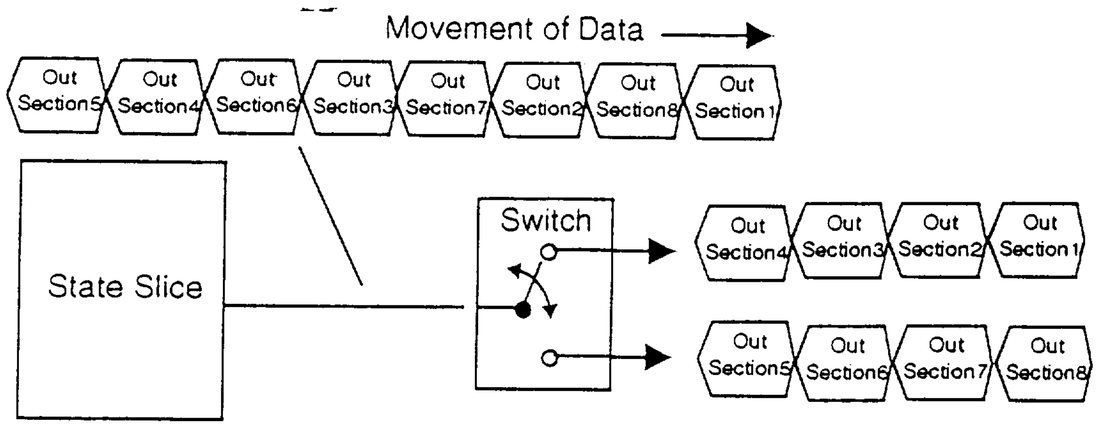


Figure 18 The Decoder front-end contains a switch which de-interleaves the data for decoding of the most likely path.

Refer to Fig. 19a which illustrates a random set of transitions through 4 sections of a 4-state trellis comprised of radix-4 sections. Note that while there is a possibility of 16 paths in each section (4 into each state), the ACS units will determine the most likely path into each state and therefore the decoder circuitry need only contend with 4 pieces of data for each section. Our decoder architecture relies heavily upon this observation.

The basic operation will now be described. At the end of the processing of each section, labels corresponding to the most likely paths into each of the 4 states are input to the appropriate input location into the block labelled *Latches*. The label notation $L_{a,x}$ is the label corresponding to the path between State a and State x. This data is latched and held at the output of the latches at the corresponding state location. At the end of the next clock cycle, a new set of label data is input to the block labelled Latches and multiplexing and latching occurs in the blocks that follow. Data is multiplexed in the blocks that follow in the same manner from input state to output state as the winning paths into the first state of latches. For example if the winning label into S1 of Section 2 is $L_{2,1}$, then the input from the S2 input of ML1 is multiplexed to the S1 output of the ML1 block. This is continued until Section 1 data is at the output of the block labelled ML3. At that point, each of the S4 outputs contain the labels of the most likely path from the initial S1 state through State 4 in Section 4 starting from ML3 back through to the Latch block while the S3 outputs contain the labels for the path starting from the initial S1 state and ending on the S3 state at the end of Section 4. The same holds for the S2 and S1 outputs.

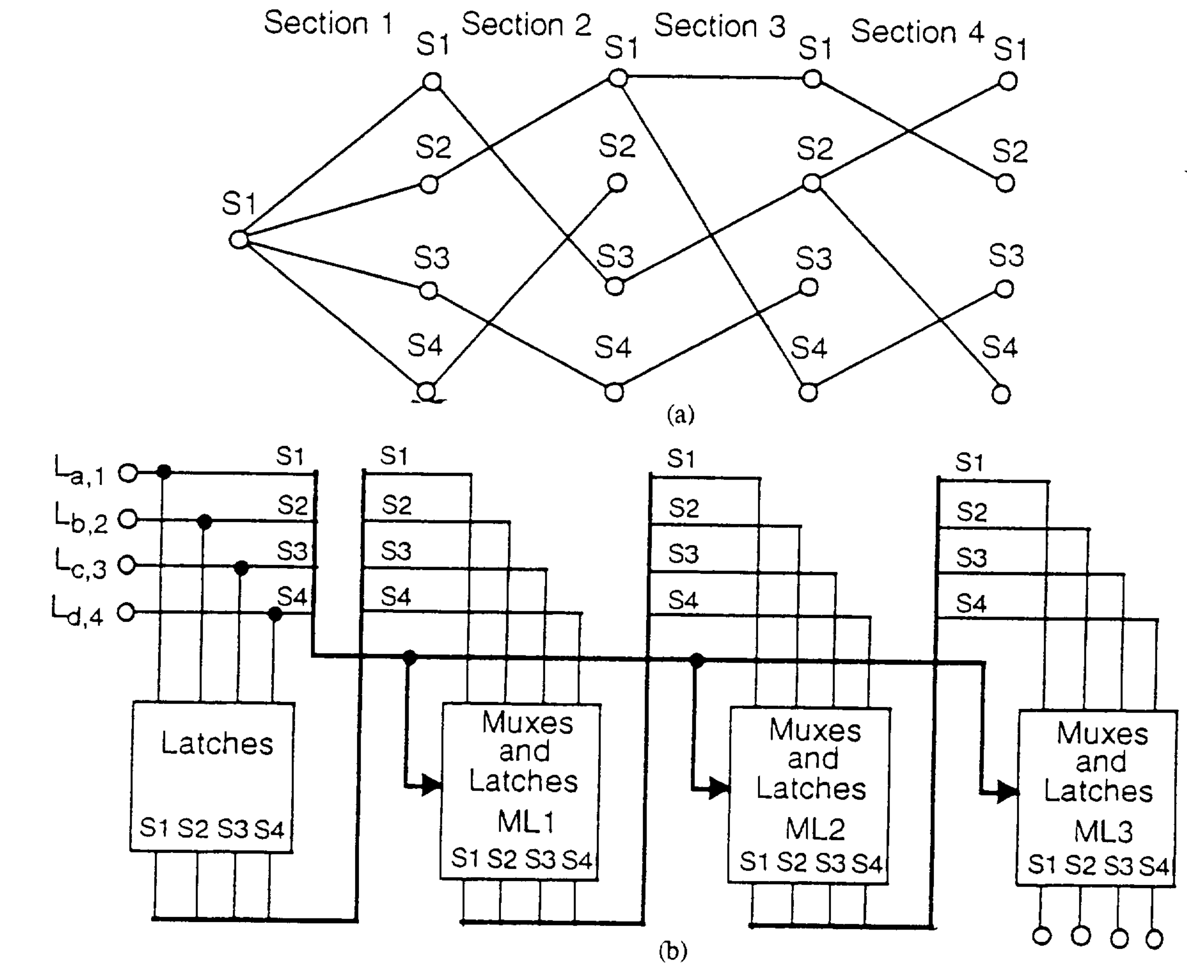


Figure 19 General concept behind the decoder architecture. (a) Example of a 4-state radix-4 trellis (vs. 8-state radix-8 for simplicity). (b) Conceptual block diagram of the decoder hardware.

In order to illustrate this more clearly, let us work through the detailed example shown in Fig. 20. Let us focus in particular on the path ending in S4 at the end of Section 4. This will result in consideration of only the data impacting this path for purposes of this discussion.

Time n — Section 1 is resolved and the winning label into S1 is $L_{1,1}$.

This label is input to the latch on the bus labelled S1 and is latched and held on the output bus labelled S1of the Latch block.

Time n+1 — Section 2 is resolved and the winning label into S3 is $L_{1,3}$.

This label is input to the latch on the bus labelled S3 and is latched and held on the output bus labelled S3 of the Latch block.

Data at the S1 input of ML1 (label $L_{1,1}$) is switched to output S3 in accordance with this winning label into S3.

Time n+2 — Section 3 is resolved and the winning label into S2 is $L_{3,2}$.

This label is input to the latch on the bus labelled S2 and is latched and held on the output bus labelled S2 of the Latch block.

Data at the S3 input of ML2 (label $L_{1,1}$) is switched to output S2 in accordance with this winning label into S2.

Data at the S3 input of ML1 (label $L_{1,3}$) is switched to output S2 in accordance with this winning label into S2.

Time n+3 — Section 4 is resolved and the winning label into S4 is $L_{2,4}$.

This label is input to the latch on the bus labelled S4 and is latched and held on the output bus labelled S4 of the Latch block.

Data at the S2 input of ML3 (label $L_{1,1}$) is switched to output S4 in accordance with this winning label into S4.

Data at the S2 input of ML2 (label $L_{1,3}$) is switched to output S2 in accordance with this winning label into S4.

Data at the S2 input of ML1 (label $L_{3,2}$) is switched to output S2 in accordance with this winning label into S4.

At this point as can be seen in Fig. 20, the S4 outputs of each of the blocks contain labels of the most likely path into S4 of Section 4. Labels corresponding to the most likely paths into each of the 4 states at the end of Section 4 will likewise be in the corresponding output location of the blocks.

The implementation will use a bit slice approach to reduce routing. This block is still in design but we believe this approach will greatly facilitate the implementation of the decoder to achieve the throughput objectives within the target die area constraints.

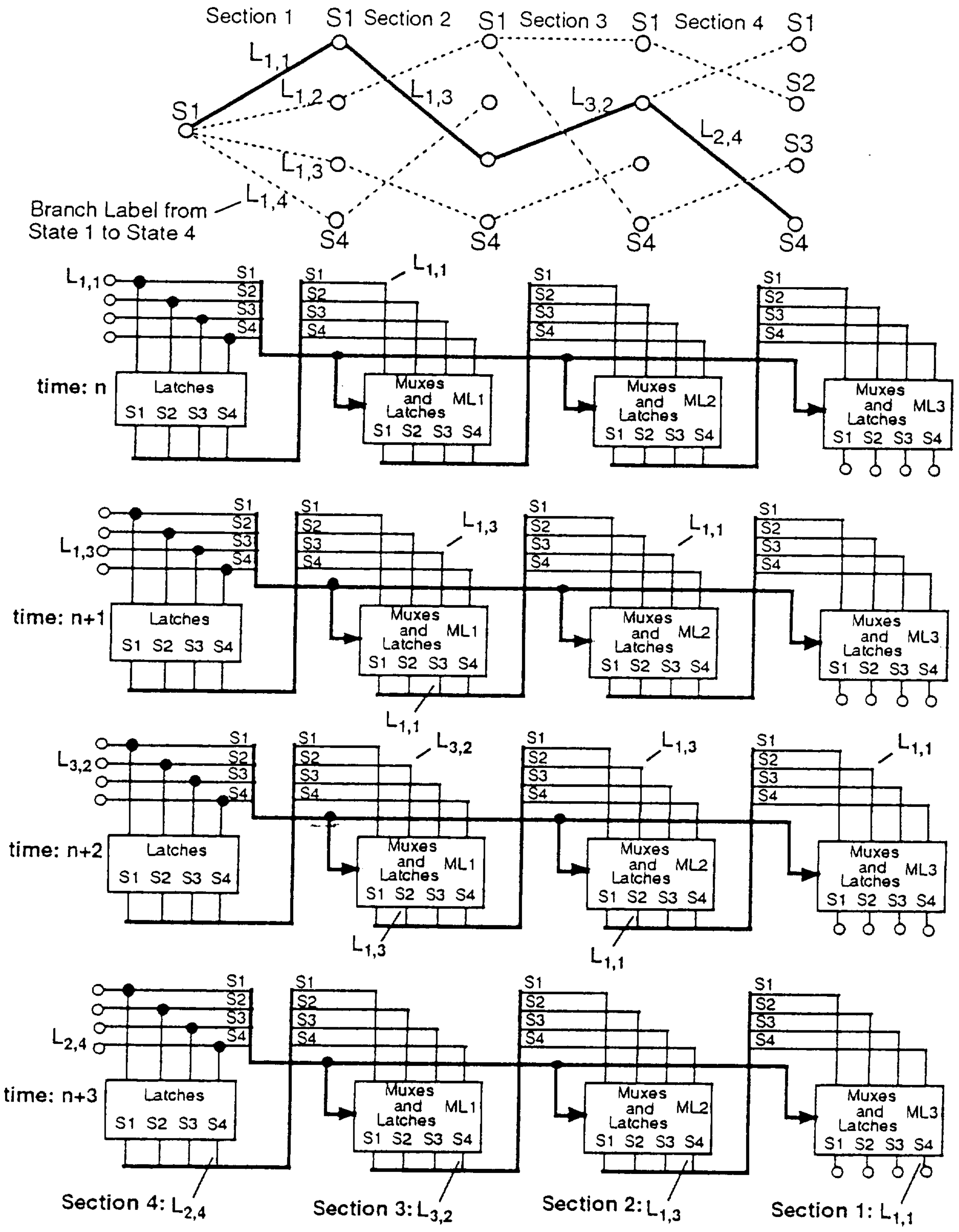


Figure 20 Example illustrating the movement of data within the decoder hardware to develop the path through the four sections into S4 of Section 4.

4. Summary and Future Work

Research Summary

In the first phase of study, we investigated the complexities of various sectionalized trellis diagrams for the proposed (64, 40, 8) RM subcode. We found a specific 8-trellis diagram for this code which requires the least decoding complexity with a high possibility of achieving a decoding speed of 600 M bits per second (Mbps). In the second phase of study which was carried out through the past year, we investigated circuit architectures to determine the feasibility of VLSI implementation of a high-speed Viterbi decoder based on this 8-section trellis diagram. We began to examine specific design and implementation approaches to implement a fully custom integrated circuit (IC) which will be a key building block for a decoder system implementation. This examination was performed in order to study the feasibility of implementing such a decoder at such high speed using primarily CMOS technology.

The results of our feasibility study indicate that it is feasible to implement such an IC meeting the objectives outlined at the beginning of Section 3 in a somewhat optimum manner assuming the use of a 0.6 μm CMOS process which is currently available to us. In this technology, current data suggests that the 600 Mbits/sec speed should be attainable using 2 parallel decoders ($N = 2$ in the Section 1 discussion).

The key results upon which we base this conclusion include:

1. Development of the optimum sequence with which sections of the trellis should be decoded in order to meet the objectives outlined above.
2. Development of an overall chip plan.
3. Circuit design and layout of the ACS unit. This includes scheduling of the data inside the ACS block which has many considerations and a large amount of data in transit.
4. Scheduling of the inputs and outputs to and from the chip and between the major blocks of the chip.
5. Die size in this technology may exceed the 10 mm per side target by up to 20% per side. This target will be easily met in a state-of-the-art technology (0.25 μm CMOS) which in principle should allow the 600 Mbits/sec speed to be implemented with $N = 1$.
6. Preliminary gate level circuit design of over 80% of the major blocks.

Much work still remains in the circuit design, layout, and simulation of the chip. However, we do believe we have solutions to most of the significant design challenges in the key blocks which include the branch metric unit, the ACS block, and the decoder.

Future Work

We will be continuing the development of a decoder system, focusing our current efforts on continuing the development of a full custom CMOS IC to implement a subtrellis which will be the key building block for the system.

The long term goal of this project is to demonstrate performance and implementation advantages of Reed-Muller codes for very high speed, bandwidth efficient communication.

REFERENCES

[1] H. T. Moorthy, S. Lin, and G. T. Uehara, "On the trellis structure of a (64,40,8) subcode of the (64,42,8) third-order Reed-Muller code," NASA Report, NAG 5-931, Report No. 95-001, March 1, 1995.

[2] A. K. Yeung and J. M. Rabaey, "A 210 Mb/s radix-4 bit-level pipelined Viterbi decoder," *ISSCC 1995 Digest of Technical Papers,* San Francisco, CA, Feb. 1995.

[3] P. Black and T. Meng, "A 140 Mb/s 32-state radix-4 Viterbi decoder," *ISSCC 1995 Digest of Technical Papers,* San Francisco, CA, Feb. 1992.

[4] P. Black, "Algorithms and architectures for high speed Viterbi decoding," *Ph.D. Thesis*, Stanford University, May 1993.

[5] G. Fettweis and H. Meyr, "High-speed parallel Viterbi decoding algorithm and VLSI-architecture," *IEEE Communications Magazine*, Vol. 29, May 1991.

[6] T. Kasami, T. Takata, T. Fujiwara, and S. Lin, "On Branch Labels of Parallel Components of the L-section Minimal Trellis Diagrams for Binary Linear Block Codes," *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences*, Vol. E76-A, No. 9, pp. 1411-1421, September 1993.