Low Power Architecture of the Soft-Output Viterbi Algorithm

David Garrett and Mircea Stan
University of Virginia - Department of Electrical Engineering
Charlottesville, VA 22903
garrett@virginia.edu   mircea@virginia.edu

1 ABSTRACT

This paper investigates the low power implementation issues of the soft-output Viterbi algorithm (SOVA), a building block for turbo codes. By briefly explaining the theory of turbo codes, and by reviewing several of the decoding algorithms, we develop the computational requirements for a SOVA implementation, and ultimately develop an architecture that completes those computations with reduced power consumption. The architecture builds on previous work on the Viterbi and Soft-Output Viterbi algorithms, and incorporates a novel orthogonal access memory structure, which provides parallel access across sequentially received data.

1.1 Keywords

SOVA, turbo codes, VA, low power.

2 INTRODUCTION

Turbo codes were introduced by Berrou et. al. in 1993 [1], and represent a significant advance in the communications field by closely approaching Shannon's channel capacity limit. The algorithms have performed with bit error rates (BER) of 10^{-5} at signal-to-noise ratios (E_b/N_0) down to 0.7 dB in simulations [2]. By comparison, the E_b/N_0 required for a commonly used 64-state convolution code decoded with the Viterbi algorithm is 4.5 dB for a 10^{-5} BER [3].

When looking at communication system design, turbo coding can have an immediate impact on the power budget, because these codes can lower the transmit power requirements by several dB. For mobile communications, turbo coding will become important in maintaining battery life by reducing the transmit power budget, at the expense of more complex decoding algorithms.

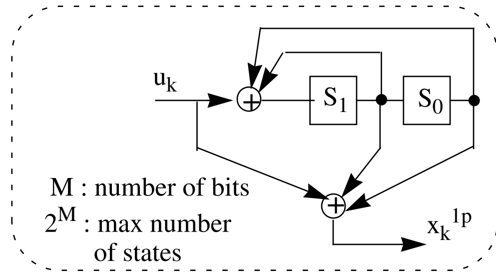

Fig. 1. Recursive Convolutional Encoder

While the turbo coding algorithm itself is ideal for low power communication systems, there is the additional challenge of designing a coder/decoder (CODEC) that uses minimal power. While there have been several implementations of turbo decoders [4][5], none of them has addressed low power as a driving factor in the realization of the algorithms (the implementation in [4] runs at 5V and consumes 1.7 watts at 40MHz). Yet in the near future, personal communications systems (PCS) will communicate directly with satellites and the lowest E_b/N_0 performance coupled with a Low-Power design are of the utmost importance in maintaining service life for these applications.

To build a low power system such as this requires a low power design methodology at all levels of abstraction. From the optimal selection of low-capacitance primitive cells, to setting an ultra-low supply voltages, intelligent clock management, and all the way up to architectural issues like data representation or parallelism. Considerable work has been done in all these levels of low power design, and for this paper we will focus on the architecture of the decoder for reducing switching activity with the assumption that a final implementation will have these and other low power techniques incorporated.

This paper is partitioned into three sections:

- a brief introduction into Turbo coding theory,
- a review of "soft-output" decoding architectures,
- a proposed architecture for a low power SOVA decoder.

3 TURBO CODES

3.1 Turbo Encoding

The basic building block for turbo encoders, as for any convolutional code, is a recursive convolutional encoder as seen in Fig. 1. At each instance of time, the particular symbol information, u_k, will generate an associated parity bit, x_k^p, according to the current state of the encoder, $\{s_1 s_0\}$. Turbo

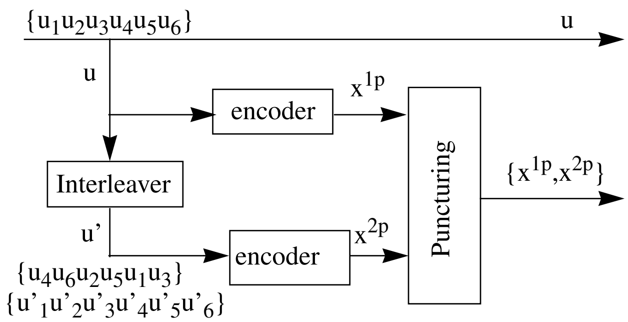

Fig. 2. Turbo Encoder

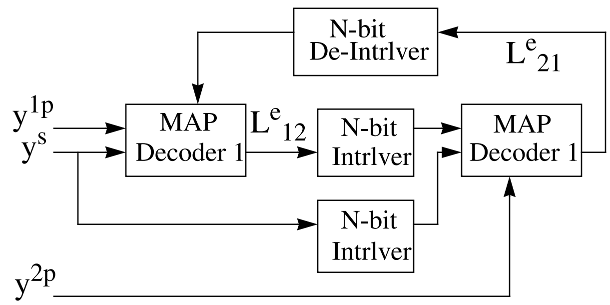

Fig. 3. Turbo Decoder

codes piece together two of these encoders operating in parallel on blocks of data, with the second encoder operating on the data after it has been re-ordered with an interleaver. The BER performance of turbo codes has been shown to be related to the interleaver block size, N [6] which leads to relatively large blocks sizes for Turbo code applications, on the order of at least 1000 symbols. Fig. 2 shows a turbo encoder with example symbol data fed into each encoder, the second encoder operating on interleaved data from this first encoder (block size of 6 for illustration purposes only). Turbo codes are not simply redundant coding techniques like inner and outer codes, but are actually a way of allowing the two parallel encoders to work together.

3.2 Turbo Decoding

Because Turbo codes use two encoders on differently ordered blocks of data, recovering the code directly from the received codewords is difficult. Berrou's breakthrough was combining the parallel concatenated convolution codes (PCCC) with a two-stage iterative decoder block (Fig. 3). The first stage performs a sequence detection using the received symbol and parity bit information from the first encoder, and calculates the maximum likelihood (ML) sequence of encoder states. From this sequence, the original symbol stream could be recovered with a deterministic function based on the encoder structure, but the important part of turbo codes is that the first stage also passes along "soft" information, L^e, that specifies the confidence in that decision. The second stage performs the same ML detection on the interleaved data, but it now has the "extrinsic" data to help skew the results toward the ML path from the first stage. If a symbol has been detected with high confidence in the first stage, the second stage is predisposed to make the same decision.

Consider the case where the first stage has a low confidence and an incorrect symbol decision, and the second stage has a high confidence with the correct symbol. The second stage, with its high confidence, will reverse the first symbol decision, and since its "soft" information is passed back to the first stage, the iterative approach will converge to the ML path and the correct bit decisions. Simulations have shown that the Turbo code performance is not very high after the first iteration, but after several stages it converges to the low BER [7].

3.3 Sequence Detector Algorithms

The original Turbo code decoding algorithm was based on the maximum a posteriori (MAP) detector which is the optimal algorithm for minimizing bit error decisions [8]. The MAP algorithm is very costly because during the traceback steps to find the competing paths, it evaluates all possible paths. The Max-Log-MAP modification takes the MAP algorithm and converts the calculations into the logarithmic domain, removing the need for multiplication operations.

As an alternative to the MAP algorithms, Hagenauer proposed the Soft-Output Viterbi Algorithm (SOVA) [9], a modification of the Viterbi algorithm (VA) that also provides confidence information with each decision bit. Although it is less than optimal for bit error detection as compared with the MAP algorithm, simulations with SOVA show only a 0.7 dB increase in E_b/N_0 to maintain the same BER as MAP (interleaver size of 1024) [9]. SOVA reduces complexity of the decoding problem by eliminating unlikely paths, and while it does find the same hard decision on the symbols as the MAP [10], it tends to overestimate the confidence of a symbol detection.

For the sake of low-power and for circuit area, SOVA is the algorithm of choice for implementation, and the remainder of this paper will evaluate this algorithm.

4 SOVA

4.1 Measure of Maximum Likelihood

The original Viterbi algorithm is the optimal choice when trying to match an incoming sequence of data and parity bits to the original states of an encoder [11]. Once the highest likelihood path of encoder states has been calculated, the exact input data sequence can be reconstructed through traceback information.

The Viterbi decoder uses a "butterfly" state decomposition, shown in Fig. 4 for the 2-bit encoder from Fig. 1. With binary symbols, each currrent state can only be reached from two previous states, and in the case of a recursive convolutional encoder, each branch comes from a different symbol decision, one is a binary '1', and the other is a binary '0'. The Viterbi algorithm determines maximum likelihood paths by evaluating the two paths into a given state, and determining the one with the highest path metric (PM). The updated path metric for each state is stored along the decision bit used to reach that state. Each of the path decisions are based on matching the incoming codeword with the transition probabilities, and each new state represents the receipt of one

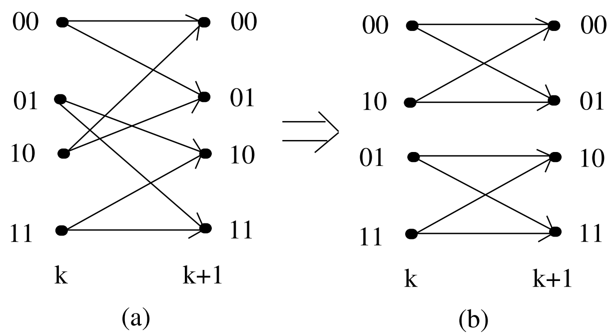

Fig. 4. Next-state trellis of 2-bit encoder

more codeword (one symbol and one parity bit).

4.2 Traceback

The path metric calculations just assigned the measurement functions to each state, but the actual Viterbi decisions on encoder states is based on a *traceback* operation to find the path of states. Fig. 5 shows an example of the possible traceback paths from time k, and each state follows backwards through its paths based on the stored decision bit. The important characteristic is that if every state from a current time is followed backwards through its maximum likelihood path, all of the paths converge at a point somewhere previous in time. This is how traceback decisively determines the state of the encoder at a given time, by showing that there is no better choice for an encoder state given the global maximum likelihood path. In Fig. 5, all of the paths have converged at time j, and the algorithm can actually release that state, along with its decision bit, as the decoded sequence.

4.3 Soft Information

SOVA extends the Viterbi algorithm with confidence information by looking at the difference of incoming paths to a state as a measure of "correctness" for that decision [9]. Whereas the Viterbi algorithm traces back over one path, SOVA traces backwards over the maximum likelihoodn (ML) path and its next competitor (if the ML approaches a state with a '1' input, the competitor traces back the '0' path). The traceback operation takes the measure of likelihood at the starting state, and updates the bits along that path with the minimum of the pathmetric difference at the start of the traceback or its current value, but only along the paths where the ML and competitor paths differ in bit decisions. The idea is that all the decisions bits along the traceback path are based on correctly choosing the ML path, and if the competitor path has a different decision, it can only be as confident as the decision to choose the competitor over the ML path. One problem with SOVA in its original form is that it had to traceback from each of the 2^m states because at that point the global ML path was not known..

Another method to reduce the complexity of the traceback operation was shown in [12]. The solution was to look forward in time in order to find the ML path with the regular VA. This reduces the number of SOVA tracebacks to just one from the ML state instead of from all 2^m states. In addition it was found that the SOVA traceback length could be reduced

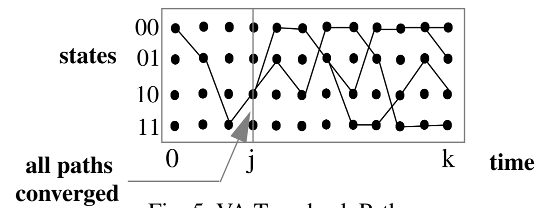

Fig. 5. VA Traceback Paths

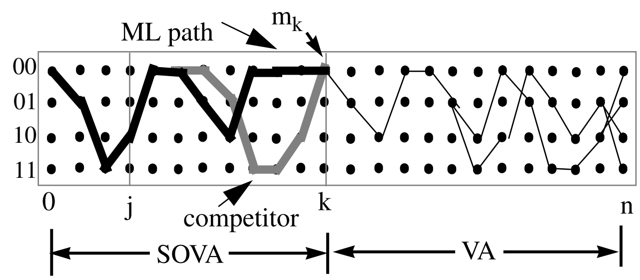

Fig. 6. SOVA combined with Viterbi

in half with a minimal degradation in performance. Fig. 6 shows an example trellis and how the VA and SOVA algorithms work together. The decoding proceeds forward calculating the path metrics of each state, but at time k where the SOVA would normally traceback for each of the 2^M possible states to determine the minimum reliability, the Viterbi algorithm can be used to search further forward in time. With a traceback algorithm, a single state, m_k, can be selected for the SOVA reliability traceback.

4.4 VA Systolic Array

The main bottleneck in implementing SOVA is that for every new piece of symbol information, the algorithm traces back over multiple states. A particularly clever solution to the traceback problem for the Viterbi algorithm was shown by using a systolic array [13]. At the beginning of the array, the butterfly matrix calculates the next states as a function of the current path metrics of all states and the most recent symbol and parity data. The paths to each state in the trellis imply the decision bits that were used to reach those states, and that information is also explicitly computed. The ML state is selected from the maximum path metrics of all the current states, and is passed along with the decision vector. The max state represents the start point for the traceback through to the encoder trellis. This structure is represented in Fig. 7.

The systolic array keeps the high throughput for the traceback operations by launching a new traceback into the pipeline on every cycle. The pipeline has two components, propagating through the stages:

- the traceback states
- the decision vectors

While the decision vectors move one step ahead on every cycle, the ML state stages actually jump ahead two steps at a time. The beginning of the pipeline is analogous to time k in Fig. 5, in that all the possible stages have a path to a previous state as defined in the decision bit vector. The state register at the beginning of the pipeline represents the state the traceback will occur from, much like state 00 in Fig. 5. On the

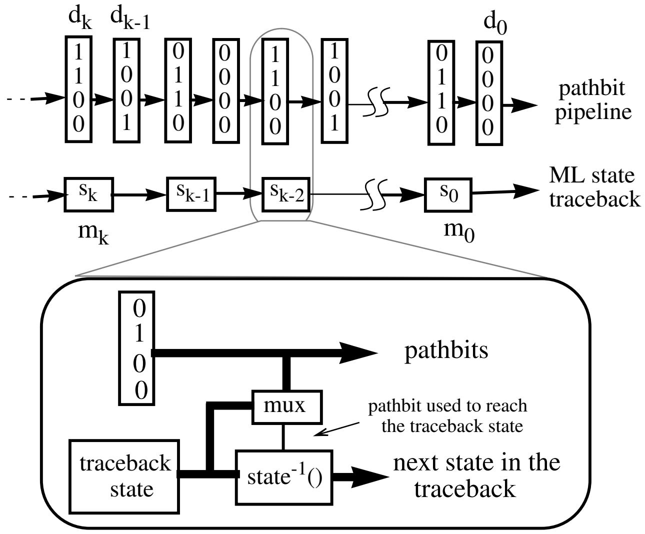

Fig. 7. Traceback pipeline

next clock cycle, the k^{th} state decision vector moves one stage, but the ML traceback state jumps forward to the decision vector for time (k-1). Again the previous state is calculated and as the state travels into the pipeline, it moves closer to the final traceback time. The pipeline is sized such that the traceback state will reach the final state where a symbol decision is made (analogous to the traceback of the ML path in Fig. 5 from time k to time 0) at the end of the pipeline, and at that point the decision vector bit can be released as the true decoded symbol.

4.5 High-Throughput Reliability Traceback

The systolic array is a powerful way to implement a high-speed Viterbi algorithm but the above description just handles the traceback of states. To extend the systolic traceback to SOVA presents some complicated design issues:

- SOVA requires path metrics differences for every state,
- traceback must occur on two paths (survivor and competitor),
- each state must have access to all information about the path metric differences and decision vectors) for that particular time.

Inherently the systolic structure of the Viterbi algorithm implementation is a “power hog” because every stage in the pipeline requires data movement on every cycle. With the addition of the path metrics for every state, the memory size for SOVA grows significantly, as does the power consumption. In order to handle the tracebacks, the structure requires parallel access to all the states simultaneously, yet new information only comes at the head of the pipeline. The following sections will show a low power traceback structure for SOVA that maintains a high throughput structure, but reduces power consumption by intelligent management of metric data.

5 LOW POWER MANAGEMENT OF DATA

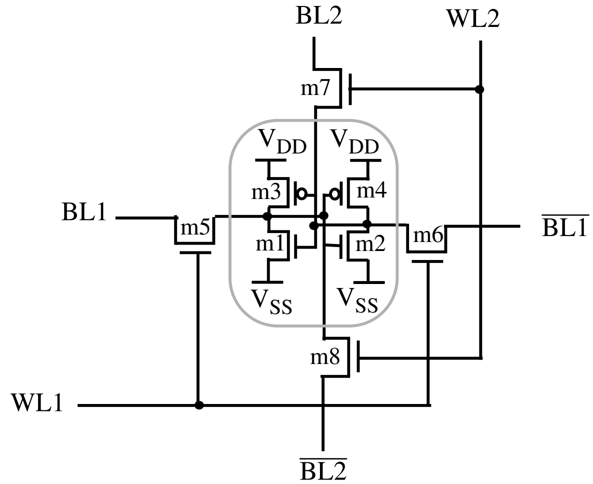

Fig. 8. Orthogonal-access SRAM

5.1 Orthogonal Memory Access

A structure for realizing the SOVA algorithm in hardware requires the storage of large amounts of data, and it also requires access to blocks of data across multiple time windows. The ideal structure is memory with orthogonal access in which the data is written in sequential order, and in a single step, information across all the stages can be read in parallel. Such a memory, where data can be written in by rows, and read out by rows or columns ensures a high throughput of the traceback, and also eliminates the wasteful data movement in the processing of SOVA.

An implementation of an orthogonal random access memory cell can be built with a slight modification to the traditional six transistor SRAM cell. With the addition of a second set of access transistors as shown in Fig. 8, intelligent control of the word lines creates an orthogonal, *write by rows read by columns* memory.

5.2 SOVA Traceback Pipeline

The structure of the low power SOVA (LP-SOVA) architecture is similar to the systolic array for the Viterbi algorithm, but it combines the orthogonal memory with two traceback pipelines with path metric difference calculations.

As with the Viterbi systolic array, the add-compare-select logic is used in LP-SOVA to generate next state path metrics and the decision vectors, but instead of launching the values into a pipeline, they are written directly into the memory array. An auto-increment pointer is used so that successive trellis states are stored sequentially. Fig. 9 shows the organization of the memory array, with the decision bits for each state ($d^{[state]}$) and the path metrics ($PM^{[state]}$). Written in by columns, the orthogonal memory allows the entire set of decision bits for a particular state to be retrieved in one cycle. With a traditional memory architecture each traceback operation would require k memory reads for a single traceback step.

In the Viterbi systolic traceback pipeline, the traceback states have a stride of 2 with respect to the decision vectors, but with the new storage arrangement, the traceback states are

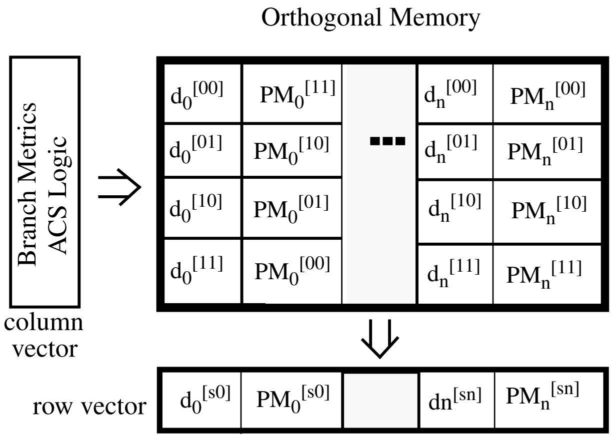

Fig. 9. SOVA on-chip memory structure

matched with the rows of the orthogonal memory. With the memory, the decisions vectors are static, and the traceback states only have to move one step each cycle.

The pipeline is arranged so that as the updated information is written into the orthogonal memory, a new traceback is spawned in lower pipeline. Similar to the systolic array, the previous state calculation is based on the current state as well as the decision vector in the memory. Instead of hardwiring the connections, the decision vectors are retrieved with a column read.

5.3 LP-SOVA Solution

All of the components, the ACS logic, the orthogonal memory, and the two traceback pipelines can be pieced together to create the LP-SOVA decoding solution. Fig. 10 shows the block diagram of the decoder. The three main pipelines contained are the maximum likelihood path, the competitor path, and the reliability information pipeline.

In terms of Low Power this design offers many potentials gains over a systolic array implementation.

- Of the utmost importance, the large block of path metric decisions vectors are static, maintaining their position in the memory for the duration of the decode operation.
- the data in memory does not need to move in a stride of 2 as with the systolic array.
- the decision vectors are recovered conditionally, eliminating wasteful memory reads.

In order to effectively evaluate the LP-SOVA architecture, it must be compared against the systolic architecture proposed in [12]. As explained in [12], the traceback operation can be partitioned into a VA traceback and a SOVA traceback. This previous SOVA architecture uses hardware to calculate the VA (which then discards the path metric calculations), and then starts a dedicated SOVA traceback in a separate hardware block, recalculating the path metric differences. In terms of power consumption, this is wasteful for two reasons:

- it performs the exact same calculation twice
- it requires that the incoming sample data be buffered in a shift register chain.

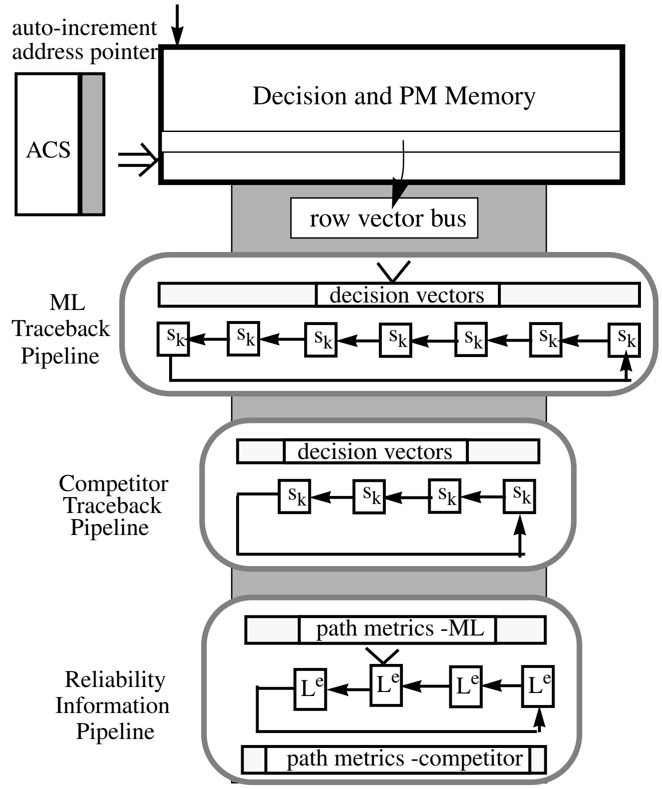

Fig. 10. LP-SOVA Architecture

The key to LP-SOVA is that it calculates the path metrics in the beginning, stores them, and then does not move them from that memory location. While this does increase storage, some of that storage is recovered in eliminating the stride 2 systolic array. Fig. 11 shows the structure of data storage for the LP-SOVA and the systolic array. The actual traceback pipelines are very similar for both implementations as they require the same information at the same rates so the main difference is in the data storage. An estimate of the power consumed in the data storage area can be measured by looking at the actively moving data in any one clock cycle. For the systolic array SOVA implementation, there are moving samples for storing the data, propagating the pathbits for the VA traceback, and propagating the pathbits along with pathmetric differences for the SOVA traceback. We can represent the storage requirements for SOVA with the following variables: *j* is the depth of the Viterbi traceback, *k* is the depth of the SOVA traceback, *m* is the number of encoder bits, *d* is the A/D precision, and *e* is the precision of the pathmetric differences. Equation (1) shows the storage requirements for the systolic array SOVA. In constrast, the LP-SOVA solution has storage for the pathbits and pathmetric differences for the both traceback stages (2). The key difference for power comes from the fact that LP-SOVA maintains mosts of the data statically while the systolic nature of the regular SOVA forces every storage location to be updated on every cycle. Equations (3) and (4) show the ratioes of active to unchanged

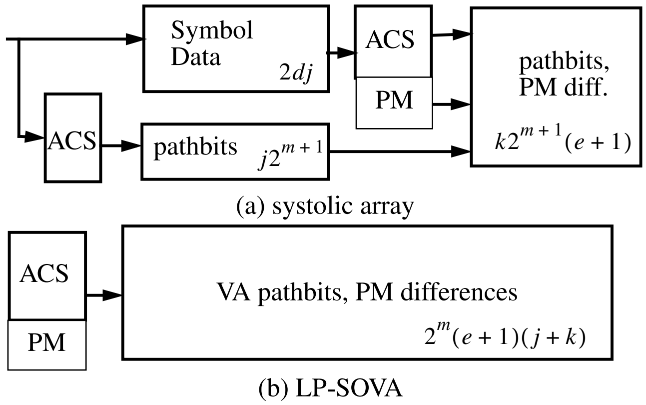

Fig. 11. Data storage comparison

samples on a per cycle basis, Consider a typical SOVA appli-

$$N_{SOVA} = 2dj + j2^{m+1} + k2^{m+1}(e+1) \qquad (1)$$

$$N_{LPSOVA} = 2^m(e+1)(j+k) \qquad (2)$$

$$\alpha_{SOVA} = 1 \qquad (3) \qquad \alpha_{LP} = \frac{2^m + k}{2^m(j+k)} + \frac{1}{2^m(e+1)} \qquad (4)$$

cation for turbo codes with 4-bit data samples and 6-bit pathmetric resolution, while the storage requirements are on par, LP-SOVA has less then 16% active for a 4-state code as seen in Table 1..

# of states		Systolic SOVA	LP-SOVA
4	storage	632	616
	active ratio	1.0	0.16
8	storage	1600	1680
	active ratio	1.0	0.09
16	storage	3688	4144
	active	1.0	0.06

Table 1: SOVA Storage Comparisons

While we have demonstrated sharp decrease in transition activity, the power is also dependent on other factors such as loading on the particular circuits, and actual circuit implementations. While memories are generally penalized for power with large bitlines, in this application the sizes are relatively small, and the RAM should not have any disadvantage over the systolic array in terms of loading. The other advantage is that access to the data is through a common bitline, thus eliminating a large section of interconnect wiring between the pathmetric and the traceback pipelines. In the data storage area, which should account for around half of the SOVA total power, we expect to see close to an 84% reduction in power for data storage based on the activity ratioses.

6 CONCLUSIONS

Turbo codes have achieved excellent coding performance, effectively receiving data when Gaussian noise contains almost as much power as the actual signal. The opportunities for turbo codes in future PCS is widespread, but due to the complex algorithms required, low-power architecture are of the utmost importance. In the turbo coding algorithm, one of largest and most important blocks in the sequence detector algorithm. This paper presents a low-power architecture for the soft-output Viterbi algorithm. The key issue is to turn the systolic array into a structure where the data is retrieved from an orthogonal memory structure. There is an enormous reduction in the transitional activity for data storage, which translates directly into power savings.

7 ACKNOWLEDGMENTS

We would like to thank Dr. Stephen Wilson, Dr. Nicholas Sidiropoulos, and Eric Hall for invaluable help in understanding the Turbo Code algorithms. This work was partially supported by NSF Career Grant MIP-9703440.

8 REFERENCES

[1] Berrou, A. Glavieux, and P. Thitimajshima, *Near Shannon limit error-correcting coding and decoding*, Proceedings 1993 International Conference on Communication, p. 1064-1070.

[2] L. Perez, J. Seghers, D. Costello, *A Distance Spectrum Intrepretation of Turbo Codes*, IEEE Transactions on Information Theory, Vol. 42., No. 6, Nov 1996, p. 1698.

[3] S. Wilson, *Digital Modulation & Coding*, Prentice-Hall, 1996, p. 604.

[4] M. Jezequel, C. Berrou, C. Dillard, P Penard, *Characteristics of a Sixteen-State Turbo-Encoder/Decoder (TURBO)*, International Symposium on Turbo Codes, 1997, p. 280-283.

[5] *CAS-5093 Turbo-Code Codec*, Comatlas, Inc. product data sheet, May 1995, rev 4.1.

[6] S. Benedetto, G. Montorsi, *Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes*, IEEE Transactions on Information Theory, Vol. 42, No. 2, March 1996, p. 409-428.

[7] K. Koora, et. al., *From Algorithms testing to ASIC Input code of SOVA algorithm for TURBO-Codes*, Proceedings of Turbo-Code Seminar, 1996.

[8] L. Bahl, J. Cocke, F. Jelinek, J. Raviv, *Optimal decoding of linear codes for minimizing symbol error rates*, IEEE Transactions of Information Theory, Vol. IT-20, Mar '74, p. 284-287.

[9] J. Hagenauer, P. Hoeher, *A Viterbi Algorithm with soft-decision outputs and its applications*, Proceedings of IEEE 1989 Globecom Conference, p. 1680-1686.

[10] P. Robertson, E. Villebrun, P. Hoeher, *Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain*, Proceedings of the '95 ICC, p. 1009-1013.

[11] A. Viterbi, *Error Bounds foor Convolutional Codes and a Asymptotically Optimum Decoding Algorithm*, IEEE Transactions on Information Theory, vol. IT-13, Apr '67, p. 260-269.

[12] C. Berrou, et. al., *A Low Complexity Soft-Output Viterbi Decoder Architecture*, Proceedings of 1993 ICC, p. 737.

[13] T. Trung, et. al, *A VLSI Design for a Trace-Back Viterbi Decoder*, IEEE Trans. on Communications, Vol. 40, No. 3, March 1992, p. 616-624.

[14] P. Gulak, T. Kailath, *Locally Connected VLSI Architectures for Viterbi Algorithm*, IEEE Journal on Selected Areas in Communications, Vol. 6, No. 3, April 1988, p. 527-537.

[15] S. Kubota, S. Kato, *Novel Viterbi Decoder VLSI Implementation and its Performance*, IEEE Transactions on Communications, Vol. 41, No. 8, August 1993, p. 1170.

[16] J. Rabaey, *Digital Integrated Circuits: A Design Perspective*, Prentice Hall: New Jersey, 1996. p. 585-587.