List of Figures

A New Architecture for the Viterbi Decoder for Code Rate k/n [1]

Hsiang-ling Li and Chaitali Chakrabarti

Department of Electrical Engineering
Telecommunications Research Center
Arizona State University
Tempe, AZ 85287-5706
Ph: (602) 965-9516
Fax: (602) 965-8325
li@dspsun.eas.asu.edu
chaitali@asu.edu

Abstract

A novel VLSI architecture is proposed for implementing a long constraint length Viterbi Decoder (VD) for code rate k/n. This architecture is based on the encoding structure where k input bits are shifted into k shift registers in each cycle. The architecture is designed in a hierarchical manner by breaking the system into several levels and designing each level independently. The tasks in the design of each level range from determining the number of computation units, and the interconnection between the units to the allocation and scheduling of operations. Additional design issues such as in-place storage of accumulated path metrics and trace back implementation of the survivor memory have also been addressed. The resulting architecture is regular, has a foldable global topology and is very flexible. It also achieves a better than linear tradeoff between hardware complexity and computation time.

[1]This research was supported in part by a grant from NSF MIP-9309504.

1 Introduction

The maximum likelihood decoding of the convolutional and trellis codes based on the Viterbi algorithm is an important problem in digital communication. It can be viewed as a technique to find the shortest path in a weighted graph, called the trellis diagram, using dynamic programming [1]-[2]. While the performance of convolutional codes improves dramatically as ν, the constraint length, increases, the decoding effort increases exponentially. Our aim in this paper is to develop architectures for long constraint length codes which are regular, have simple interprocessor connection and are flexible enough to meet a wide range of design requirements.

Traditionally, the implementations of Viterbi algorithm have been (i) *state-parallel* where the path metrics of all states are updated in parallel using multiple path metric updating units (called Add-Compare-Select units or ACS units), (ii) *state-serial* where the path metrics are updated sequentially using a single unit and (iii) a combination of both. Based on these schemes, several architectures have been proposed for the case when the input to the encoder is a single bit (code rate R=$1/n$) [3]-[7],[11][13]. In such a case, the trellis diagram has a simple and symmetric structure, and can easily be mapped into one or more processing units. Many of the architectures for code rate $1/n$ that have been implemented in VLSI [8]-[10] achieve decoding rates as high as 140Mb/s.

For code rate k/n, the encoder has k input bits connected to k shift registers (1 bit per register) [2]. When the sizes of the shift registers differ by at most one bit, the encoder can be modelled as a radix-2^k shift register [12]-[14]. The k input bits are then shifted in bulk into the radix-2^k shift register in each cycle. The output codewords of this structure are the same as those of the original encoder structure. Daneshgaran and Yao [14] have developed an architecture for code rate R=k/n based on this structure. Their architecture is based on a systematic procedure called the Iterative Collapse Algorithm (ICA) which allows downscaling to any level. It also achieves a better than linear tradeoff between hardware complexity and computation time. The number of interconnections between the modules can be as large as that of a fully connected graph, making its implementation cumbersome.

In general, the sizes of the encoder's shift registers may differ by more than one bit and the design of a Viterbi decoder for code rate R=k/n should be based on the original encoder structure. The state transitions for this case are however much more complicated, and the trellis diagram no longer has the simple structure of a $R = 1/n$ decoder. Chang and Yao have proposed an architecture based on the original encoder structure in [11]. Their architecture consists of a systolic array of processors, with each state being assigned to a processor. The disadvantages of their architecture are that it cannot be scaled, and that it is less effective in decoding codes when the shift registers are not of equal size. Black and Meng [13] have proposed a unified approach for the scheduling of APM updates for both R=$1/n$ and $R = k/n$ decoders. The matrix frame work of their unified approach is elegant and supports architectures ranging from cascade to parallel cascade architectures. However, for $R = k/n$ decoders, the transpose units (between every two ACS units) that are required to route the data are quite complex. As a result the latency of their architecture as well as its complexity increases.

In this paper, a new architecture is presented for the implementation of long constraint length Viterbi Decoder (VD) when the number of input bits to the trellis diagram is greater than one. The encoder structure under study is based on the original convolutional encoder. The trellis diagram is analyzed hierarchically so that each level in the hierarchy can be designed independently. This decomposition makes the state transitions in a level quite regular and straight forward. Furthermore, the concepts, design procedures and the trellis diagram developed for code rate $1/n$ can be applied here. The resulting architecture is regular, achieves 100% processor and interconnection utilization, has a foldable global topology, and is very flexible. Also, the number of global interconnections between the modules is quite small - at each stage each module communicates with only two others.

The rest of the paper is organized as follows. In Section 2, the global topology of the whole system is described. Then in Sections 3, the detailed design of each level, including issues of communication, allocation and scheduling of ACS units is presented. Memory design including storage of the accumulated path metrics and survivor sequences is described briefly in Section 4. In Section 5, it is shown that this architecture has a much better than linear tradeoff between area complexity and computation time. A procedure to fold the global topology is described in Section 6. The paper is concluded in Section 7.

2 Overview of the Whole System

Let l_i be the length of the ith shift register, $1 \leq i \leq k$, of a $R = k/n$ convolutional encoder. If the input data is binary, the state label in the trellis diagram is a string of binary bits obtained by connecting each shift register in series. In fact, the state label can be divided into k blocks, with the ith block corresponding to the ith shift register. For instance, $\underbrace{b_{k,l_k} \ldots b_{k,1}}_{blockk}, \ldots, \underbrace{b_{1,l_1} \ldots b_{1,1}}_{block1}$ is a valid state label. The input to every block is connected to the LSB (least significant bit) of the corresponding shift register. A similar representation of state labels has been proposed in [12] [13]. The trellis diagram resulting from this topology can be described by the relation, $\bar{0}b_k^{l_k-1}, \ldots, \bar{0}b_1^{l_1-1} \longrightarrow b_k^{l_k-1}\bar{0}, \ldots, b_1^{l_1-1}\bar{0}$. Here $\bar{0}$ denotes the 1-bit binary combinations and $b_i^{l_i-1}$ represents the fixed portion of the ith block $b_{i,l_i-1} \ldots b_{i,1}$. The comma "," is employed as a separation between the blocks, and every state is connected to 2^k states.

If we aggregate all those states having the same bits in the kth block into one single node called the *supernode*, the trellis diagram for $R = k/n$ reduces to the diagram for $R = 1/n$. Since the connection pattern between these supernodes is the same at every stage of the trellis diagram, we refer to this pattern as *fixed*. In this case, every supernode communicates with two parent and two descendant supernodes at every stage. The communication pattern between supernodes can also vary from stage to stage with a period of l_k. We refer to such a communication pattern as *dynamic*. This pattern is accomplished by shifting circularly the bits of the kth block by one bit at each trellis stage. The result is a trellis diagram with periodic butterflies as described in [12]. For dynamic communications, every supernode communicates externally with only one parent and one descendant supernode at every stage of the trellis diagram. Note that while the interconnection between the supernodes is different for fixed and dynamic connections, the interconnections between the states within every supernode are the same. Figure 1 describes the global topology of the proposed system. The routing network shown in the figure supports either fixed or dynamic communication pattern.

3 Internal Topology of the Supernodes

3.1 Processor Allocation

The connectivity between the states in the supernodes can be described by $\bar{0}b_m^{l_m-1},\ldots,\bar{0}b_1^{l_1-1} \longrightarrow b_m^{l_m-1}\bar{0},\ldots,b_1^{l_1-1}\bar{0}$, where $m = k-1$. This is very similar to the equation in Section 2 with the exclusion of the string of binary bits from the kth block. We group those states whose state labels have the same $(l_i - j_i)$ MSBs in the ith block, and assign an ACS unit to this group of states, $0 \le j_i \le l_i - 1$ and $1 \le i \le m$. Thus, every ACS unit updates the APMs of $2^{j_m+\cdots+j_1}$ states of a group. The ACS unit label consist of m blocks, where the ith block consists of $l_i^{'} = l_i - j_i$ MSBs of the ith block of the state label. The notation used is $ACS(b_m^{l_m^{'}},\ldots,b_1^{l_1^{'}})$, where $b_i^{l_i^{'}}$ denotes a string of bits $b_{i,l_i}\ldots b_{i,(j_i+1)}$. The parameters $j_1,\ldots,j_m$ are determined by the design constraints of the VD.

Consider the following relation between the ACS units,

$$ACS(\bar{0}b_m^{l_m^{'}-1},\ldots,\bar{0}b_1^{l_1^{'}-1}) \longrightarrow ACS(b_m^{l_m^{'}-1}\bar{0},\ldots,b_1^{l_1^{'}-1}\bar{0}). \tag{1}$$

From eqn(1), we conclude that the communication patterns between the states and between the ACS units are the same. Every ACS unit has to communicate with 2^k other ACS units equally distributed between two destination supernodes (2^{k-1} ACS units in each supernode). At the nth stage (t=nT), each state receives 2^k APMs, one from each parent ACS unit. The 2^k parent ACS units are equally distributed between two parent supernodes. The updated APM at the nth stage is sent to 2^k descendant states at the $(n+1)$th stage. The 2^k descendant states are equally distributed between two descendant ACS units in two descendant supernodes. For the case of dynamic communications between supernodes, every ACS unit needs to communicate only with 2^m parent ACS units in a parent supernode and 2^m descendant ACS units in a descendant supernode. Each ACS unit has a memory to store the APMs transferred from the parent ACS units.

To help explain the design procedures of this hierarchical VD, we use the following example throughout the rest of the paper.

Example: The decoder parameters are $k = 3$, $l_1 = 5, l_2 = 5, l_3 = 2, j_1 = 3$, and $j_2 = 2$. Consider the state 00100,10010 which is updated by $ACS(001, 10)$. The allocations of ACS units to related parent and descendant states are illustrated in Figure 2. □

3.2 Processor Clustering and Interconnections

Since the interconnection between the ACS units is quite costly, we propose a second aggregation phase similar to [14] to increase the overall efficiency of the communication links to 100%. Define the set of ACS units on the left-hand side of eqn (1) as the departure equivalence ACS units or DEACS. The DEACS label is generated from the fixed portion of each block in the ACS unit label. For example, $DEACS(b_m^{l_m'-1},\ldots,b_1^{l_1'-1})$ is generated from $ACS(\bar{0}b_m^{l_m'-1},\ldots,\bar{0}b_1^{l_1'-1})$. Thus the number of ACS units included in a DEACS is 2^m. These ACS units share 2^m global links that are connected to 2^m descendant ACS units distributed among 2^m DEACSs in one descendant supernode. Similarly, each of these 2^m ACS units is connected to 2^m parent ACS units included in one DEACS inside a parent supernode. These parent ACS units share one of 2^m global links to route the updated APMs. Thus, every DEACS has 2^m global links with one descendant supernode and 2^m global links with one parent supernode. A switching network is associated with each DEACS to route the APMs without any conflicts.

Example: Consider the DEACS labeled $DEACS(01,0)$ which includes ACS(001,00), ACS(001,10), ACS(101,00), and ACS(101,10). To make the notations simpler, we index $ACS(b_{m,l_m}b_m^{l_m'-1},\ldots,b_{1,l_1}b_1^{l_1'-1})$ by ACS_i, where $i = dec(b_{m,l_m}\ldots b_{2,l_2}b_{1,l_1})$ and $dec(b_m\ldots b_2b_1)$ is defined as the decimal value of the string of binary bits $b_m\ldots b_2b_1$. Thus $ACS(001,00)$, ACS(001,10), ACS(101,00), and ACS(101,10) are indexed as ACS_0, ACS_1, ACS_2 and ACS_3 respectively. Each of these four ACS units is connected to the following four descendant ACS units: (i) ACS_0 in $DEACS(10,0)$, (ii) ACS_0 in $DEACS(10,1)$, (iii) ACS_0 in $DEACS(11,0)$, and (iv) ACS_0 in $DEACS(11,1)$.

The parent ACS units connected to ACS_0 in $DEACS(01,0)$ are the four ACS units in $DEACS(00,0)$. Similarly, the parent ACS units connected to ACS_1, ACS_2, and ACS_3 in $DEACS(01,0)$ are the ACS units in $DEACS(00,1)$, $DEACS(10,0)$, and $DEACS(10,1)$, respectively. This communication pattern is illustrated in Figure 3. □

So far we have described the following four levels in the hierarchical design of the VD. (i) The highest level is the supernode level with 2^{l_k} supernodes. (ii) The second level is the DEACS level with $2^{l_m'+\cdots+l_1'}/2^m$ DEACSs in each supernode. (iii) The third level is the ACS unit level with 2^m ACS units in each DEACS. (iv) The lowest level is the state level. Each ACS unit updates $2^{j_m+\cdots+j_1}$ states.

3.3 Processor Scheduling

Every ACS unit updates the APMs of $2^{j_m+\cdots+j_1}$ states sequentially. Consider the ACS units in $DEACS(b_m^{l_m'-1},\ldots,b_1^{l_1'-1})$. The ACS unit labeled $ACS(b_{m,l_m}b_m^{l_m'-1},\ldots,b_{1,l_1}b_1^{l_1'-1})$ communicates with one distinct descendant ACS unit for time $T/2^m$ also denoted as a time interval (T: input data period). At the ith time interval, $i=0,1,\ldots,2^m-1$, it communicates with $ACS(b_m^{l_m'-1}b_{m,j_m},\ldots,b_1^{l_1'-1}b_{1,j_1})$, where $dec(b_{m,j_m}\ldots b_{1,j_1}) = [dec(b_{m,l_m}\ldots b_{1,l_1})+i] \bmod 2^m$. For instance, ACS_2 in $DEACS(01,0)$ communicates with ACS_0 in $DEACS(11,0)$ at interval 0, with ACS_0 in $DEACS(11,1)$ at interval 1 and so on (refer to Figure 3).

In each time interval, an ACS unit updates the APMs of $2^{j_m+\cdots+j_1}/2^m$ states sequentially according to the increasing order of the number, $dec(b_m^{j_m-1}\ldots b_2^{j_2-1}b_1^{j_1-1})$, where $b_i^{j_i-1}$ denotes the string $b_{i,(j_i-1)}\ldots b_{i,1}$. Such a scheduling can easily be implemented using registers and some control logic, and do not require general-purpose routing networks.

Example: The ACS unit, $ACS(001,00)$ communicates with two descendant ACS units in two descendant supernodes at a time (totally, 8 descendant ACS units in two descendant supernodes) and updates the APMs of 32 ($=2^{j_1+j_2}$) states. Thus, 8 states are updated sequentially during an interval of $T/2^m=T/4$. The schedule is shown in Figure 4. □

4 Memory Management

In this section we briefly describe the memory required to store the APMs in-place and the survivor sequences. The memory to store the APMs is distributed among the ACS units and is controlled locally, while the survivor sequences are stored in a single survivor memory.

4.1 In-place Storage of the Accumulated Path Metrics (APM)

To update the APMs of the states, an ACS unit has to read a set of parent APMs from its local memory in an order dictated by the update schedule. At the same time, the ACS unit receives from its parent ACS units a set of newly updated parent APMs for computations in the next cycle. Figure 5 describes the time steps at which the APMs of 64 parent states required by the update schedule shown in Figure 4, are generated (updated). Every state label in Figure 5 represents two parent states from two supernodes which are updated at the same time. The notation i_j represents the jth time step of the ith interval. There are only two kinds of access for the stored APMs as shown in Figure 6. The memory scheme can be generalized for the case when $j_1 \geq 2$ and $j_n = 2$ for $n = 2, \ldots, m$. Since each ACS unit updates $2^{j_m+\cdots+j_1}$ states, its local memory contains $2^{2(m-1)+j_1}$ cells, where each cell stores 2 parent APMS that are fetched sequentially. The first m address bits of a cell represent the block (out of 2^m blocks in the memory) that it belongs to, and the remaining $m + j_1 - 2$ bits index the cell in the block. The address generator needs to support only two kinds of memory access patterns. The addresses of a parent APM in the two access patterns are related by $A^m B^m C^{j_1-2} \longrightarrow B^m A^m C^{j_1-2}$, where A^m or B^m represents a block of m bits and C^{j_1-2} represents a block of $j_1 - 2$ bits. The structure of the data buffer unit for this case is shown in Figure 7. It consists of 2 blocks with 2^{j_1-2} memory banks in each block. The unit is pipelined in order that the ACS computations for one interval can be overlapped with data fetch for computations of the next interval. 2^{m-1} iterations are required to finish updating all the states in an interval. In each iteration, the 2^{j_1-2} banks in block 1 are accessed one after the other, each access lasting 2 time steps. Assume that the ACS unit, $ACS(b_{m,l_m} b_m^{l_m'-1}, \ldots, b_{1,l_1} b_1^{l_1'-1})$ is updating the states in the ith interval. Here a_i is equal to $2^{j_1-2} * \{[dec(b_{m,l_m} \ldots b_{1,l_1}) + i] \bmod 2^m\}$. The parent APMs for the computations of the $(i+1)$th interval are loaded in cyclic mode; first into the 0th position of each of the 2^{j_1-2} banks in block 2, then into the 1st position of each of the banks, and so on. For other sets of j_is values, the memory accesses are not as simple and the in-place storage cannot always be utilized. This is not really restrictive, since the values of j_is are chosen by the designer. In fact, the performance of the VD is affected by the value of $j_m + \cdots + j_1$, and not on the individual values of j_is. Consider an example where $k = 4$, and the lengths of the shift registers are 4, 3, 3, and 2. Let $j_1 + j_2 + j_3 = 7$ from performance considerations. Then for efficient APM memory desig, we choose $l_4 = 2$, $l_1 = 4$, $l_2 = 3$, $l_3 = 3$ and $j_1 = 3$, $j_2 = 2$, $j_3 = 2$.

4.2 Survivor Memory Management

In our design of the VD, the time required to complete one global cycle is $2^{j_m+\cdots+j_1}$ clock cycles. This is usually long enough to perform the trace back operation. Hence, the throughput is not actually limited by the rate at which the survivor sequence can be decoded. The classical one or K-pointer trace-back technique can be easily employed here, the specific choice depending on the value of $j_m + \cdots + j_1$. Thus, the structure of the survivor memory used in our design is similar to the one proposed in [15].

5 Tradeoffs between Area and Computation Time

In this section we compare the purely state-parallel VD, where an ACS unit is assigned to every state, with the proposed design, where one ACS unit is assigned to $2^{j_m+\cdots+j_1}$ states. Let the propagation delay (with storage and communication time included) of an ACS unit be T_d and let the data path be pipelined to N levels. If the delay of each pipelining latch is t_l, then the delay per stage of the pipelined ACS unit is $\frac{T_d+N*t_l}{N} = T_c$, where T_c is set equal to the clock period (i.e. one time step) and $T_c < T_d$. Figure 8 compares the area and time complexities of the purely state parallel implementation with the proposed architecture. The global cycle time of our architecture is increased by a factor of $\frac{(2^{j_m+\cdots+j_1}+N)*T_c}{T_d} = (1 + \frac{2^{j_m+\cdots+j_1}}{N}) * \frac{1}{1-t_l/T_c}$, while the number of ACS units and the number of global communication links are both reduced by a factor of $2^{j_m+\cdots+j_1}$. If we neglect the routing area and assume the memory requirement to be invariant, then the tradeoff between the hardware complexity and the execution time is given by

$$(1+\frac{2^{j_m+\cdots+j_1}}{N}) * \frac{1}{1-t_l/T_c} * \frac{1}{2^{j_m+\cdots+j_1}} \approx \frac{1}{N} * \frac{1}{1-t_l/T_c} = \frac{T_c}{T_d} < 1$$

Since the number of global communication links is also reduced by a factor of $2^{j_m+\cdots+j_1}$, we conclude that the proposed architecture has a better than linear tradeoff between hardware complexity and speed. The factor $2^{j_m+\cdots+j_1}$ is determined by the value of $j_m + \cdots + j_1$, and so the individual values of $j_m, \ldots, j_1$ cannot directly affect the performance of the VD. This enables us to choose suitable values for $j_m, \ldots, j_1$ to obtain different ACS unit allocations or efficient APM memory access (see Section 4.1), and still maintain the same performance improvement.

6 Folding the Global Topology

In the analysis so far, the number of supernodes is 2^{l_k} since the kth block has been chosen at the highest level. While any of the blocks could have been chosen at the highest level, if the area constraints demanded 2^l supernodes, where $l < l_i \ \forall i$, then the global topology would have to be folded. Let a *hypernode* be defined as a collection of supernodes which have the same $l' = l_k - j_k$ MSBs. Thus each hypernode consists of 2^{j_k} supernodes and the global topology consists of $2^{l'}$ hypernodes. All the hypernodes are processed in parallel and each hypernode sequentially processes the supernodes assigned to it. The scheduling and interconnections between the ACS units in a hypernode follow the same rules as discussed in Section 3.

7 Conclusions

In this paper we presented a novel architecture to implement long constraint length Viterbi decoder for the case when the number of input bits to the trellis diagram is more than one. The architecture has been designed in a hierarchical fashion by breaking the system into several levels, and designing each level independently. The resulting architecture is regular, supports folding and achieves better than linear tradeoff between hardware complexity and computation time. Another notable feature of this architecture is its flexibility. Different architectures can be obtained for different parameter choices (j_is), and yet all the architectures achieve the same performance improvement as long as the value of $(j_k + \ldots + j_1)$ remains fixed.

References

[1] G. D. Forney,Jr., "The Viterbi Algorithm," *Proceedings IEEE*, pp. 268-278, March 1973.

[2] S.Lin and D.J.Costello,Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, 1983.

[3] P.G.Gulak and E.Shwedyk,"VLSI structures for Viterbi receivers: Part I-General theory and applications," *IEEE J. Select. Areas Commun.*, vol. 4, pp 142-154, Jan. 1986.

[4] P.G.Gulak and T.Kailath, " Locally Connected VLSI Architectures for the Viterbi Algorithm," *IEEE Jour. on Selected Areas in Communications*, pp. 527-537, April 1988.

[5] H.D.Lin, C.B.Shung, and D.G. Messerschmitt, "Folded Viterbi Decoders for Convolutional Codes," *VLSI Signal Processing IV, New York: IEEE Press 1990*, pp. 381-391.

[6] C.B.Shung et al.,"Area-efficient architectures for the Viterbi algorithm-Part I: Theory," *IEEE Trans. Commun.*, vol. 41, no. 4, Apr. 1993.

[7] G.Feygin, P.G.Gulak, and P.Chow, "A Multiprocessor Architecture for Viterbi Decoders with Linear Speedup," *IEEE Trans. Signal Proc.*, vol. 41, no. 9, pp. 2907-2917, Sep. 1993.

[8] Jens Sparso et al.,"An area-efficient topology for VLSI implementation of Viterbi decoders and other shuffle-exchange type structures," *IEEE J. Solid-State Circuits*, vol. 26, no. 2, Feb. 1991.

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

[10] Shuji Kubota et al.,"Novel Viterbi decoder VLSI implementation and its performance," *IEEE Trans. Commun.*, vol. 41, no. 8, Aug. 1993.

[11] C.Y.Chang and K.Yao, "Systolic Array Processing of the Viterbi Algorithm," *IEEE Trans. on Information Theory*, vol. 35, no. 1, pp. 76-86, Jan 1989.

[12] H.D.Lin and C.B.Shung,"General in-place scheduling for the Viterbi Algorithm," in *Proc. ICASSP* 1991, pp.1577-1580

[13] P.J.Black and T.H.-Y.Meng,"A unified approach to the Viterbi algorithm state metric update for shift register process," in *Proc. ICASSP*, vol. 5, Mar. 1992, pp. 629-632.

[14] F.Daneshgaran, *VLSI Architectures for Parallel Implementation of Long Constraint Length Viterbi Decoders*. Ph.D. thesis, Univ. of California, Los Angeles, 1992.

[15] R.Cypher and C.B.Shung, "Generalized Trace-back Techniques for Survivor Memory Management in the Viterbi Algorithm," *Journal of VLSI Signal Processing*, vol. 5, pp.85-94, 1993.

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

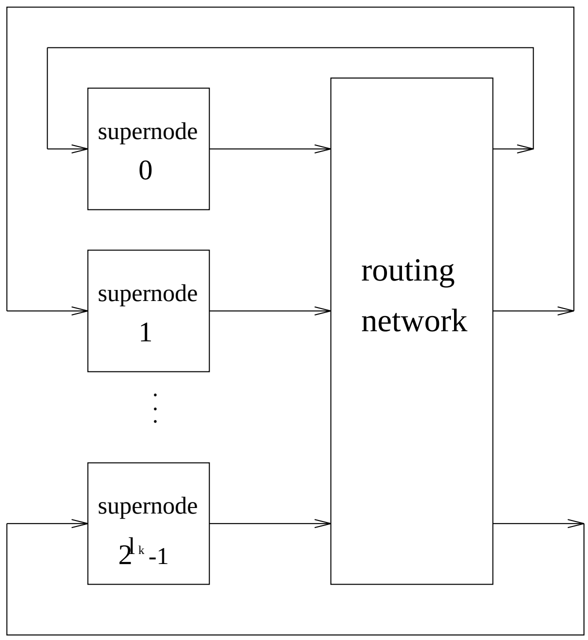

Figure 1: The architecture model for the whole system.

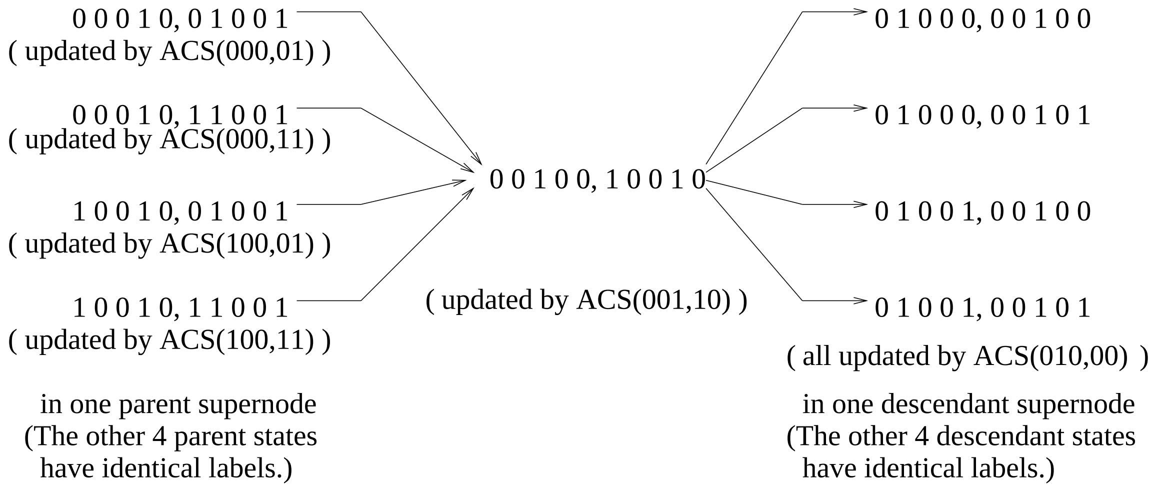

Figure 2: The allocation of ACS units to related parent and descendant states in the example.

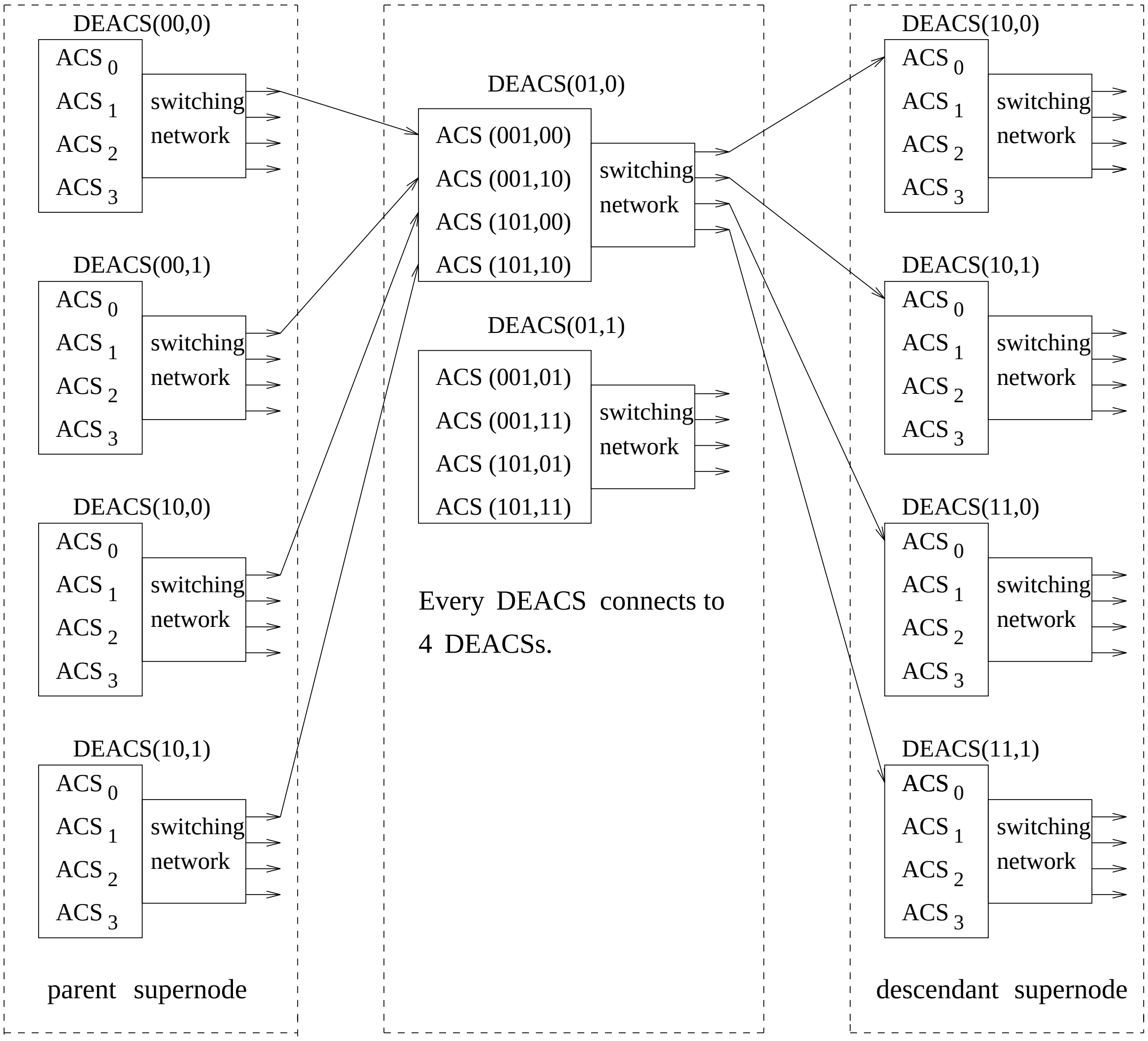

Figure 3: The clustering and connectivity between ACS units for the example where $m = 2$.

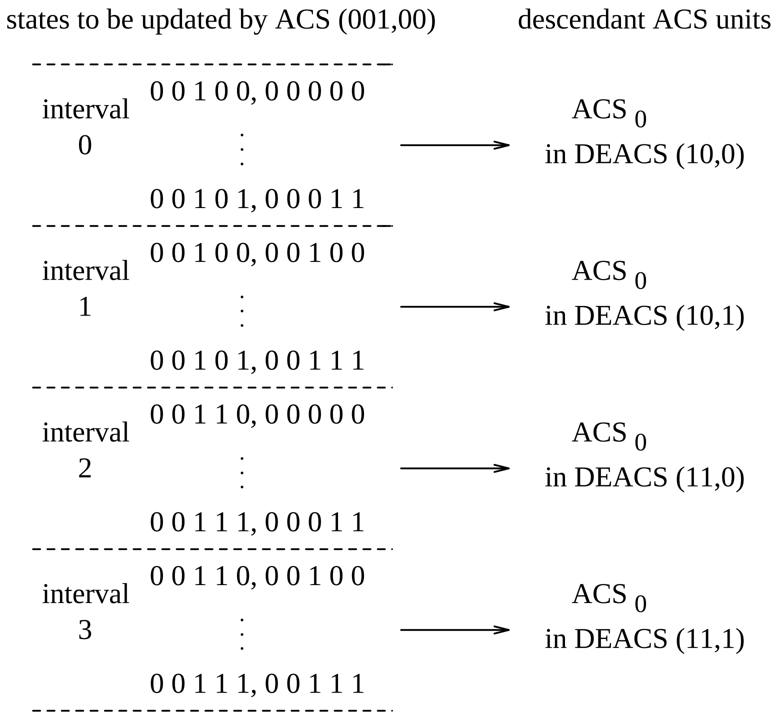

Figure 4: The update schedule for $ACS(001,00)$ in $DEACS(01,0)$ in the example.

	parent states stored in the local memory of ACS (001, 00)	updated at time step	updated by (in DEACS (00,0))
interval 0	1 0 0 1 0, 0 0 0 0 0	0_0	ACS 2
	1 0 0 1 0, 0 0 0 0 1	0_1	
	0 0 0 1 0, 1 0 0 0 0	1_0	ACS 1
	0 0 0 1 0, 1 0 0 0 1	1_1	
	0 0 0 1 0, 0 0 0 0 0	2_0	ACS 0
	0 0 0 1 0, 0 0 0 0 1	2_1	
	1 0 0 1 0, 1 0 0 0 0	3_0	ACS 3
	1 0 0 1 0, 1 0 0 0 1	3_1	
interval 1	1 0 0 1 0, 0 0 0 1 0	0_2	ACS 2
	1 0 0 1 0, 0 0 0 1 1	0_3	
	⋮	⋮	⋮
	1 0 0 1 0, 1 0 0 1 0	3_2	ACS 3
	1 0 0 1 0, 1 0 0 1 1	3_3	
interval 2	1 0 0 1 1, 0 0 0 0 0	0_4	ACS 2
	1 0 0 1 1, 0 0 0 0 1	0_5	
	⋮	⋮	⋮
	1 0 0 1 1, 1 0 0 0 0	3_4	ACS 3
	1 0 0 1 1, 1 0 0 0 1	3_5	
interval 3	1 0 0 1 1, 0 0 0 1 0	0_6	ACS 2
	1 0 0 1 1, 0 0 0 1 1	0_7	
	⋮	⋮	⋮
	1 0 0 1 1, 1 0 0 1 0	3_6	ACS 3
	1 0 0 1 1, 1 0 0 1 1	3_7	

Figure 5: The time steps at which the APMs of 64 parent states are generated (updated).

0_0 , 0_1		0_0 , 0_1	→ I	0_0 , 0_1	
1_0 , 1_1	→ I	0_2 , 0_3	→ II	1_0 , 1_1	→ I
2_0 , 2_1		0_4 , 0_5	→ III	2_0 , 2_1	
3_0 , 3_1		0_6 , 0_7	→ IV	3_0 , 3_1	
0_2 , 0_3		1_0 , 1_1	→ I	0_2 , 0_3	
1_2 , 1_3	→ II	1_2 , 1_3	→ II	1_2 , 1_3	→ II
2_2 , 2_3		1_4 , 1_5	→ III	2_2 , 2_3	
3_2 , 3_3		1_6 , 1_7	→ IV	3_2 , 3_3	
0_4 , 0_5		2_0 , 2_1	→ I	0_4 , 0_5	
1_4 , 1_5	→ III	2_2 , 2_3	→ II	1_4 , 1_5	→ III
2_4 , 2_5		2_4 , 2_5	→ III	2_4 , 2_5	
3_4 , 3_5		2_6 , 2_7	→ IV	3_4 , 3_5	
0_6 , 0_7		3_0 , 3_1	→ I	0_6 , 0_7	
1_6 , 1_7	→ IV	3_2 , 3_3	→ II	1_6 , 1_7	→ IV
2_6 , 2_7		3_4 , 3_5	→ III	2_6 , 2_7	
3_6 , 3_7		3_6 , 3_7	→ IV	3_6 , 3_7	

I : interval 0 II : interval 1 III : interval 2 IV : interval 3

Figure 6: Two kinds of access for the stored APMs in Figure 5, where the states are indexed by the time steps at which they are updated.

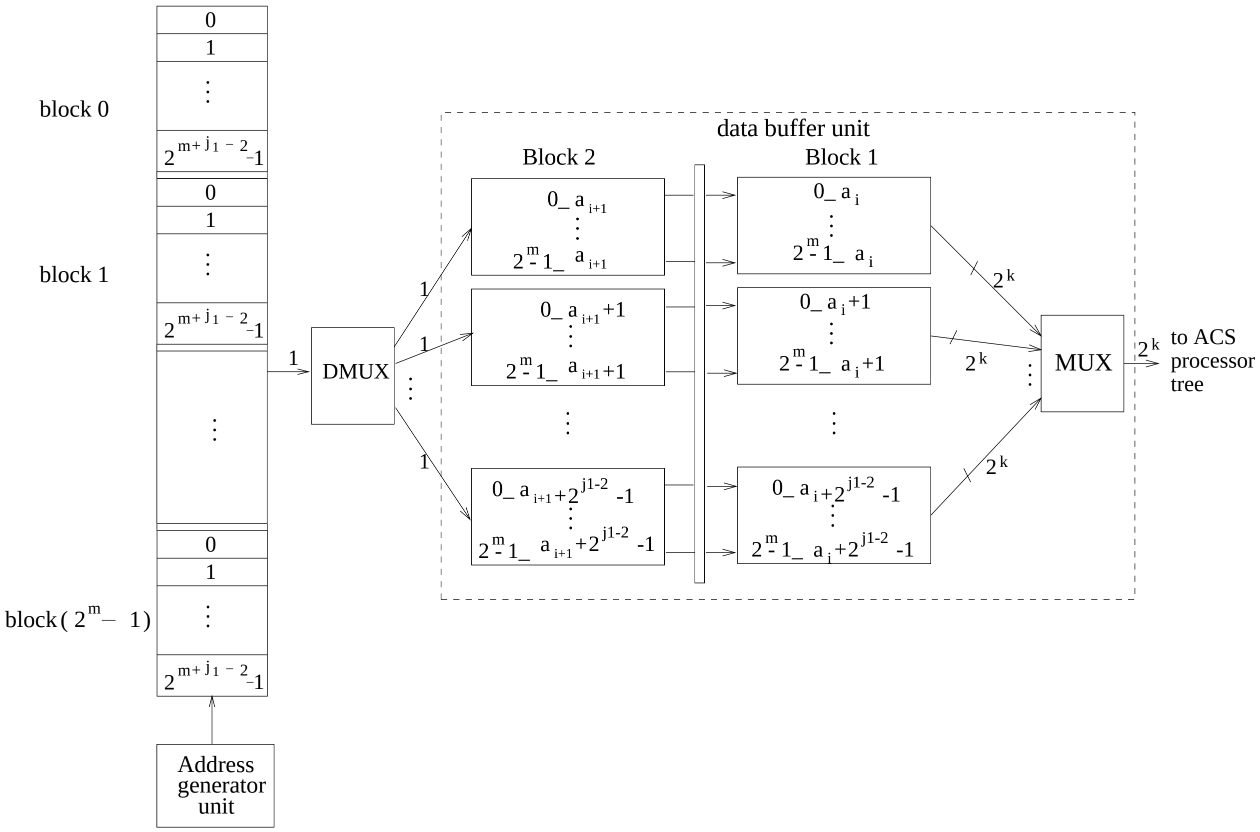

Figure 7: The memory scheme and the structure of the data buffer unit for the case when $j_1 \geq 2$ and $j_i = 2$ for $2 \leq i \leq m$. The notation i_j represents the jth memory cell of the ith block in the APM memory.

	State-parallel	Proposed architecture
Propagation delay (without pipelining)	T_d	T_d
Time to update the APM of one state	T_d	T_c
Global cycle time	T_d	$(2^{j_m+\cdots+j_1} + N) * T_c$
Complexity of an ACS unit	2^k	2^k
Number of ACS units in a supernode	$2^{l_m+\cdots+l_1}$	$2^{l'_m+\cdots+l'_1}$
Number of global commun. links for		
fixed communication	$2^{l_m+\cdots+l_1} * (2^{l_k} - 1) * 2$	$2^{l'_m+\cdots+l'_1} * (2^{l_k} - 1) * 2$
dynamic communication	$2^{l_m+\cdots+l_1} * 2^{l_k} * l_k$	$2^{l'_m+\cdots+l'_1} * 2^{l_k} * l_k$
Memory size requirements: APM memory and survivor memory	The APM memory in an ACS unit reduces if the number of ACS units increases but the total size remains unaffected and is fixed by the sum of $l_1, \ldots, l_k$.	
	The survivor memory is the same for both cases.	

Figure 8: Comparison of area and time complexities of the state-parallel VD and the proposed architecture. The complexity of an ACS unit is in terms of the number of inputs.