Parallel Viterbi Decoding Implementation by Multi-microprocessors

AUTHOR(S):

ZHAO, Hui; YUAN, Xiao Kang; SATO, Toru; KIMURA, Iwane

CITATION:

ZHAO, Hui ...[et al]. Parallel Viterbi Decoding Implementation by Multi-microprocessors. Memoirs of the Faculty of Engineering, Kyoto University 1992, 54(2): 105-118

ISSUE DATE:

1992-04-30

URL:

http://hdl.handle.net/2433/281453

Parallel Viterbi Decoding Implementation by Multi-microprocessors

by

Hui ZHAO*, Xiao Kang YUAN**, Toru SATO* and Iwane KIMURA*

(Received December 24, 1991)

Abstract

The Viterbi algorithm is a well-established technique for channel and source decoding in high performance digital communication systems. However, excessive time consumption makes it difficult to design an efficient highspeed decoder for practical application. The central unit of a Viterbi decoder is a data-dependent feedback loop which performs an add-compare-select (ACS) operation. This nonlinear recurrence is the bottleneck for a high-speed parallel implementation. This paper describes the implementation of parallel Viterbi algorithm by multi-microprocessors. Internal computations are performed in a parallel fashion. The use of microprocessors allows low-cost implementation with moderate complexity. An organization network, separate memory blocks and programs provide proper operation. For a fixed processing speed of given hardware parallel Viterbi decoding allows a linear speed up in the throughput rate by a linear increase in hardware complexity.

1 Introduction

The Viterbi algorithm (VA) is widely used for decoding convolutional codes[1]. It is an optimum decoding algorithm but its computational complexity grows exponentially with the constraint length of the encoder. To enhance the achievable throughput rate of an implementation algorithm, parallel and/or pipelined architectures are used. Much research has been done in recent years[2,3]. G. Fettweis[4] proposed a method which can realize the linear scale solution of Viterbi algorithm. P. G. Gulak[5] proposed several pipeline architectures for the Viterbi algorithm. R. Schweikert[6] also proposed a parallel Viterbi algorithm concatenated by single-parity-check coding. Parallel Viterbi decoding by two microprocessors is implemented by the authors[7]. The fully parallel Viterbi decoding algorithm was first proposed by Forney in 1973[11]. The main stumbling block for implementing parallel Viterbi decoding is its complexity in synchronization and contention among the processors and occupation of

* Department of Electrical Engineering II, Faculty of Engineering Kyoto University

** Shanghai Research Institute of Radio Equipment, Ministry of Aerospace Industry, Shanghai 200082, P. R. China.

space by connection lines. Usually pipelined algorithms provide inferior throughput and fewer timing problems, so it is a trade-off between the throughput rate and the implementation complexity. For high-speed implementation of an algorithm, architectures are desired that lead to a linear increase in hardware complexity for a linear speedup in the throughput rate if the limit of computational speed of the hardware is reached.

In this paper a fully parallel implementation of Viterbi algorithm is proposed where one state is assigned for a microprocessor. For a convolutional code of constraint length K there are 2^K states, and hence, 2^K microprocessors. For values of K greater than 8, Viterbi decoding becomes increasingly complex and impractical. On the other hand, the required hardware logic speed is only $1/2^K$ operations per symbol interval. This type of full parallel layout dominated by a large inter-processor wire area, is the architectural organization with the greatest possible throughput for a given microprocessor. In section 2 the architecture of microprocessors is described briefly. Viterbi decoding is introduced in section 3. Parallel Viterbi decoding is found in section 4. Hardware implementation of Viterbi decoding is described in section 5. Experiments and results form section 6. Section 7 summarizes the conclusions of the present paper.

2 Architecture of Microprocessors

Here we design the Viterbi decoder by microprocessors. It is relatively simpler and cheaper as compared to a specially proposed Viterbi decoder made by VLSI.

Figure 1 shows the architecture of a Z-80A microprocessor.

CPU is a Central Processing Unit. Z-80A CPU is an 8-bit microprocessor. It is supplied in a 40-pin dual in-line package. It has 40 lines : the data bus is 8 lines, the address bus is 16 lines, the control bus is 13 lines, the clock signal is one line, and the power

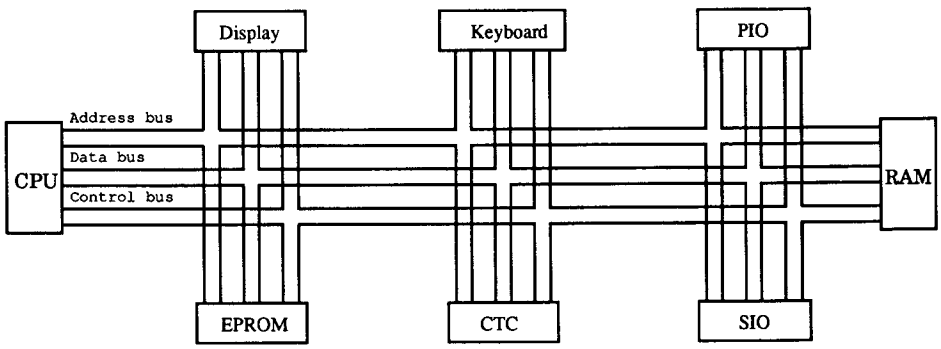

Fig. 1 Architecture of the microprocessor.

occupies two lines.

1. Address bus : Z-80A CPU uses 16 address lines. It is a tri-state output bus : high-level state, low-level state and high-impedance state. The address bus is used to provide the address when exchanging data between CPU and memory, CPU and a peripheral device, or a peripheral device and memory. Memory data exchanges use a 16-bit address bus. I/O data exchanges use the lower 8 bits of the address bus to select one of 256 I/O ports.
2. Data bus : It is a tri-state bidirectional bus. The word length of Z-80A CPU is 8-bit, and therefore the data bus is 8 lines. It can be put into a high-impedance condition if another device is used to supply data to the bus.
3. Control bus : There are three kinds of control lines : 1) system control lines, 2) CPU control lines, and 3) bus control lines.
4. Clock : The clock signal is generated externally to the Z-80A CPU. Timing control for the execution of instructions by the Z-80A CPU is provided by a single -phase square-wave clock signal. Z-80A CPU is available, operating with a maximum clock frequency of 4MHz.

PIO is a Parallel Input and Output interface. The Z-80A PIO is one of a set of chips manufactured to facilitate Z-80A CPU interface. The PIO circuit is designed to provide a two-port, programmable, parallel data transfer between the Z-80A CPU and peripheral devices.

SIO is a Serial Input and Output interface. The SIO is used as a cost effective alternative to parallel communication over long distances. The Z-80A CPU communicates with the serial I/O circuitry via 8-bit parallel bytes. The function of the serial interface is to make the translation from serial-to-parallel or parallel-to-serial.

CTC is a Counter-Timer Circuit. The CTC performs timing and event counting functions. CTC is a programmable component that can operate in either Counter Mode or Time Mode.

EPROM is a Programmable Read Only Memory. It is used for storing special programs, management programs and constant data.

RAM is a Random Access Memory. It is used for storing intermediate results and constant tables. The characteristic of the RAM is that information stored in it will disappear when the power is turned off.

3 Viterbi decoding

The Viterbi algorithm (VA) was first presented by Viterbi as a method of decoding convolutional codes[1]. Since then it has been proven to be a solution for a variety of digital estimation problems. The VA is an efficient realization of optimization sequence estimation

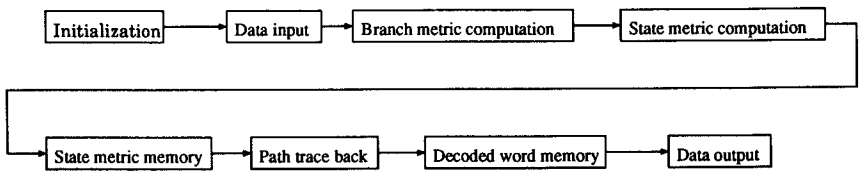

Fig. 2 General serial architecture for Viterbi decoder.

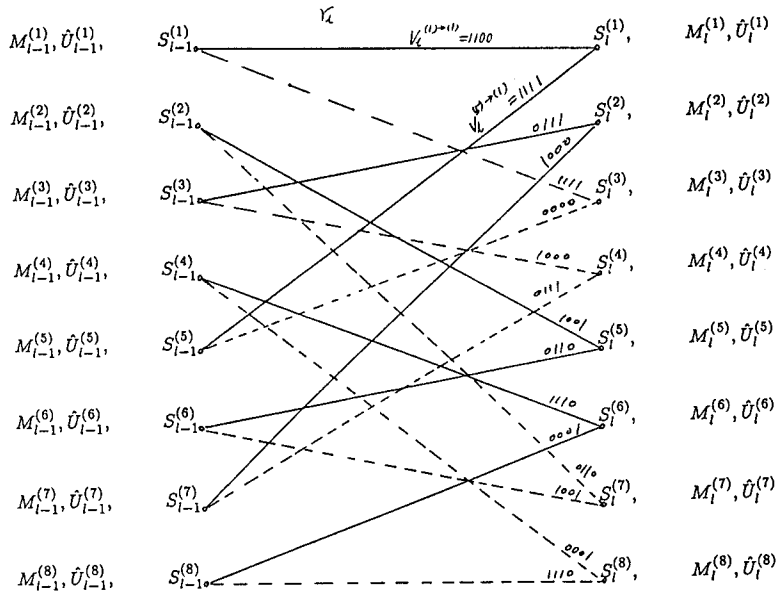

Fig. 3 The l-th stage trellis diagram of (4, 1, 3) convolutional code.

of a finite state discrete-time Markov process where the optimum can be achieved by criteria such as maximum-likelihood or maximum a posteriori. The Viterbi algorithm can be thought of as a dynamic programming solution to the problem of estimating the state sequence of a finite-state Markov process observed in memoryless noise[8]. Forney[9] suggested an application of Viterbi algorithm to symbol-by-symbol data transmission. The basic theory behind the Viterbi decoding algorithm is described in detail in Viterbi and Omura[10], and a good survey is given by Forney[11]. Here we show in Figure 2 the diagram of a serial Viterbi decoder, which consists of the following elements:

1. Data input (Input section). Data is input by SIO and/or PIO interfaces.
2. Branch metric calculation section. Figure 3 shows the l-th stage trellis diagram of the (4, 1, 3) convolutional code with

$$\mathbf{G}=[1+D^2+D^3,\ 1+D+D^3,\ 1+D+D^3,\ 1+D+D^2+D^3]$$

where S_l^i and $S_{l-1}^i(i=1, \ldots, 8,\ l=, \ldots, L)$ denote the ith state at stages l and $l-1$, M_l^i and $M_{l-1}^i(i=1, \ldots, 8,\ l=1, \ldots, L)$ denote the ith state metrics at stages l and $l-1$. $\hat{U}_l^i$ and $\hat{U}_{l-1}^i(i=1, \ldots, 8,\ l=1, \ldots, L)$ denote the decoded information of ith states at stages l and $l-1$. In the diagram, the solid lines mean input branch signal $\mathbf{u}_l=0$, and the dashed lines mean input branch signal $\mathbf{u}_l=1$. The 4-digit numbers over the solid and dashed lines mean branch codeword $\mathbf{v}_l=v_l^{(1)}v_l^{(2)}v_l^{(3)}v_l^{(4)}$. When decoding by a microprocessor, the branch metrics are calculated for all possible branches at every stage $l(m<l<L)$. According to the l-th received branch, $\mathbf{r}_l$(codeword), Viterbi decoder computes the branch metrics leading to every state S_l^i. Since there are always two possible branches leading to the same state, there are two branch metrics $\Delta M_{l,j}^i$ to every state S_l^i ($j=1$, 2 and $i=1$ to 8). Because there are two branches entering each state, there are two branch metrics. The 16 standard branch codewords $\mathbf{v}_l$ are stored in the memory.

3. Path Metric Calculation Section. The branch metrics are used in conjunction with the stored state metrics at stage $l-1$ to compute the path metrics of the present stage l. It reads out an old state metric value stored in a RAM, adds the corresponding branch metric, and gets the path metric.
4. State metric selection section. In this section the Viterbi decoder compares two path metrics leading to a state, and chooses the path metric which has the minimum value as the state metric of the state.
5. Retained path selection section. The Viterbi decoder selects a path which has the minimum metric as the retained path of every state.
6. Output section. According to the retained path, the Viterbi decoder gets the decoding information and sends out the information.

Figure 4 shows the flowchart of the Viterbi decoding.

4 Parallel Viterbi Algorithm

4.1 Principle of parallel Viterbi decoding

The principal limitation on the practical application of the VA is that the complexity of decoding is proportional to 2^K, the number of encoder states. The decoder memory is proportional to 2^K. Also, since 2^K comparisons must be performed per unit time, decoding time is proportional to 2^K. This exponential dependence on K limits the use of the VA to the value of K equal to 8 or less.

The speed limitation of the VA can be alleviated by employing parallel decoding. Since the 2^K comparisons that must be performed at each time unit are identical, 2^K identical

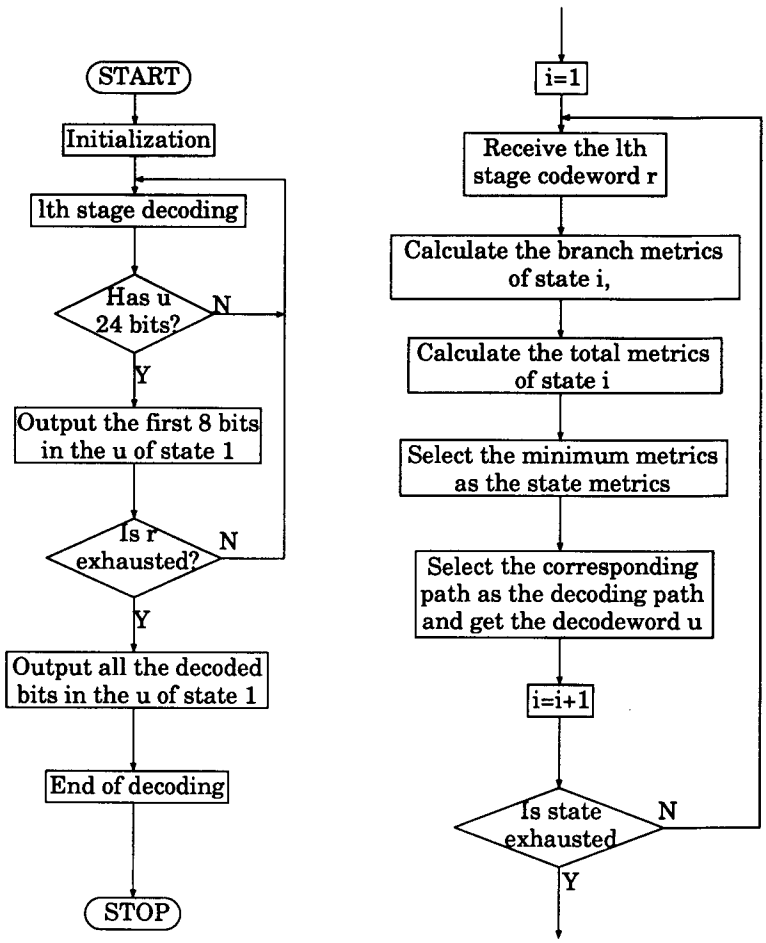

Fig. 4 The Viterbi decoding flowchart. The left panel shows the overall flow, and the right panel describes the l-th stage of the decoding.

processors can be provided to do the comparisons in parallel rather than having one processor doing all 2^K comparisons in sequence. Each processor computes the metric values of the 2 paths entering every state, selects the path with the minimum metric, and stores that path and its metric in the decoder's memory. This parallel implementation of the VA then requires 2^K processors doing one computation per unit time. Parallel decoding therefore implies a factor–of–2^K speed improvement over a standard decoder, but requires 2^K times as much hardware.

MIMD (multiple instruction, multiple data) microprocessor architecture is used for parallel Viterbi decoding illustrated in Figure 5.

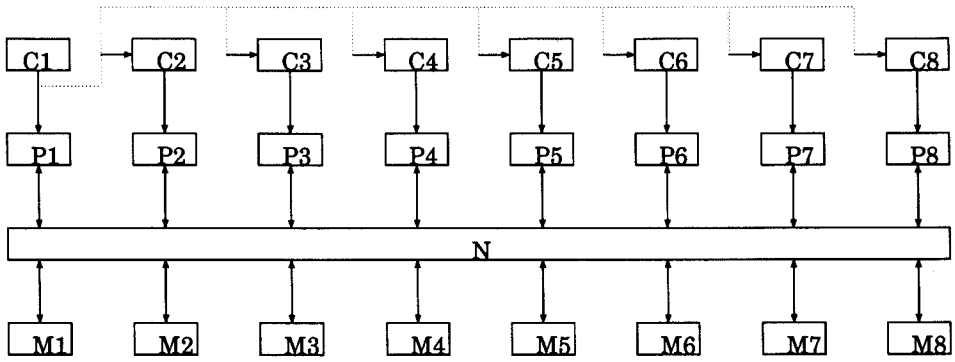

Fig. 5 MIMD Microprocessors, (C: Control Unit; P: Processor; N: Organization Network; M: Memory).

The MIMD machine consists of M processors, M memory units, an interconnection network, and M controllers. Such a multiprocessor system can achieve a linear speed-up factor.

4.2 Problems to be solved in order to realize parallel decoding

In the practical application there are many factors which affect performace, such as:

1. Synchronization: some of the processors may be idle waiting for other processors to catch up.
2. Algorithm: an algorithm designed for a serial machine may not be the most efficient on a multiprocessor machine.
3. Contention: if multiple processors require the same resource they may have to use it one after another.

The MIMD microprocessor architecture for Viterbi decoding has the following characteristics:

1. In each step, $M(=8)$ processors are used simultaneously.
2. All processors in every stage execute different sets of instructions, because each processor is assigned a control unit.
3. The inter-processor communication between two consecutive steps is independent of the results of processing in these steps.
4. There are memory read conflicts.
5. All the processors must compute synchronously at every stage.

Because Viterbi algorithm is related to dynamic programming and has a data dependent feedback loop, the most difficult problems for fully parallel implementation are timing and contention. In order to make the best use of them, waiting periods in computation should

be avoided.

4.3 Facts that affect decoding speed in parallel algorithm

There are some cases in which parallel algorithm is not helpful in reducing time, because some processes cannot be divided into several parts and cannot be assigned multiple CPU's to process them synchronously. As shown in Fig. 4, only the parts in the right panel can be processed in parallel and this part takes most of the time in Viteribi decoding (from state 1 to state 2^K). Processes shown in the left panel of Fig. 4 cannot be processed in parallel. In parallel Viterbi decoding two factors affect the decoding speed: 1) Not all the processes can be done in parallel; 2) Inside the parallel decoding there are also some processes which can not reduce calculating time such as "take the received branch r_l", for which M CPU's will take exactly the same time as one CPU does. In serial decoding one CPU will take the received branch r_l once and in parallel decoding each of M CPU's will take the received branch r_l once.

4.4 Memory organization for survior paths

For a minimum error rate, the length of the survivor path memory should be made as long as possible. The difficulty with long survivor paths is that each of the 2^K words of storage must be capable of storing a long path plus its metric. The compromise is to truncate the path memory of the decoder by storing only the most recent τ blocks of information bits for each survivor. Experience and analysis have shown[12] that if τ is on the order of five times the encoder memory or more, all 2^K survivors stem from the same information block τ time units back, with probability approaching 1, and there is no ambiguity in making the decoding decision. Hence after the first τ blocks of the received sequence have been processed by the decoder, the decoded memory becomes full. After the next block is processed, a decoding decision must be made on the first block of k information bits, since they can no longer be stored in the decoder's memory.

There are several possible strategies for making this decision[12]. Here an arbitrary survivor is selected, and the first k bits on this path are chosen as the decoded bits. After the first decoding decision is made, additional decoding decisions are made in the same way for each new received block processed. Hence, the decoding decisions are always delayed from decoder input by an amount equal to the path memory (i. e., τ block).

A practical case has $\tau=15$ stages and a scheme is used in which after decoding 24 bits of information, the first 8 bits are sent out in parallel. The delay varies randomly from 16 bits to 24 bits.

5 Technical consideration of hardware implementation

5.1 Control units

Control units consist of EPROM's, and management programs and Viterbi decoding programs are stored in them. Every processor is assigned a control unit and instructions for the CPU are stored in the control unit. The CPU works under the control of the instructions. At any given time each processor can execute different instructions, and programs should be designed to let all the processors calculate synchronously in every stage.

5.2 Memory arrangement

For parallel Viterbi Decoding, 8 CPU's will access a memory frequently and synchronously. In this case contention will happen. In order to solve this problem two methods can be used: 1) Let CPU's access memory one after another. In this way there are many wait times. This is what we want to avoid. 2) The other way is to use 8 separate memory blocks. Every CPU is assigned a memory block. Each CPU can access a different memory block synchronously. Memory is divided into $2^K=8$ areas and 8 separate memory blocks are used to store the information of the 8 states. Each memory block will store $M_l^{(i)}$, $M_{l-1}^{(i)}$, $\Delta M_{l,1}^{(i)}$, $\Delta M_{l,2}^{(i)}$, $\hat{U}_l^{(i)}$, $\hat{U}_{l-1}^{(i)}$, $(i=1, \ldots, 8, l=1, \ldots, L)$ as shown in Figure 6, where one memory unit (8 bits) is used for each $M_l^{(i)}$, $M_{l-1}^{(i)}$, $\Delta M_{l,1}^{(i)}$, and $\Delta M_{l,2}^{(i)}$, and three memory units (24 bits) are used for each $\hat{U}_l^{(i)}$, and $\hat{U}_{l-1}^{(i)}$. Eight storage blocks are required in order to assign all CPU's to access RAM simultaneously. Because Viterbi decoding is a dynamic feedback algorithm, when the CPU calculates at a present stage, the information at a previous stage is required. So in each storage block the memory is divided into two areas for storing the information at stage l and at stage $l+1$. At stage l the l-th stage memory area is used to store the information at the l-th stage and at the $(l+1)$-th stage the $(l+1)$-th stage memory area is used to store the message of the $(l+1)$-th stage.

The surviving state metrics for each group of 2^k-state metric which is stored in the arithmetic unit is finite, so that the storage for state metrics must be prevented from overflow. One memory unit (8 bits) is assigned to each $M_l^{(i)}$ state metric, so the maximum value of a state metric is $2^8=256$. When there is overflow, the minimum $M_l^{(i)}$ state metric is selected as a common factor, and it is subtracted from all the state metrics producing the new state metrics.

5.3 Organization Network

From Fig. 3, we can see that every CPU will access three memory blocks: two in the previous stage, and one in the present stage. Every memory block will be accessed by three CPU's. Therefore memory blocks become common resources for CPU's. When CPU's

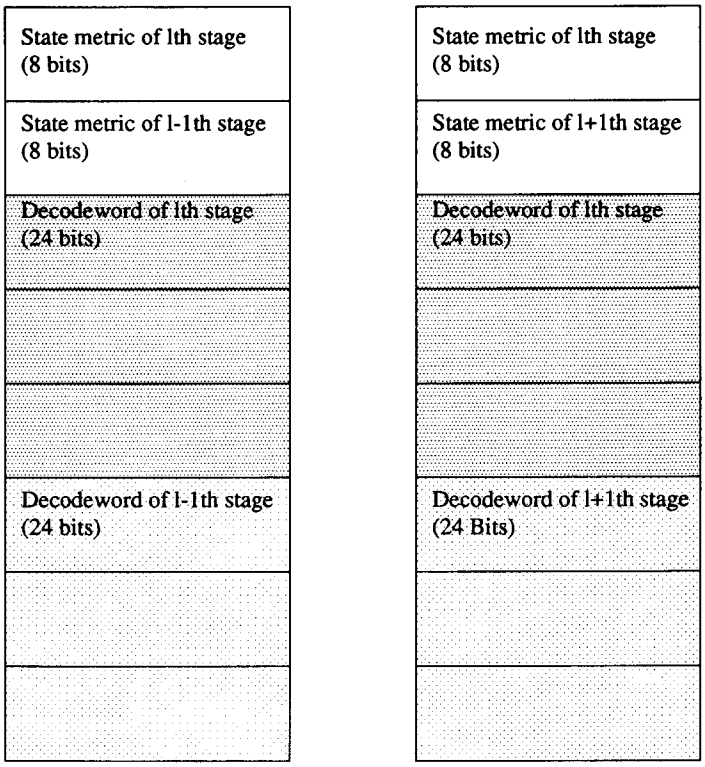

Fig. 6 Distribution of $S^{(i)}$ state store area at l-th stage.

access the same RAM they will use the same bus. So the CPU's will be connected to the common bus. If CPU's are connected to the common bus, they can only use the bus one after another. To solve this problem, an organization network has designed. It has two functions: 1) data transmission among CPU's and RAMs, and 2) preventing contention when CPU's are using same bus. It has the following characteristics: 1) A CPU cannot access another CPU; 2) A CPU can only access one RAM at a time; 3) The network is controlled by CPU's.

Figure 7 shows the architecture of the organization network. It consists of connection lines and three-state control gates. One three-state control gate consists of three 74LS245 chips. One chip can control 8 lines. When a CPU wants to access a RAM block, the corresponding three-state gate turns on and the three buses (data bus, address bus, and control bus) are connected to the RAM. Otherwise, the three-state gate is off, and the three buses of CPU cannot connect to the RAM. If a CPU will access a RAM block, a three-state control gate will be assigned to it. Because every CPU can access three different RAM

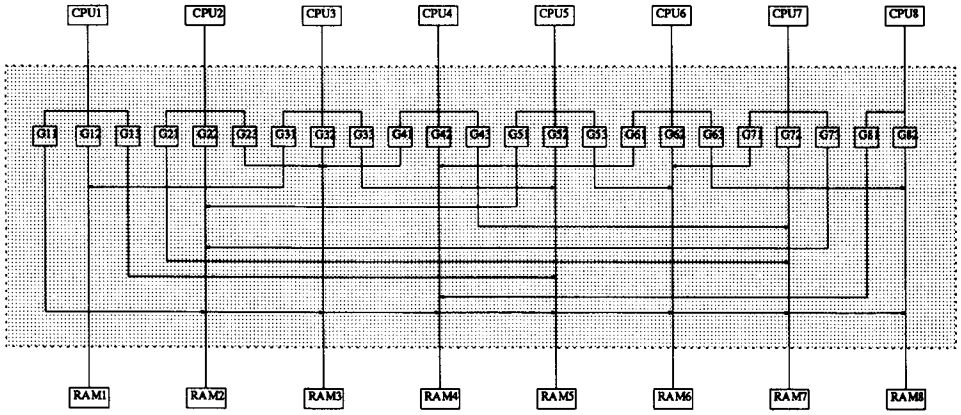

Fig. 7 Organization network.

blocks, three control gates are needed for each CPU. The connection lines and the control gates will occupy a large space. The purpose of use three-state control gates is to make the CPU's isolated from each other. Different CPU's can access different RAM blocks at the same time and the same RAM block at different times, but cannot use the same RAM block at the same time.

5.4 Synchronization of CPU's

Viterbi algorithm is a dynamic feedback algorithm. For parallel operation, synchronization is absolutely necessary, and a precise timing scheme is required. The following two methods are used in realizing synchronization: 1) Central processor controlling synchronization: When decoding start, the central processor sends out a signal, and all the processors start decoding synchronously; 2) Programs synchronization: Because all the processors in any step execute different sets of instructions and all the processors must compute synchronously at every stage, program synchronization is used. That is to let all the processors take the same calculating time at every stage. When some processors use the same memory block at the same time, conflicts will occur. In this design we can avoid that by programs. Because of the structure of the trellis diagram, when we use the M separate memory blocks, every CPU will access three memory blocks. So we can let the CPU's use different memory blocks every time. For CPU's of state 1 and state 3, CPU1 can first access $S_{l-1}^{(1)}$, and then $S_{l-1}^{(5)}$, and CPU3 can first access $S_{l-1}^{(5)}$ and then $S_{l-1}^{(1)}$. In this manner conflicts are avoided and the CPU's can compute continuously without waiting.

As Viterbi decoding is a dynamic feedback algorithm, there are many data exchanges among CPU's and RAMs at every stage. For the prevention of collisions three methods are used together as mentioned above: 1) organization network, 2) assigning M separate

memory blocks to M processors, and 3) program and timing. Collision among CPU's can be prevented only by combining these three methods.

6 Experiments and results

6.1 Experimental setup

First, programs for serial Viterbi decoding algorithm (shown in Fig. 4) are designed and tested on a single microprocessor. The Z-80A microprocessor is used to carry out Viterbi decoding. The programs are designed by assembly mnemonic, and then translated into machine codes by computer. These machine codes are then stored in EPROM's as commands for CPU's. The assembly mnemonic has the advantages of

1. saving the internal memories and CPU resources,
2. having a very high program executing speed,
3. directly using all the resources in a system, and
4. knowing the exactly executing time of CPU.

Second, programs for parallel Viterbi decoding of two CPU's are designed and circuits for two-CPU parallel Viterbi decoding are designed and tested. The two-CPU parallel Viterbi decoder is based on the Z-80A microprocessor. One CPU is borrowed from the Z-80A microprocessor system including a CPU, a PIO, an SIO, a CTC, an internal clock, a power on reset circuit etc.. Circuits for the other CPU system (including an EPROM, a PIO, an SIO, an organization network etc.) are designed. Combining these two parts by an

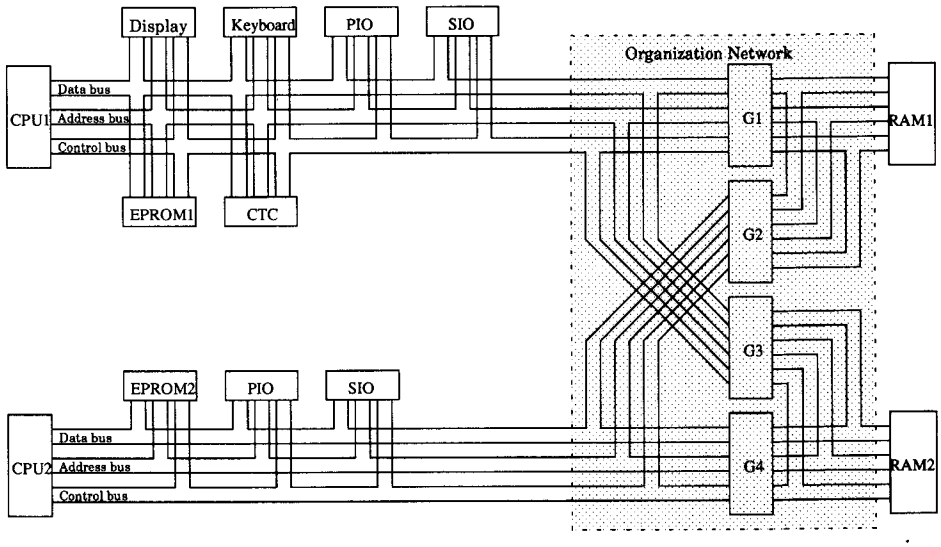

Fig. 8 Parallel Viterbi decoding using two microprocessors.

organization network we get the parallel Viterbi decoder. The parallel Viterbi decoder using two CPU's is shown in Figure 8.

Two CPU's use the same external clock. The CPU in the Z-80A microprocessor system is the main CPU and the other CPU is the sub-CPU. The sub-CPU works only in parallel Viterbi decoding. The main CPU does everything : receive codewords, control the sub-CPU in synchronization, do parallel Viterbi decoding, send out decoding information, and determine whether decoding is finished or not. So the main CPU is a control CPU of the parallel Viterbi decoding system.

When the data transmission rate is higher than the decoding speed, codewords are temporarily stored in a RAM. After solving the previous group of codewords, CPU's process the present group of codewords stored in the RAM. When the transmission rate is lower than the processing speed, the CPU's will wait for the codeword.

6.2 Experimental results

Viterbi decoding speed is determined in two ways :

1. By programs. Programs are designed by assembly mnemonic. Every instruction has an exact execution time. By computing the execution time of instructions in the decoding programs we can know the decoding speed.
2. By test. Another way to get the decoding speed is directly testing the decoding speed.

All of the elements are commercially available chips. Z80A CPU's are used, and the clock frequency is 4 MHz. For serial decoding of one microprocessor the decoding speed is 20 kbits/s. For parallel Viterbi decoding of two CPU's the decoding speed can reach 38 kbits/s. It is expected that for the parallel Viterbi decoding of 8 CPU's the decoding speed can reach 120 kbits/s.

7 Conclusion

In this paper the implementation of parallel Viterbi decoding and the problems which need to be considered in parallel Viterbi decoding by microprocessors have been discussed. Multiple microprocessor architecture is used which results in a decoder of moderate complexity and cost as compared to a specially proposed Viterbi decoder made by VISI. The presented method of implementing the Viterbi algorithm allows the use of hardware with a limited processing speed to achieve a very high throughput rate.

References

1) A. J. Viterbi ; IEEE Trans. Inform. Theory, **IT-13**, 260 (1967)

2) N. J. P. Frenette est ; IEEE Journal SAC, **SAC-4**, 160 (1986)
3) P. G. Gulak and E. Shwedyk ; IEEE Journal SAC, **SAC-4**, 142 (1986)
4) G. Fettweis and H. Meyr ; IEEE Trans. Commun., **COM-37**, 785 (1989)
5) P. G. Gulak and T. Kailath ; IEEE Journal SAC, **SAC-6**, 527 (1988)
6) R. Schweikert and A. J. Vinck ; IEEE Trans. Commun., **COM-39** (1991)
7) H. Zhao ; "Parallel Viterbi Decoding Implementation by Use of Microprocessors", Masters degree dissertation, Shanghai University of Science and Technology, P. R. China (1988)
8) J. K. Omura ; IEEE Trans. Inform. Theory, **IT-15**, 117 (1969)
9) G. D. Forney Jr. ; IEEE Trans. Inform. Theory, **18**, 363 (1972)
10) A. J. Viterbi and J. K. Omura ; "Principles of Digital Communication and Coding", McGraw-Hill, New York (1979)
11) G. D. Forney, Jr. ; Proc. IEEE, **61**, 268 (1973)
12) Shu Lin and D. J. Costello, Jr. ; "Error Control Coding : Fundamentals and Applications", Prentice-Hall, New Jersey (1983)