# A Low-Power Self-Timed Viterbi Decoder

P. A. Riocreux, L. E. M. Brackenbury, M. Cumpstey, S. B. Furber
{peter.riocreux,lemb,cumpstem,sfurber}@cs.man.ac.uk
Amulet Group, Dept. Computer Science, University of Manchester,
Oxford Rd., Manchester, M13 9PL, UK.

## Abstract

*Viterbi decoders are used for decoding data encoded using convolutional forward error correction codes or data that suffers from inter-symbol interference. They occur in a large proportion of digital transmission and digital recording systems, including digital mobile telephony and digital TV broadcast, CD-ROM and magnetic disk reading. This paper describes a design for a self-timed Viterbi decoder. The new design is based upon serial, unary arithmetic for the manipulation and storage of metrics. In the trace-back system, multiple concurrent trace-backs may be running and trace-backs are terminated as soon as they cease to be useful. The new architecture occupies between 29% and 23% less area than a selection of synchronous implementations with the same design parameters which use the same process and cell-library.*

## 1 Introduction to the Viterbi algorithm

The Viterbi algorithm[6, 3] is used widely in the digital transmission field as a means of decoding convolutional forward error correction (FEC) codes. The Viterbi algorithm is a maximum likelihood probability (MLP) method; the output of the decoder is the sequence most likely to have been transmitted, given an observed received sequence.

Convolutional encoding is a highly asymmetric coding scheme - encoding is trivial, but decoding is very computationally expensive. As a result of this and the fact that the decoding is often performed at the same time as other computationally intensive tasks, decoders are often implemented in dedicated hardware, although smaller decoders and those operating at lower data rates have come within reach of software in recent years.

### 1.1 The encoding process

In order to understand the decoding process, the encoding process must first be understood. The basic building block of the encoder is a convolution and parity calculation. The last $k$ bits of the input stream are stored in a shift register and these are convolved (XORed) with a $k$-bit long pattern and the parity of the resulting stream is the encoder's output corresponding to the most recent input bit.

Redundancy (and hence FEC) is added to the bit stream by outputting multiple bits for each input bit, with each of the output bits being the result of using a different pattern in the convolution process. The ratio of input bits to output bits ($v/n$), called the *rate*, may be any rational number less than 1 but must be selected at design time. Artificial rates lying between the real rate and 1 can be produced by "puncturing", and this is the usual means of obtaining rates between $1/2$ and 1. Standard convolution patterns exist for each (non-punctured) value of $v/n$ and $k$ that have been shown to be optimal for additive, white, Gaussian noise. The decoder described herein is a $1/2$-rate decoder with a constraint length ($k$) of 7 and therefore 64 states.

The presence of $k$ storage elements in the encoder means that it acts as a state machine of $2^{k-1}$ states. Every state has two routes from it, corresponding to the bit arriving most recently being a zero or a one, and two routes into it, corresponding to the bit which has just exited the shift register being a zero or a one.

It should be noted that a transition into exactly half the states corresponds to the encoding of a one and the other half to the encoding of a zero.

### 1.2 The decoding process

When the received pair of encoded bits arrives at the decoder, the decoder must infer, from what is received, the data that were fed to the encoder. The encoded version of these data may have been corrupted on its journey from source to destination, so the decoder must be able to infer the intended values from this corrupted stream.

As a transmitted pair has different significance depending on the state the encoder was in when it generated them, so must the receiver attach different significance to a received pair depending on which state *it believes* the encoder to be in. As the decoder does not *know* anything about the state the encoder was in when it generated the received pair, it must hypothesise about this and test the hypothesis in some way.

This is done by generating, for each possible state that the decoder can be in, a measure (called a branch metric or BM) of how far the received data is from the "perfect" data that receiver would expect to see for the error-free transmission of a 0 and a 1. To complete the hypothesis testing, these BMs are combined with a measure (called a state metric or SM) of the likelihood, based upon previously received data, that the encoder was in each state. These combinations of state and branch are called path metrics (PMs) and are measures of the likelihood of each branch being correct.

In other words, the decoder calculates simultaneously for each state and branch - *what is the likelihood that this state is the correct one to be in, and therefore that the data received is due to the encoding of data corresponding to this branch.*

As each destination state has two routes into it, and each route has an associated PM, it will be seen that the likelihood of a state being the correct starting point for the next cycle is the higher of the two likelihoods leading to it, i.e the more likely of the two routes to it. This higher likelihood is saved and becomes the SM for that state in the next cycle. The identity of the path into a state that had the highest likelihood (the *local winner*) is also recorded in the survivor memory for use later to establish which path through the state space the encoder took.

In practice the likelihood metrics are reversed in sense, so that the most likely path has the lowest metric. This means that when the data are error-free, the winning branch from the correct state - the *global winner* - has a zero value BM associated with it in each cycle and therefore its PM and hence SM also remain at zero.

The decision made by the trellis system regarding the best path is based only upon the most recently received $k$ bits. A burst error could easily last longer than this so another stage which accumulates information over a longer period is used as well. Any state will, if its ancestry is followed back a sufficient number of time intervals, prove to be descended from the correct path, and the local winner values encountered along the route indicate the path taken through the state space to get there. Thus the local winner data can be used to find the correct path through the state space when a large number of sets of local winners are stored. The process of following these local winner indications to find the path is known as a trace-back.

The data output by the decoder is taken from the oldest set of local winners in the memory, and thus the system effectively accumulates evidence for the correctness of its estimate for a number of cycles equal to the depth of the survivor memory.

## 2 A conventional decoder

In the reference decoder design from our industrial partner, the architecture is largely determined by following the algorithm as described in the previous section. There are three basic sections to the design, the branch metric unit (BMU) the path metric unit (PMU) and the survivor memory unit. The BMU is responsible for creating the BM values from the incoming data that are passed to the PMU. The PMU is responsible for adding the BMs to the SMs to create PM values and determining local winners, which in their turn become SMs. The survivor memory unit is responsible for recording the local winner identities and producing the output stream by performing trace-back.

### 2.1 Conventional PMU

The PMU accumulates BM values, and determines the lower value of a pair of these accumulated totals. This add-compare-select (ACS) operation is almost invariably implemented in a parallel fashion in high speed decoders as it is in this design, with there being a discrete unit corresponding to each state.

In order to avoid arithmetic overflow of the PMs as they grow, two different methods are commonly used, both of which exploit the fact that the greatest possible *span* of PMs (the difference between the largest and smallest) is $(k-1) \times BM_{max}$ [7].

The first of these is to subtract a proportion of the maximum accumulated value when all of the states have accumulated at least that. This is practical provided the arithmetic width of the metrics is sufficiently large as there is a predictable minimum number of cycles between the last cycle during which not all of the metrics were greater than the chosen value, and the first cycle in which any of them could overflow[5].

The second is to use arithmetic units that ignore the overflow condition, but where the comparison system understands that an apparently small value may in fact be a larger value which has wrapped around[4].

In this reference architecture, no effort is made to find the global winner (the state whose SM is the smallest).

### 2.2 Conventional survivor memory unit

The survivor memory unit is a design somewhere between a one-pointer trace-back and a k-pointer trace-back ($k = 2$) architecture[2] with equal size write, merge and decode blocks, plus a further block in which to store the data that are to be written out.

The memory is split into four units whose function rotates. Local winner identities are written into the first, primary and secondary trace-back are performed in the second

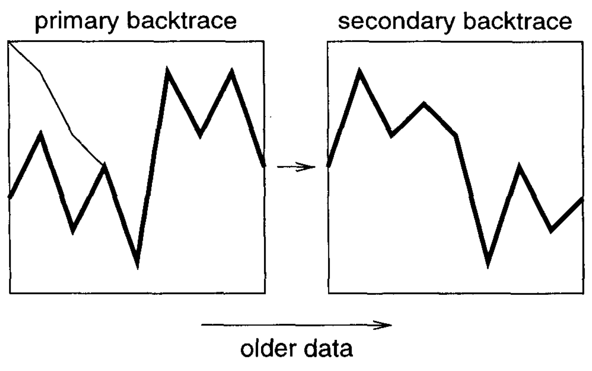

**Figure 1. Primary and secondary trace-back.**

and third, and the fourth is idle. Figure 1 illustrates the primary and secondary trace-back process; the quadrilaterals represent an array of local winners (each column representing a time interval, the youngest on the left hand side where the trace-back process starts), the thicker line indicates the true path through the states and the thinner one the path taken by the trace-back process.

In primary trace-back, it is assumed that *any* starting state will eventually converge on the correct path, so an arbitrary location (say the top row in Figure 1) is chosen and trace-back begins there. This method attempts to find the global winner at the far end of this primary trace-back by assuming this convergence. The state at which this primary trace-back finishes is then used as the starting location for the secondary trace-back. The states navigated in this secondary trace-back are recorded so that the bit for which they encode may be determined (recall that all possible paths that end at half of the states indicate the encoding of a one and those ending in the other half of the states indicate the encoding of a zero).

The two stage process is necessary because, as can be seen from Figure 1, until convergence occurs, the wrong path is taken through the memory. If this path were used to generate the output then the wrong data would be generated.

## 3 A new, self-timed architecture

In this new architecture the design is a mixture of 2-phase and 4-phase circuits. 4-phase operation was chosen for the control aspects of the architecture for ease of design. The datapath is implemented as an event storage based circuit with a number of 2-phase handshakes being enclosed by the 4-phase control system. 2-phase is chosen here for reasons of speed (a halving of the number of events, whilst not doubling the speed does bring a significant improvement) and area (a 2-phase FIFO has half the number of control elements of its 4-phase equivalent). An additional consideration is the requirement for the unambiguous detection of full and empty states on FIFOs, which is substantially more difficult with a 4-phase implementation than with 2-phase.

The three sections described below and illustrated in Figure 2 fulfil basically the same functions as their equivalent in a synchronous system. The presence of a clock signal in Figure 2 is due to the design being created for integration in a larger, synchronous system, which requires synchronised input of received data and output of decoded data.

The sections can operate in separate pipeline stages and, in the case of the BMU and history unit (HU, known as survivor memory unit in synchronous architectures), these are sub-pipelined. Unfortunately, due to the nature of the recirculation of data in the PMU, it is impossible to further pipeline its operation, although some pseudo-pipelining is achieved by overlapping non-exclusive parts of the operation.

An additional constraint on the PMU is that in our design it requires a self-synchronisation point, at the interface between the PMU and the HU. This synchronisation point is caused by the need to have found (or at least attempted to find) the global winner state, which requires all the local winner states to have been found.

### 3.1 Asynchronous BMU

As a design aid, a C program was written to quickly allow the simulation of this system with different policies on the sizes of BMs, and many other properties of the design. In order to reduce the arithmetic size of the BMs, prescaling, predecrementing, and capping[1] are desirable. Use of the C simulator allowed us to verify that reduction of the maximum size of BM that must be handled from 98 (as a 3-bit soft decision system is used) to only 6 would still permit the required error correction performance from the decoder.

The BMs are not represented as binary values however, but rather as the pattern of states that would be seen in a 2-phase micropipeline event-FIFO built from Muller C-elements if that number of transactions had been completed on its input port. The BMs are formatted this way to facilitate their loading into the PMU - it describes the state the C-elements must be set to. Because the event FIFO is 2-phase, the number of events is represented by the number of changes of phase in the pattern.

Thus a zero is encoded as 000000, one as 111111, two as 000001, three as 111101, four as 000101, etc. The encodings for 0 and 1 appear to be the same but the event-FIFOs are reset between cycles so there is an implicit 0 to the right of the patterns. There is also an implicit bit to the left of the patterns that matches the left most bit.

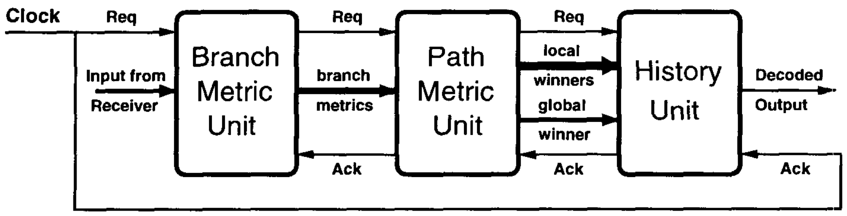

**Figure 2. The asynchronous architecture.**

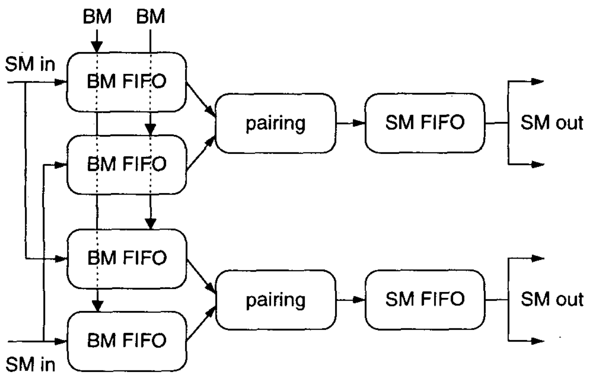

**Figure 3. A pair of nodes from the path metric unit.**

## 3.2 Asynchronous PMU

The PMU in this new design is a fully parallel system as was the reference design, but differs from it in that is has no conventional arithmetic, and values are stored in a serial, unary manner, as a number of events in a micropipeline event-FIFO. The Viterbi algorithm only cares about the relative sizes of the BMs or SMs to be chosen between, rather than the absolute size of any of them, and this is easily computable with this unconventional representation of the data. The absolute value of the smaller BM must be preserved, but its value need not be known or expressed explicitly.

The flow of data to and from a pair of state processors (known as nodes) is illustrated in Figure 3. Node-pairs are almost invariably used as the unit of replication in Viterbi designs because, due to the nature of the "butterfly" interconnection network between the right-hand and left-hand sides of the nodes, particular pairs of nodes have similar routing requirements. Making the unit of replication a pair simplifies the placement and routing stage of design[1].

### 3.2.1 Event add-compare-select

The ACS function is implemented as two parts in the new architecture. The BMs are bit-parallel loaded into the BM FIFOs and then the SMs are added to them as a series of events on the FIFO input port.

In order to compare the sizes of two values stored in the BM FIFOs, the output requests of the two FIFOs are fed via a Muller C-element to another event-FIFO, so that this lower FIFO ends up with a number of events in it equal to the smaller number of events in the upper two FIFOs. Thus the local winner value is determined, and the local winner identity is merely an encoding of which of the two branch metric FIFOs became empty first. This equates to the compare-select part of the ACS.

A more detailed diagram of the datapath of a single node is shown in Figure 4. Here we see the overflow units which allow us to try to pass up to 7 tokens from the feeding SM FIFO into a BM FIFO which may already contain 6 (of its maximum 7) without error. Normally the overflow units pass the requests to the BM FIFO and pass the acknowledges it receives from the BM FIFO back to the SM FIFO that is feeding it. This continues until the BM FIFO indicates that it is full. From that point input requests are diverted straight to input acknowledges.

The phase compensators are required because the SM FIFOs that are feeding the BM FIFOs have unknown phase (they can be in state all-1s or all-0s when empty). This also impacts the value loaded into the BM FIFOs, which must be inverted if the SM FIFO is in the all-1s state when empty. The complexity of the data path between the output of the SM FIFOs and the input of the BM FIFOs represents a serious bottleneck for this design.

### 3.2.2 Global winner determination

As the global winner identity *is* required immediately in every cycle in our architecture (unlike in most conventional architectures) this must be determined from the values stored in the SM FIFOs. The Viterbi algorithm requires only that the relative sizes of the SMs be correct so we may subtract

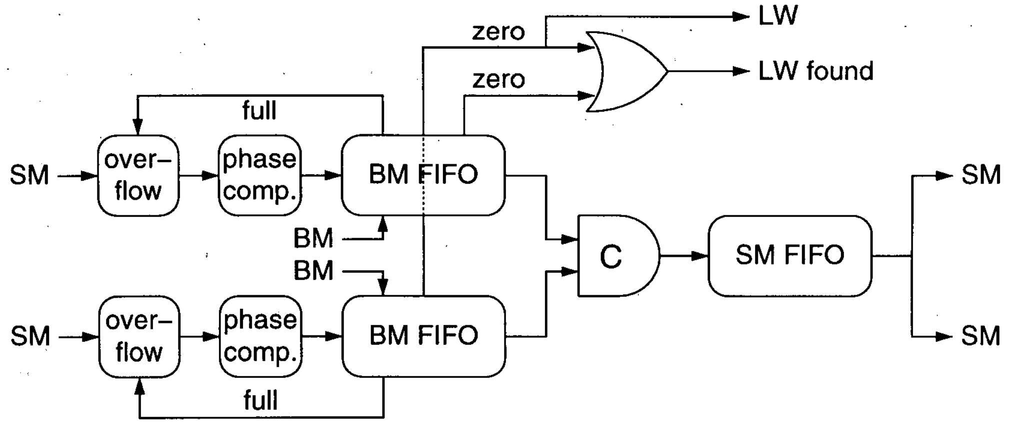

**Figure 4. A path metric unit node data path.**

an arbitrary value from all SMs. Thus the global winner may be found simply by subtracting 1 from each of the SM FIFOs (removing an event) repeatedly until one or more of them is empty (an easy condition to detect from a circuit standpoint). This is only practical to do if SMs remain small, but if it is done whenever possible, the SMs do remain small.

Using the C simulator program, analysis of the distribution of the global winner's score prior to this global decrementing shows that for the majority of the cycles the global winner has a 0 score, and in most of the remaining cases the score is 1; the frequency with which the global winner has a score of more than 2 is so low that we may treat it as a pathological error case and decide that no special efforts should be taken to improve performance for this case. This distribution of global winner scores is due to the fact that the majority of received data are, in practice, error-free, even at poor SNRs.

In light of this fact, it was decided to change the algorithm for decrementing all the SMs and rather than decrement until a zero score was found, to decrement either zero or one times and defer this decrement until the following cycle. This was done in order to simplify the high-level control of the design, whilst maintaining the same overall mode of operation.

If a decrement is needed and thus no global winner were available this is indicated to the HU which does not initiate a trace-back for that cycle. In this manner, the global decrementing system need not perform a decrement and search-for-zero loop in order to try to establish when a zero is available; this could be very slow for a large number of nodes and would require synchronisation after each decrement. The fact that a state has a SM value of zero may be detected by the pairing system, and thus confirmation of the presence of at least one SM with a zero value is available before it would be possible to detect this solely from the contents of the SM FIFOs.

Once the global winner has either been established or is determined to be unknown, the SMs can be recycled across the butterfly network under local control and the whole cycle recommences. During this recycling process, if a global decrement is needed, the first transaction on the BM FIFOs' input ports is simply acknowledged without any action.

Everything in the core of the PMU is controlled by extremely local handshakes, except the butterfly network connections and the determination of whether any node has a SM value of zero. The butterfly routing problem can also be largely overcome by use of appropriate floor-planning which allows the node-pairs to be placed so that none must connect to anything more than 3 replication units distant.

### 3.2.3 SM capping

Other analysis of the system dynamics revealed that states with scores above a certain value were almost never incorporated into the path, so retaining much information about their score was pointless. This means that we may place an artificial limit on the maximum value that the FIFOs need be able to store, and discard incoming events once this value is reached. It is desired to keep the SMs to the smallest size consistent with meeting the specification as the size of the SMs has an impact on the area, power consumption and speed of the design. The effect of this is illustrated in Figure 5, which shows the BERs for various capping limits relative to the uncapped case.

It can be seen that a limit of 6 is clearly too little, but that 7 quickly converges with the uncapped case and is coincident with it by a SNR of 5dB. In the case of our $k = 7$, $n = 2$ decoder this limit need only be 7, as this still allowed us to meet our design specification.

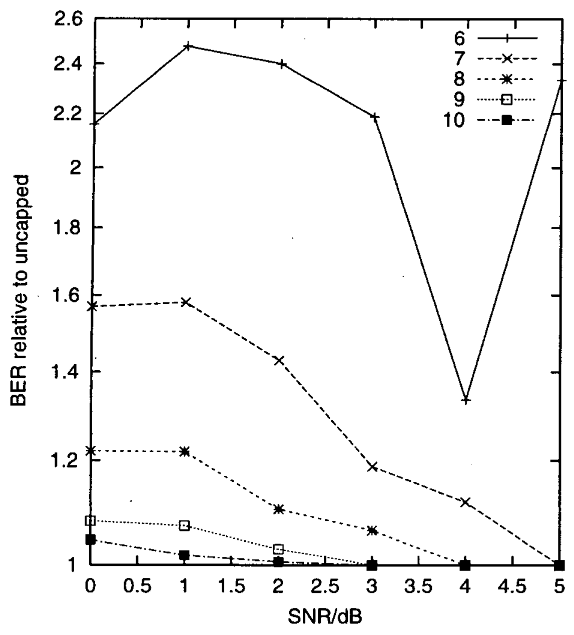

**Figure 5. BER for various capping levels relative to uncapped values.**

The capping is performed in our design by the use of the relatively short BM FIFOs and the overflow units. This proved to be an extremely problematic aspect of the design. In addition to the overflow action this overflow unit also handles the discarding of the first increment request in order to perform the decrement operation described in the previous section. It is believed that this unit represents the most significant bottleneck in the design and had the complexity of this unit been anticipated, a different system might have been designed.

## 3.3 Asynchronous HU

The HU also differs significantly from the survivor memory unit found in most conventional architectures. In our design, the local winner data are not stored in a RAM-like structure, but instead in distributed storage structure which is currently implemented as an array of latches. This appears a much less efficient means of storage from an area point of view but it is effectively 65-ported in its read access and single ported in its write access. This structure is necessary to allow the operation described below. A 65-port structure would normally have an astronomical cost, but in this case each read access port need only ever connect to one address.

In this new HU design, instead of having three or four blocks of memory that are used for trace-back, all four operations (data entry, primary and secondary trace-back and readout) take place in one region. This single block of storage acts as a sliding window, holding data from the last $D$ cycles of the PMU. In addition to containing the local winner data it also contains the global winner data. The value of $D$ may be chosen at implementation time and can have any value desired. There exists a rule-of-thumb that the trace-back history should be at least five times $k$ for to ensure convergence in (practically) all circumstances, in this case 35. For our design a value of 65 was chosen for $D$ as we were unsure of the efficacy of our trace-back unit.

The output from the HU is derived from the oldest entry in the winner memory. As may be recalled, exactly half of the encoder and decoder states correspond to the most recent bit being a zero, and the other half to a one. If the binary numbering of the states is performed in the correct manner, the least significant bit (LSB) of the state number indicates to which it corresponds. So the global winner corresponding to the oldest entry encodes, in its LSB, the datum that should be output.

### 3.3.1 Asynchronous trace-back

It may be recalled that the global winner cannot be relied upon to be correct at all times, so a trace-back process is still required. When a new local and global winner entry is made into the winner memory (by overwriting the slice containing oldest entry), the hypothesised previous global winner is computed and passed to the slice containing the entry made in the previous cycle; parent computation is trivial, simply involving a 1-bit shift and the replacement of the emptied bit with the local winner that led to the current global winner. This next slice compares the hypothesised value with the winner that it has recorded and if it is the same, and the winner path therefore continuous, the trace-back operation is *retired*. If the two values differ, the hypothesised value is assumed to be correct, the stored value is overwritten with it, and a new hypothesised value for the next oldest entry is computed etc. In this way, the trace-back continues down the history (backwards in time) until *exactly* the point at which it will no longer change anything in the store.

This trace-back operation is managed at an entirely local level, by handshake between adjacent slices of the history memory, without the need for any global control and is decoupled from the other processes in the HU as soon as it is initiated. It can also be seen that because of this fire-and-forget nature, multiple concurrent trace-backs may operate at any time without any performance overhead. In addition, the speed at which the trace-back process operates is unimportant as the trace-back and data storage processes move in opposite directions around the HU and are entirely separated from one another.

Multiple concurrent trace-backs are required to enable errors in the global winner determined by the PMU to be corrected. When the trellis first diverges from what is actually the correct path because of corruption of the received data, a trace-back is initiated because the previous and current winner are not continuous. When the trellis returns to the correct path some short interval after the received data cease to be sufficiently corrupt, a second trace-back must start as there is again a discontinuity (there may be many others during the corrupt period). The first trace-back will converge with the existing path reasonably quickly as when an error first occurs it cannot have diverged from the true path very far. The final trace-back will continue to at least the same point as the first, to correct the erroneous changes made by the first or intermediate trace-backs. If at any point the trellis indicates that it has not found the global winner then a trace-back process is simply not initiated.

This mechanism appears at first glance to be a k-pointer trace-back architecture[2] with $k = D$, but it differs in some important respects:

- the value of D is not constrained by the algorithm in any manner, only by the error-correction performance required
- no part of the storage has a particular purpose at any time: it is not split into write, merge and decode blocks
- a variable and unknown number of trace-back passes will occur over every slice of the HU
- there are a variable and unknown number of trace-backs running at any time
- the best estimate of the global winner is maintained at each slice of the HU.

There is a theoretical chance that a pathological data set could cause a trace-back to run all the way to the oldest entry of the memory and collide with the data entry phase occurring at the other end of the HU. In these circumstances the trace-back must be forcibly retired, and prevented from interfering with the data entry. To do this a mutual exclusion element is used to ensure that both operations cannot try to access the same resource. This required the use of the only custom element in the design, which was otherwise built using only the elements available in our industrial partner's standard cell library.

### 3.3.2 HU implementation

The trace-back architecture is illustrated in Figure 6 which shows the concept of the HU as a sliding window on the data. The evaluate channel handles the multiple trace-backs while the token channel handles which slice of the HU is at the *head*, receiving new data and outputting decoded data. The control unit of each slice is responsible for tracking the stored global winner, managing/retiring the trace-backs and passing the head token.

The winner store's non-RAM based implementation is driven by the multiple concurrent trace-back algorithm: many slices of the memory may be accessed simultaneously and asynchronously with respect to each other. The access patterns are also unusual in that writing into the store is performed on all the bits (one for each state) in one slice simultaneously, but only a single bit from each of possibly multiple slices is read at a time. Because of the desire to implement as much of the design as possible with standard cells, this unusual structure was implemented with 2-to-1 multiplexers acting as latches, and a tree of 4-to-1 multiplexers selecting the single bit. Multiplexers were used as the storage element because the 2-to-1 multiplexer was significantly smaller than the simplest latch in the cell library. Undoubtedly this could be implemented in a much more efficient manner if full-custom design were used.

## 4 Comparison of conventional and new architectures

The decoder was designed as part of a collaborative project to examine low-power techniques. Each of four partners at separate institutions designed a decoder (the other three synchronous) to the same specification using the same fabrication process and, if appropriate, cell library. The relative merits and demerits of the four approaches (the industrial reference design, asynchronous, SPL logic and algorithmically aware synthesis) are to be examined.

The four designs have finished fabrication on a $0.35\mu m$ 3-layer metal process and are undergoing more extensive performance testing at the time of writing. A test board has been designed to allow high-speed testing with a real video signal, and the supporting system to do this is currently undergoing final construction and test. As a result, detailed performance figures are not yet available

Unfortunately the layout parameter extraction rules supplied by the industrial partner for the process in which the designs are being fabricated do not allow reliable capacitance extraction so accurate power figures are were not available through simulation. However the fabricated devices have been tested and decoding an uncorrupted stream at a rate of 25Msymbol/s the decoder parts of the devices (excluding the peripheral block which is common to all designs, and the pads) consume 203mW (reference), 88mW (SPL), 70mW (synthesis) and 9.2mW (our design). This represents a much larger reduction in power dissipation than the authors' wildest hopes believed possible, so some doubt regarding the testing remains.

As mentioned above, a C simulator of the new architecture was created to determine required parameters for the

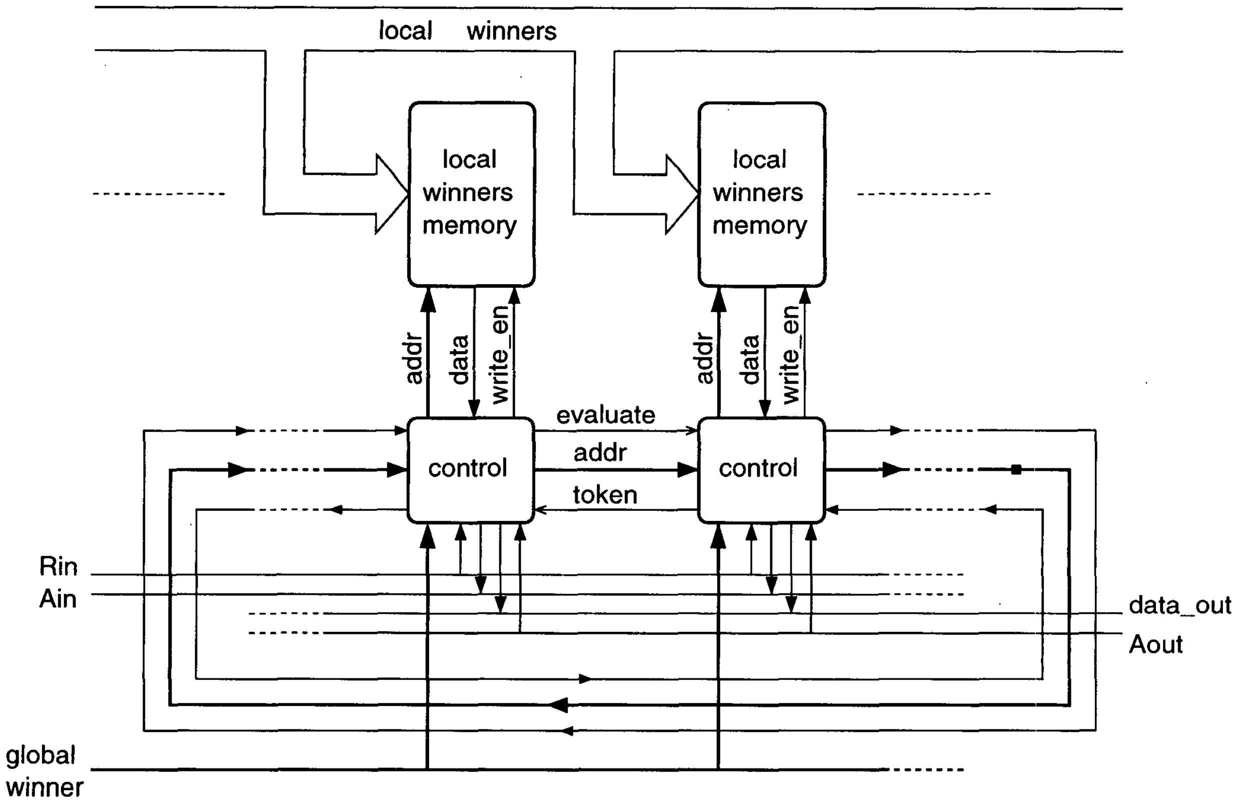

**Figure 6. The asynchronous trace-back mechanism.**

new design. Figure 7 shows the bit error rates (BERs) versus the signal to noise ratio (SNR) for the C simulator, the Verilog simulations of the architecture, the design specification and the performance that would be achieved if no coding scheme were used.

The C simulator prediction crosses the required performance curve for a short segment around the corner of the specification, but it will be observed that the performance as predicted by the Verilog simulations is consistently better than that predicted by the C simulations, and is comfortably within the performance specification at all times. This discrepancy is due to the performance of the trace-back mechanism. The C simulator perfectly reflects the operation of the BMU and PMU, but has much more difficulty with the multiple concurrent trace-backs and cannot perfectly reflect their operation. Despite many hypotheses and attempts to discover the true cause for this discrepancy, no firm explanation has been forthcoming, and thus it has been impossible to make the C simulator more accurately reflect the Verilog simulation results.

Our design has an interesting and possibly advantageous characteristic of its power consumption in that it will vary depending on the SNR of the received signal: when the SNR is low, more trace-backs will be required and those that are done will be longer on average. This will increase the power consumption in those circumstances, although without accurate extraction data it is not clear how much this will affect the overall power consumption.

Area figures show that the usual asynchronous area overhead is not present here, in fact our design is the smallest by a significant margin: 29% smaller than the reference design (a commercial product) and 23% smaller than the SPL-based design. A large part of this gain was achieved by the removal of the SRAM that all the other designs use in their survivor memory units. The reference design uses four single-ported 64-bit by 128 word SRAMs while the other synchronous designs use one, dual-ported 64-bit by 128 word, one 6-bit by 128 word SRAM and two small single-ported SRAMS. Our design uses only 64-bit by 65 word storage, and the area of that, even though implemented with multiplexers is about the same as the one large dual-ported SRAM.

In order to avoid as far as possible the creation of new cells for use in the design, functions such as Muller C-

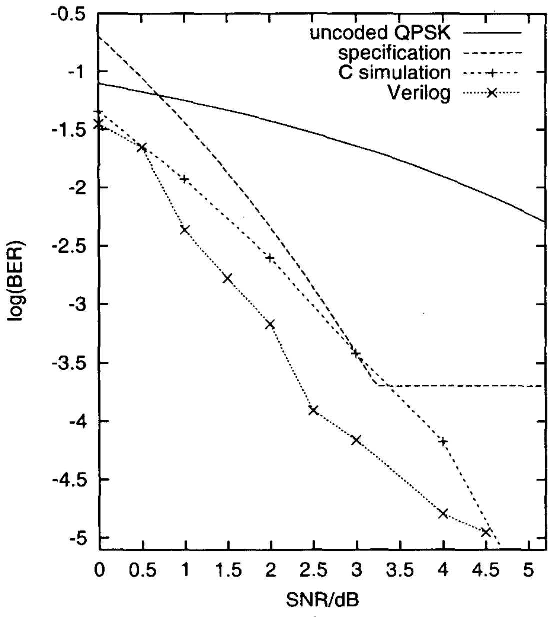

**Figure 7. Error correction performance of the new architecture.**

elements were built from standard cells (and-or complex functions in the case of Muller C-elements) and these were significantly slower than the speeds we believe could have been achieved with custom cells. In addition, where C-elements have to be resettable (most of them), using the standard cell implementation required the insertion of an extra gate delay in both the forward and feedback paths. As the whole of the datapath makes heavy use of 2-input C elements and they are liberally used elsewhere, a custom cell would undoubtedly improve the speed. A significant area reduction would also result from the use of custom cells as the standard cell implementations of the Muller C-element are highly non-optimal and custom versions could be implemented with fewer transistors (8 instead of 12 for a 2-input Muller C-element).

Further area savings are also possible as the design that was fabricated was automatically placed and routed from a flattened cell netlist. Subsequent work has established that the cell density may be raised from the ≈ 85% in this design to greater than 99% for the HU and more than 95% in the PMU by hand placement and hand optimisation of automatic routing; these two units account for ≈ 95% of the total cell area of the design. In this case hand placement and hand optimised routing is practical because the enormous amount of replication that exists in those two units mean that only a handful of blocks need be done in this way.

One of the principal advantages, from an architectural standpoint, of our design is the reduction in latency. In the reference design the latency is 540 bits, more than eight times the 67 that is required in our design. The latency in our design has been artificially increased to around 144 so that all three of the new designs may share a test system - the other designs require a latency of at least 134.

Latency is important in the area of digital TV decoding (the application of the reference design) as in a typical application the higher level control system will have unknown operating conditions (such as the code rate, receiver characteristics, etc.) and a multi-variable parameter space must be searched to find the correct parameters. This is done by re-encoding the decoded data and comparing them with the received data to decide when the correct parameter values have been chosen - the error rate will be about 50% if the parameters are wrong. A high latency means that a larger number of samples of received data must be stored and more samples are required before any error counting may begin.

The drawback of our design is its speed of operation which in simulation varies between 77% and 44% of what was sought (45Msymbol/s). Where punctured codes are used the rate of samples presented to the decoder is a function of the coding rate and the symbol rate at the receiver. When the same clock rate is used for all code rates, as here, the "invalid" symbols that pad the data stream are simply ignored by our decoder so that for code rates where these are more prevalent (e.g. 1/2) we can achieve a higher proportion of the desired performance. Testing of the fabricated silicon indicates that the decoder is functional up to at least 25Msymbol/s.

The ideas developed in the design of the HU could be applied to a clocked design, albeit in a perhaps less efficient manner and maybe requiring a larger design overhead. Indeed, an earlier version of the HU design underlies the survivor memory unit used in the two new clocked designs. The storage mechanism for the local winners presents a difficult design problem in this respect. Either a structure similar to that used in this design is required, or if a conventional multi-ported RAM structure were used, a complex arbitration system to govern access to the ports is required.

The PMU design cannot, however, be used in a clocked design as the means of performing arithmetic on and storage of the data are essentially a self-timed concept.

The PM comparison method if ported to a synchronous system would require the repeated incrementing and decrementing of an up-down counter; this would be an inefficient way to find the smaller value and would require an impractically high clock speed to allow the sort of performance that is sought. It would perhaps be possible to use a one-hot encoding scheme instead of a binary representation for the values, although communicating values encoded in this way around the butterfly would have a prohibitively large

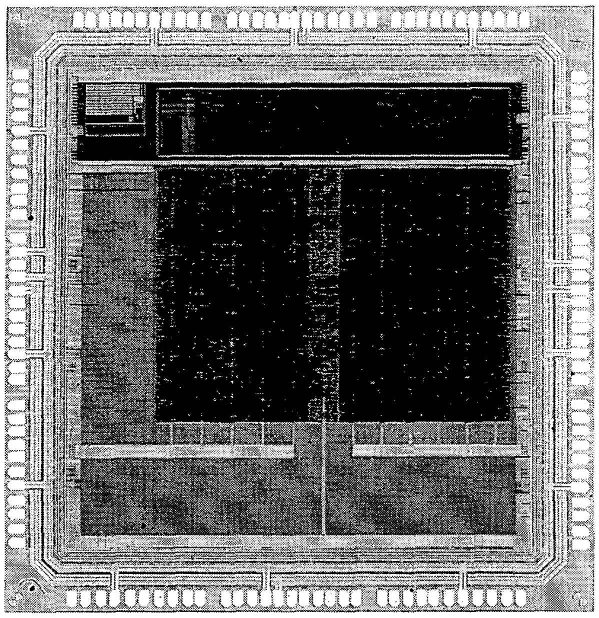

**Figure 8. The asynchronous decoder silicon.**

routing overhead.

The partners creating synchronous designs have adopted some of the architectural elements of our design whilst remaining in a synchronous framework. In particular a single-memory trace-back system is used, but it only allows one active trace-back, plus up to 16 suspended trace-backs to be operational. They must also find the global winner, which is done by means of a collapsing tree of binary comparators; they do not have a zero SM score other than by chance.

A photograph of the manufactured silicon containing our design is shown in Figure 8. The picture shows the PLL needed by the common, synchronous, peripheral block in the top left corner, the peripheral block itself across the rest of the top of the die, and our decoder occupying the space below. The large amount of unused die area around the edges of the decoder are due to the fact that all of the designs were to occupy the same pad ring.

## 5 Conclusions

We have described a new, self-timed architecture for a Viterbi decoder. It occupies somewhat less area than alternative synchronous designs with manifold sources of further area reduction available.

The power consumption also appears to be approximately an order of magnitude less than that of the the other novel designs, and twice that again less than the reference design (when decoding an uncorrupted stream).

The design cannot be translated directly into a synchronous decoder and given that it should have the property of low EM emissions which is particularly desirable in a communications-related circuit, it has the potential to demonstrate the commercial appeal of asynchronous circuits.

In the authors' view, the most exciting aspect of the architecture was the the exploitation in the HU of one of self-timed technology's greatest assets - doing nothing when there is nothing useful to do. This has had a significant impact on the power consumption when good signal conditions exist, but should not in any way compromise performance when signal conditions are poor. This aspect of the architecture is also the one which is least possible to translate into a synchronous system.

An interesting aspect of the design is that it illustrates the use of both high-speed serial operation (in the PMU) and lower speed, (possibly very) parallel operation (in the HU) in the same architecture. These two approaches were applied where their use seemed most appropriate. This diversity of architectural styles would have been considerably more difficult to achieve in a clocked design.

## 6 Acknowledgements

The work described above was supported by the EPSRC PowerPack project (project code GR/L27930) and the EU PREST project (project code EP25242).

The authors wish to acknowledge the assistance of Keith Lyons of Mitel Semiconductor (formerly GEC Plessey Semiconductor) in understanding Mitel's design.

## References

[1] G. C. Clark Jr. and J. B. Cain. *Error-Correcting Coding for Digital Communication*, chapter 6, pages 256–265. Plenum, 1981.

[2] G. Feygin and P. G. Gulak. Survivor sequence memory management in Viterbi decoders. Technical Report CSRI-252, University of Toronto, January 1991.

[3] G. D. Forney Jr. The Viterbi algorithm. *Proc. IEEE*, 61(3):268–278, March 1973.

[4] A. P. Hekstra. An alternative to metric rescaling in Viterbi decoders. *IEEE Trans. Commun.*, 37(11):1220–1222, Nov 1989.

[5] J. Sparsø, H. N. Jørgensen, E. Paaske, and S. Pedersen. An area-efficient topology for VLSI implementation of Viterbi decoders and other shuffle-exchange type structures. *IEEE J. Solid-State Circuits*, 26(2):90–97, Feb 1991.

[6] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inform. Theory*, 13:260–269, April 1967.

[7] A. J. Viterbi and J. K. Omura. *Principles of Digital Communication and Coding*, chapter 4, pages 258–261. McGraw Hill, 1979.