# An Asynchronous Viterbi Decoder

**AMULET Group, Dept. of Computer Science, University of Manchester, Oxford Road, M13 9PL**

## 1 Introduction

### 1.1 The Viterbi Algorithm

The Viterbi algorithm is much used in GSM, interactive terrestrial, satellite and cable digital television applications. It is a forward error correction (FEC) system (one which pads the data with redundancy to enable recovery of the data from a corrupted stream) used in situations where streams of data are to be transmitted across a noisy channel. Receiving and decoding using the Viterbi algorithm yields a data stream that has the highest *a posteriori* likelihood of having been the data that were encoded and transmitted. The decoder also has the inherent ability to correct itself after a prolonged stream of incorrect input, and this also results in a decoder with no data framing requirement.

When each symbol is received, a Viterbi decoder hypothesises that each *possible* transmitted value resulted in the received data, and based upon how different the received and hypothesised data are, provides a metric indicating the probability of how likely each hypothesis was to have been correct. This metric, scaled to make the most likely 0, with less likely hypotheses having higher values, is called the branch metric (BM).

The state machine encoding the data that are transmitted must go through a sequence of states to be in a position to have been able to transmit the data that each hypothesis says arrived. There is a likelihood (based on previous iterations of the viterbi) of the encoder having been in each state, known as the path metric (PM). This PM is added to the BM to find the likelihood of each branch. This results in a doubling of the number of metrics, as each encoder state (PM) can result in two possible future states, depending on whether a 1 or a 0 was encoded. However, for a given new bit, two previous states result in the same resultant state, merging two branches. We now choose between the two merging paths by taking the one with the lowest BM+PM, as this represents the most likely of the two paths that the encoder followed through the state space. The chosen branch is known as the survivor, and its value of BM+PM becomes the PM for the next iteration of the process. The identity of the branch which is chosen is added to the survivor history memory for later use.

The branch with the overall lowest score represents the best estimate of the true path the encoder took, and this path is retraced to its starting point and the path at that starting point is output as the received data. Thus there is a latency approximately the same size as this path history.

### 1.2 Asynchronous Design

Lower power is best achieved by a combination of techniques ranging from algorithms through to technology. At Manchester, we have adopted an asynchronous approach as a route to lowering power. As well as exhibiting low EMC, dispensing with a global clock and replacing the timing with local handshake signals between blocks means that work is only done when necessary by the system and the idle power is near zero. Furthermore, there is an inherent advantage to asynchronous operation in that the system can switch almost instantaneously between the idle state and maximum performance; this is far more difficult to organise in a clocking system.

An asynchronous approach encourages designers to design in a modular manner. This means that sections can operate independently, concurrently and at their fastest natural rate. In the Viterbi decoder, the design partitions into two main operating sections namely the branch metric unit and the path backtrace unit. These can operate asynchronously to each other and while the branch metric unit is linked to the input data rate, the backtrace unit can be retracing at a higher rate which is clearly advantageous. Another consequence of the modularity is that synchronous designs do not necessarily translate into an asynchronous design in an obvious fashion; this arises from the level at which the local handshakes need to be placed in an asynchronous system.

## 2 Asynchronous Viterbi Decoder Design

### 2.1 Design Approach

In undertaking asynchronous design, there is a need to start afresh with the design rather than attempting to convert an existing synchronous design into an asynchronous implementation. This is the approach which has been taken in the design to be described. To aid the design and assist with our understanding of the algorithm, a ‘C’ simulator which models the effects of our proposed decoder operation has been developed. Parameters such as the retrace length, the metric count limit, and the way the comparison between received and transmitted possibilities are computed can be varied so that their effect on the error rate can be seen.

The simulator has proved to be an invaluable aid to the design process, particularly in regard to establishing a more accurate comparison algorithm and the handling of much smaller numbers in the metric unit than used in current decoders. Most of these results are applicable to all Viterbi decoders, regardless of their timing model. The greater accuracy in the comparison also leads to better detection rates than those obtained from a conventional decoder.

The reduction in number size that need be handled by the system results from simplifying the way the BMs are computed and from renormalising the PMs (subtracting the smallest value from all of them) whenever possible. The small count sizes which result enable the 8-bit-parallel add-compare-select (ACS) circuits, which are a dominate feature of the BM unit in conventional decoders, to be replaced with a serial buffer where the metric counts are held as a set of events. We believe that this approach results in significant design simplification and the consequent reduction in the logic required will lead to a significant reduction in the power requirement. As previously stated, this reduction of number size is applicable to all Viterbi decoders.

The ‘C’ simulator has also revealed that significant simplification is possible in the backtrace unit. Here, the conventional approach is to have multiple RAMs whose function rotates and which perform full backtrace on each block of RAM. Our simulations show that the great majority of winners emanate from the last winner and hence if this winning backtrace route is available, there is no need to perform backtrace at all in these cases. Furthermore, in the small number of cases that backtrace does need to be performed, the new backtrace route usually converges on the previous route within a few stages making full backtrace a rarity. Incorporating these features, only a single multi-sectioned RAM is required.

We have used the key findings identified above as a starting point for the design and have placed them in an asynchronous framework as described in the following sections which define the composition of the core decoder.

## 2.2 System Overview

The decoder consists of four blocks as shown in Figure 1. The encoder produces, for every input bit, two output bits whose values depend on the current input bit and (in this case) the previous six bits. The number of bits which affect the output is known as the constraint length (and often designated by the letter k), and in this case is 7. When the receiver passes these bits to the Viterbi decoder, it does so as three ‘soft bits’ for each transmitted bit. These three bits represent the receivers best estimate of the value that was transmitted, and its confidence in that estimate, based on what it actually received from the channel - ranging from ‘almost certainly a 1’ through ‘could be anything, but I shall choose a 1’ and ‘could be anything but I shall choose a 0’ to ‘almost certainly a 0’. These soft bits are compared with all the possible (ideal) values that can be encoded and the linear distance computed (or looked up) gives two separate three-bit quantities which are used as BMs by the BM unit.

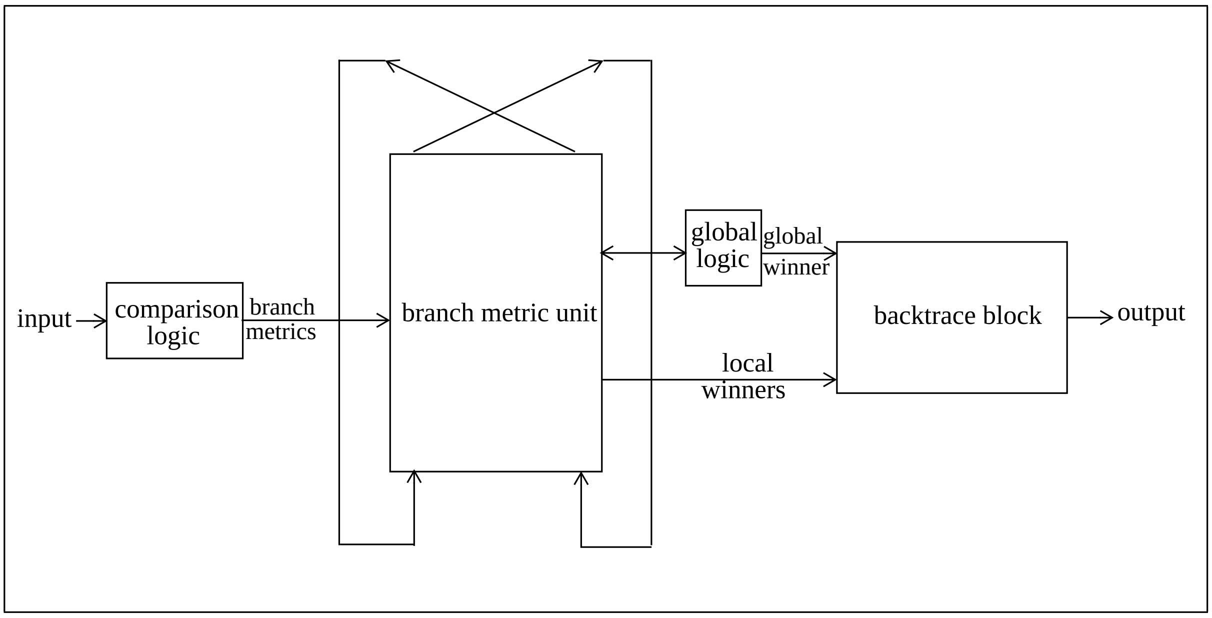

**Figure 1: Overview of the Viterbi Decoder**

The number of nodes in the branch metric unit is determined by the number of possible states of the encoding state machine, which in turn is determined by the number of bits used to create the output bits. The 7 bits used leads to a system with 64 states ($2^{7-1}$). In any one of these states, a received ‘0’ (coded as three bits) causes a transition to a new state while a received ‘1’ causes a transition to different state. This gives a total of 128 branches resulting from 64 states or nodes. Since the state transitions are predictable and constant, it is usual to represent them in the form of a trellis with the passage of time, as shown in Figure 2 for a typical 8-state system. From any state, the next state if an input ‘0’ is received is given by the lower path from the node while the next state is given by the upper path for a ‘1’ input. The transition pattern, of course, repeats at each time slot and thus the connection network does likewise. The pattern of interconnection is known as a butterfly network.

Each branch in the trellis is associated with a particular input encoding. In the BM unit, the existing PM at a state/node is added to the appropriate BM to obtain a new overall BM. Each output state can be reached from two branches and the one with the lower overall metric is selected as the new PM, and the higher valued branch is discarded; branches with the lower metric have a closer match between the received data and the encoded data represented by the branch. Conventionally, an 8-bit ACS circuit for each node is used to perform the node to branch additions and the selection of the lower overall weight. However, an asynchronous design has enabled the counts to be stored and manipulated as a series of events.

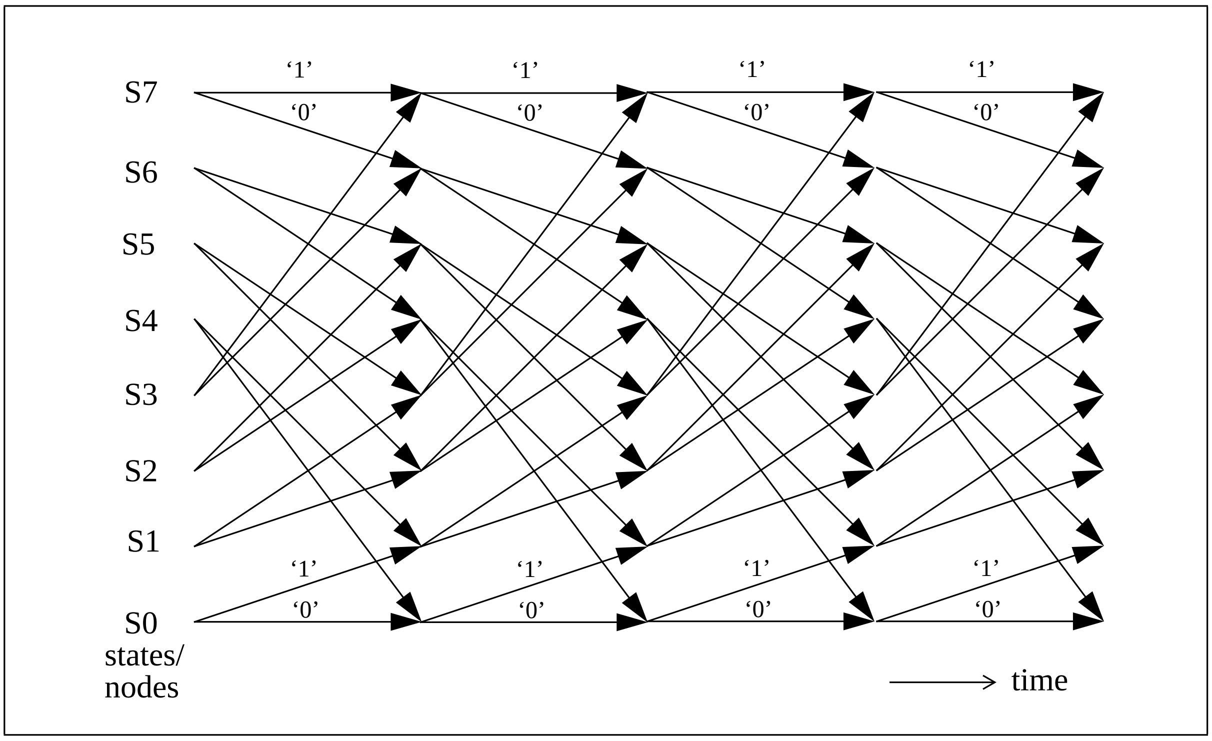

**Figure 2: Example Trellis**

Following this computation, the new PMs pass to the global logic so that the winning path may be identified and the most likely data bit be output. The overall winner is the node with the lowest overall metric; more than one node can exhibit this score. Often, the lowest metric will be 0, indicating a perfect match between received data and one particular data encoding. In the asynchronous system, renormalisation is performed whenever possible to keep the PMs and BMs as small as possible. Thus if all metric/state counts are greater than zero, global decrementing is performed until one or more of the nodes indicates a PM of 0. This has the effect of making it easy to identify the overall winner(s) while at the same time renormalising the PMs. One overall winner is identified from all potential overall winners (all those nodes for which PM=0). This could be a random choice but is most easily done using a priority encoder circuit.

The overall winner, encoded as a 6-bit value corresponding to which node produced it, plus the local winner information from the branches is passed to the backtrace block. The latter indicates whether the branch winner at a node (i.e. the one with the lower PM+BM) was from the node’s upper or lower branch. The information passed to the backtrace block enables that block to identify the most likely data bit and output it.

The backtrace block stores a history of results from the BM unit; it stores the identity of the winning branch rather than any metrics themselves. This history extends over many time slots, which allows a large accumulation of evidence for a given path, allowing high confidence in the output, and great error-correcting ability.

In a conventional decoder, the various possible paths are assumed to have converged on a single, correct path by a certain number of symbols after the data were transmitted, i.e. if the path is constructed backwards from the present for a given number of symbols, all possible starting points will converge on the same end point. This may not be the case however, and if it is not, it condemns the path unit to produce a number of erroneous bits. The single path from the end point of this backtrace is then used as the output. This backtracing is illustrated in Figure 3 for an 8-node 4-symbol example.

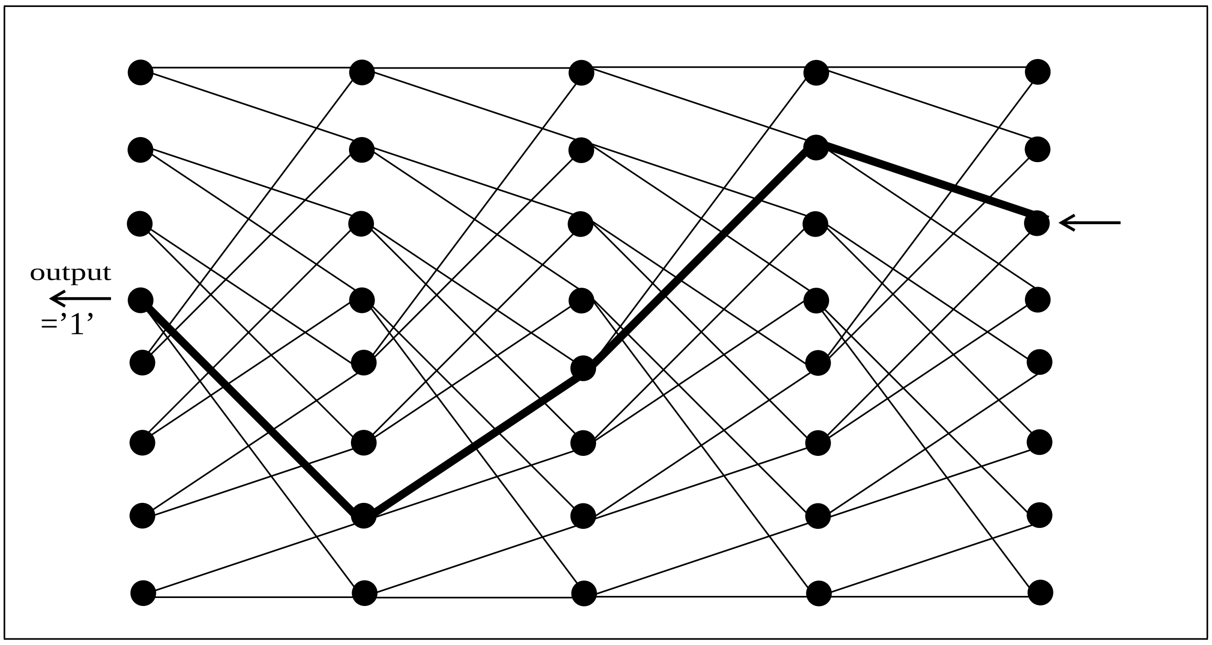

**Figure 3: Conventional Retracing**

We believe that the backtrace system can be greatly simplified without loss of accuracy since backtrace through the entire path is not required most of the time. Furthermore, this allows the storage requirements to be reduced to a single three-sectioned RAM. Here, the local branch winner information and the encoded overall winner is passed to the backtrace unit. The local branch winner information is added to the previous path history information in the RAM. The overall winner is added to the winner history section in the RAM which can be regarded as containing a good path. The global winner is also compared with the previous winner to see if it is its offspring. Most of the time, this proves to be the case and the ‘good’ path remains good and can be used to output the correct bit without the need for a compute intensive full path retrace.

If the new winner proves not to be a child of the previous winner, then a retrace using the path history is commenced with the new winning path being entered into the winner section of RAM as the backtrace proceeds. However, simulations show that in the event of this occurrence, the new path being retraced (and entered into the winner history) usually rejoins the previous good path after a few bits. Hence, further retrace is unnecessary and is not performed after the two paths merge; the output bit from the old path is still correct. A backtrace which has not converged after a few steps will generally require the establishment of an entirely new ‘good’ path; this should only ever occur in the presence of a prolonged burst error, at initialisation, or when there is a fault (as opposed to merely noise) with either the encoding, transmission or reception of the data.

Occasionally, a backtrace needs to be initiated which interrupts the ongoing backtrace. This would happen when a number of bits are in error that are in close proximity. In this case, backtrace will proceed a few steps, then be interrupted and restarted, then again interrupted, etc., until good data are received again. At that point the decoder will regain the correct path and trace back to correct the overwriting that has been done by the erroneous backtraces before it. The third section of RAM, the backtrace history, is then used to mark the furthest such interruption point; this marks a discontinuity in the path. For any subsequent retrace to converge with the existing path, it must pass the interruption point, thus ensuring that any winners stored in the backtrace path were generated either by the most recent convergent backtrace, or are marked as being incorrect.

## 2.3 Input Comparison

Each branch in the trellis is associated with a particular 6-bit input encoding representing two input bits (on orthogonal channels) which have been further soft encoded to three bits each. These ‘soft values’ must be used to generate the BMs. The soft values may be thought of as perturbations away from ‘perfect’ values - the noise introduced by the channel.

Since the same value must be compared to a perfect ‘1’ and a perfect ‘0’, we can say $r = p_0 + n_0$ and $r = p_1 - n_1$, therefore $p_0 + n_0 = p_1 - n_1$ and $p_1 - p_0 \equiv n_0 + n_1 \equiv 2^3 - 1$, where $r$ is the received value, $p_0$ and $p_1$ are the ‘perfect’ ‘0’ and ‘1’ values, and $n_0$ and $n_1$ are the noise that is added to the perfect values.

The actual BMs themselves though are functions of the noise values, i.e. $f(n_0)$ and $f(n_1)$. For a given value $r$, every BM will be either $f(n_0)$ or $f(n_1)$, so these BMs can be pre-decremented by $min(f(n_0), f(n_1))$. This leaves the BM increments ($a_0$ and $a_1$) being:

$$a_0 = f(n_0) - min(f(n_0), f(n_1)) = max(0, f(n_0) - f(n_1)) \text{ and}$$

$$a_1 = f(n_1) - min(f(n_0), f(n_1)) = max(f(n_1) - f(n_0), 0)$$

Further, it can be seen that if $f(x) \equiv x^2$, then:

$$f(n_0) - f(n_1) \equiv n_0^2 - n_1^2 \equiv (n_0 + n_1)(n_0 - n_1) \equiv (2^3 - 1)(n_0 - n_1)$$

and similarly for $f(n_0) - f(n_1)$. We can conclude that using the quadratic metric is no different from using the linear metric other than a scaling, which can be discarded, as it is applied equally to all BMs, and thus to all PMs.

This use of the linear metric and pre-decrement of the BMs results in the pairs of values for the BMs changing from [49,0], [36,1], [25,4], [16,9], [9,16], [4,25], [1,36], [0,49] to [7,0], [5,0], [3,0], [1,0], [0,1], [0,3], [0,5], [0,7]. This has the immediately obvious effect of halving the number of necessary additions, as half of all the PMs (for a particular channel) are always zero, and thus reducing the power consumption.

Another advantage is that, in order to prevent excessive arithmetic width being needed, the sum of the X and Y channels’ PMs are often scaled, in the case of Mitel’s design to 15, whereas using these methods the maximum value of the sum of the PMs is already only 14. The scaling step introduces some quantisation noise which degrades the performance of the algorithm.

In our design, in order to keep the PMs sizes smaller still, a further non-linear scaling is also applied, changing the BM pairs to [4,0], [3,0], [2,0], [1,0]. This non-linear scaling introduces a small discrepancy from the theoretical performance, but it is regained elsewhere in the design.

## 2.4 Branch Metric Unit

This features the logic to add BMs to existing PMs, to select the path into a node with the smaller metric and to then forward the local and overall winner results to the global logic and the backtrace block.

The regularity of the trellis diagram indicates that the logic can be designed in a similarly regular manner. In particular, the next state of a particular pair of nodes can be derived from just the current state of two nodes. This is illustrated in the trellis diagram of Figure 2 where the state of nodes 0 and 1 derive from nodes 0 and 4 (only), nodes 2 and 3 from nodes 1 and 5 etc. For this reason, it is only necessary to design the logic for a single node pair and this can then be replicated to provide the desired number of nodes in the system.

An overview of the design of an asynchronous node pair is given in Figure 4. The fundamental component in the asynchronous design are the up-down counters (called incdec on Figure 4) used to hold the system weights. A count is held as a set of events, one per stage, in a two-phase FIFO (First-In First-Out buffer); internally an event is stored as a transition in a stage. Incrementing the count by one requires an input handshake to the buffer to add an event while decrementing requires a handshake at the output to remove a transition.

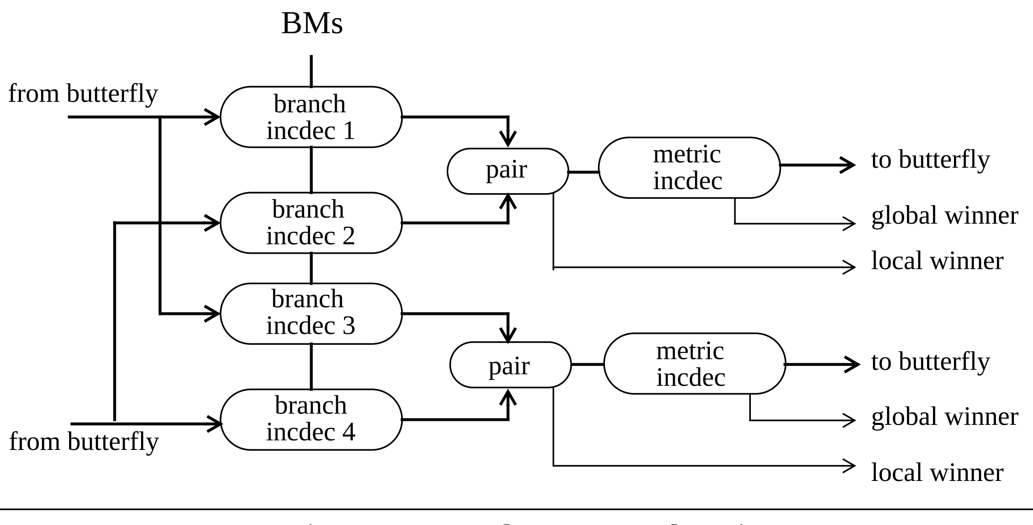

**Figure 4: Asynchronous Node Pair**

The node pair is organised as a set of serial communications between incdec components. The four incdecs on the left hold the PMs associated with a ‘0’ and ‘1’ data input for the two input nodes. The incdecs on the right hold the new metric count for the two output nodes. At the start of a time slot, the branch incdecs are parallel loaded with the BM resulting from the input comparison; this overwrites any existing count in these components. As previously stated, this value depends upon the linear distance between the two received input bits and their ideal ‘0’ and ‘1’ codings. The same BM is input to the incdecs 1 and 2 and this will be between 0 and 8; similarly with the two lower branch counters. The PMs connected in across the butterfly are now transferred serially into the BM incdecs, with the upper input node incrementing indecs 1 and 3 and the lower input node incrementing incdecs 2 and 4. Simulations show that there is little advantage to keeping BM values over around 6 as high scoring nodes almost never win. Thus a limit of 6 is imposed on the BM and PM incdec size and any further events that arrive once this limit is reached are discarded.

Once the PMs have been transferred in and the receiving PM incdec in the node pair is empty, then the determination of the new PM for the output node commences. Events in the two upper and two lower branch incdecs are paired and transferred as a single input event to their PM incdec. This continues until one BM incdec of the pair is empty, at which time the transfer is complete and the branch winner of the pair is identified. This local winner identity is sent to the backtrace unit to be entered in its path history.

Concurrent with the transfer of the BM incdec values to the PM incdecs, renormalisation of the PMs is performed; this also leads to the identification of the overall global winner. All PMs being non-zero causes all PMs to be decremented and this continues until one or more of the PMs reaches zero. The indication of global winner(s) is sent to the backtrace block for entry into the winner history. Renormalising the PMs ensures that the values to be handled by the system are minimised and reduces the amount of communication of events across the butterfly. Hence, there will be a power benefit from renormalisation.

## 2.5 Global Logic

This unit is considerably smaller in size than the other two and provides a link between the global winner information output by the BM unit and the history information recorded in the backtrace block. A block diagram of its operation is shown in Figure 5.

This unit detects when all BM pairs have started to generate PM events, and when they have all finished. Based on this information it controls the global decrementing of the PM incdecs, and when that is complete, manages the latching of the global and local winner signals and the recirculation of the PM values to the BMs

This completes the interaction with the BM unit for this input symbol. It also completes the processing of the PMs and these are now ready to be transferred into the BM incdecs as described above.

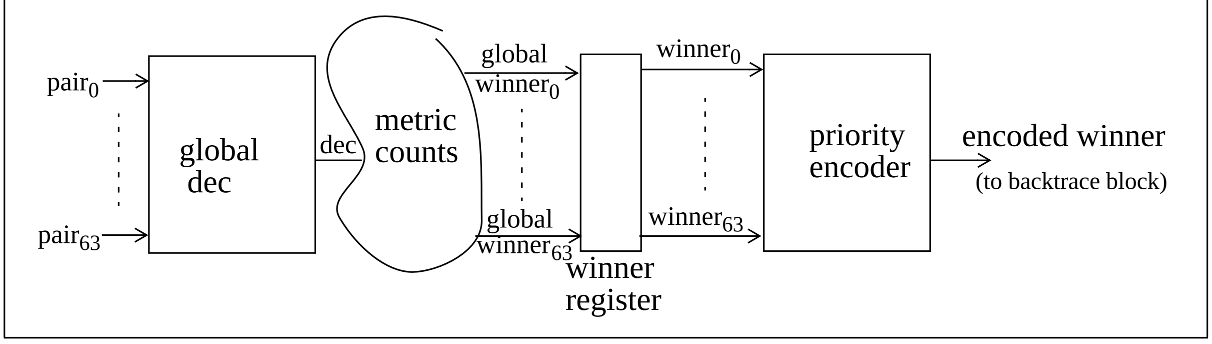

**Figure 5: Overview of Global Logic**

The winner register is now priority encoded to provide a 6-bit winner. This winner is passed to the backtrace block where it is written into the winner history section of the RAM and also used to determine if backtrace is required, as described below.

## 2.6 Backtrace

The core of the backtrace block is the three-sectioned RAM which holds the backtrace history, the winner history and the path history for the last 128 symbols received. The number of bits of past history kept makes it more likely that the output bit will be determined correctly. High levels of confidence in the output occur for path lengths which are four to five times the constraint length (7); a path length of 128 bits is used by Mitel.

A block diagram of the backtrace block is shown in Figure 6. The memory acts as a sliding window of length 128, by allowing the address to wrap around and there is no need to worry about the start point within the RAM. Using address pointer A which points to the current global winner position in memory, the 6-bit encoded winner from 128 bits ago (the tail end) is read to determine the output data bit; this can be derived from the winner's LSB (or MSB, depending on the direction the data passed through the coder). The 6-bit encoded global winner is then written into the 128-bit by 6-bit winner history section of the RAM using pointer A (at the head end) and at the same time, a bit corresponding to which branch contributed the local-winner for each of the 64 nodes is entered into the path history; this latter information will only be required if a path backtrace is required. The pointer is then updated to point to the next RAM location.

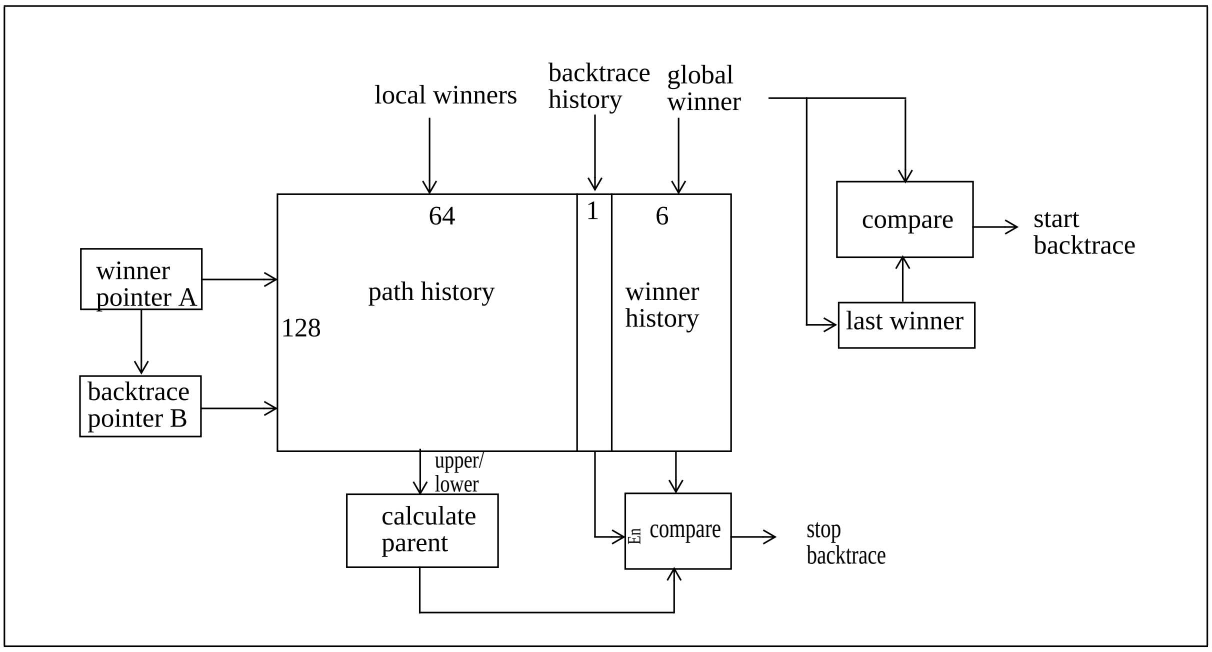

**Figure 6: backtrace Block**

As well as being added to the winner history RAM, the 6-bit encoded global winner is also compared with the encoded winner from the previous symbol which is held in a latch. Relationships between connecting nodes are fixed and simple to compute so that it easy to determine if the incoming winner is a continuation of the path from the last winner and hence whether backtrace is required. Having done the comparison, the new winner is entered into the latch (ready to be compared with the next winner). This completes the actions of updating the RAMs with information about the new winner and this sequence is repeated for each winner detected by the global logic regardless of whether a backtrace is or is not also being executed. Any backtrace is decoupled from this process.

If backtrace is not required, then the activity is confined to that described above. However, if the new winner is not descended from the previous winner, then backtrace is initiated. Here, the current operating position in the RAM (address pointer A) is copied to the backtrace pointer to act as the backtrace starting point. The path history is read from the RAM and this is used to determine the ancestor node in the new path. If this ancestor is coincident with the old path then the two paths are said to have converged, and backtrace may stop. If the old and new paths are not coincident, the new path is stored, and its ancestor determined, with this ancestor generation and path comparison continuing until convergence occurs. Simulation indicates that backtrace is infrequently required, and when it is performed, it typically continues for only a few stages.

Every time the head-end decides that a backtrace is required, backtrace will begin. However, if a backtrace is already in progress, this must be abandoned in a controlled manner. When a backtrace is abandoned, a flag is set to indicate that a backtrace was abandoned, and the last location in the winner history is marked in the backtrace history RAM. The backtrace is now restarted, but if the abandoned backtrace flag is set, convergence is not considered to have occurred until the marked point in the backtrace history has been passed. As the set bit in the backtrace history is passed, it, and the abandoned backtrace flag are cleared. This means that convergence may occur, or, if this backtrace is abandoned, there will only be one bit marked in the backtrace history RAM. If a backtrace is interrupted that has not passed a previously abandoned backtrace, the backtrace history RAM and flag are not modified.

This backtrace method suppresses noise effectively, as only the true data are capable of setting up a consistent, convergent path; the backtrace unit will always, end up with this data in the winner history in preference to any other data, as this is the only stream that will consistently have global winners that have a convergent, consistent descendancy path.

In the presence of a burst error, the non-consistent nature of the global winners will mean that a series of short backtraces will be started that corrupt the data in the winner history. This corruption is corrected once a correct, consistent stream of data are received. If the burst error continues too long, then corrupted data will be read by the tail end of the winner history, and a possibly incorrect bit output. At some point after this, a complete backtrace (i.e. one where the backtrace pointer becomes coincident with the tail-end pointer) is required.

## 2.7 Timing

The operation of the winner detection portion of the system can be pipelined to a depth of three, as indicated in Figure 7, with the third stage (global output and backtrace) being sub-pipelinable to any degree desired. Here, the first stage comprises the comparison between the received and transmitted possibilities and the parallel loading of the BMs, the second stage features the operation of the node pairs and the identification of the global winner while the third stage updates the history profile in the RAM and outputs a bit. The time for the first stage to operate depends on the time taken in the second stage since loading of the new branch weights cannot occur until the node pairs have been merged and the local winners are latched. In the second stage, the amount of computation will vary depending on the number of events to be moved between the incdec components in the BM unit. Hence, the time for this stage to operate will fluctuate.

The input data rate of the depuncture unit is 45MHz, and if a 1/2 rate code is used the depuncture unit will pass data to the Viterbi core at 45MSymbol/s. If another rate is used however, the symbol rate out of the depuncture can be as high as 78.75MSymbol/s (with a puncturing code that generates 7/8 rate data), allowing 12.6ns per symbol for these two stages.

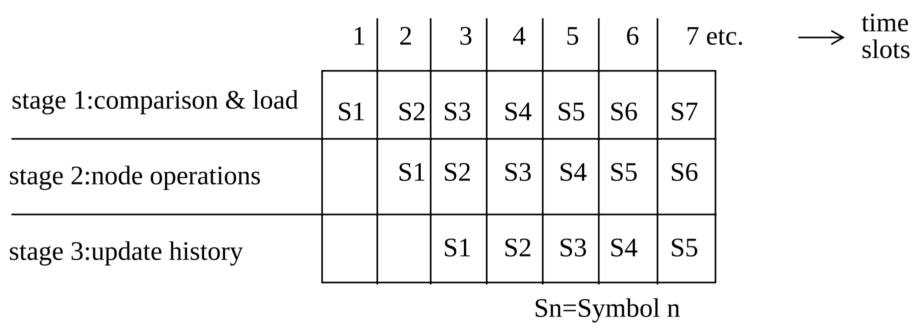

**Figure 7: Pipelined Operation: space-time diagram**

Within the BM and global units, there is a significant amount of concurrent operation. This is shown in Figure 8 where dotted arrows are use to emphasise the variable time at which the operations given occur. At the start of the slot the comparison values have been parallel loaded into the BMs as a set of events. The PMs are then added to these by serial event addition, emptying the PM incdecs. The time to complete this varies depending on the BM value and the incdec state at the start of the addition and the value in the PM incdec. As soon as both the metric transfers for a node are complete, their new metric count can be formed. Events from both branches are merged and transferred as an event from the BMs to the PM. Merging and transferring paired events into the metric count ceases when a BM incdec indicates that it is empty. Again the asynchronous nature of the design means that the starting time and duration for this operation within the slot is variable. Once merging has completed and the path to the metric counts is closed, the branch incdecs are free to be parallel loaded with the branch weights for the next symbol of input data.

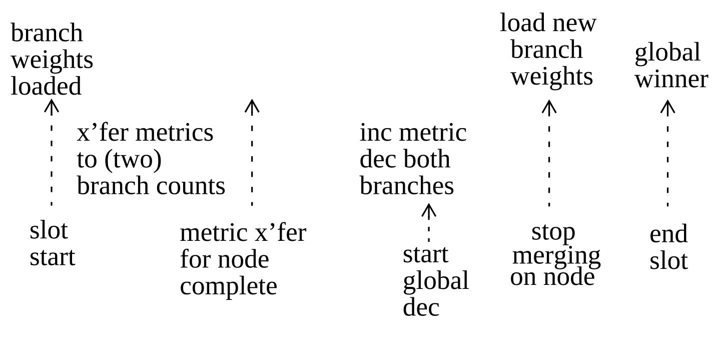

**Figure 8: Concurrency within the branch metric and global units**

Concurrent with this merging and transfer activity to the PM incdecs, the global decrement of metric counts can commence as soon as all PM indecs indicate they hold at least one event (i.e they are non-empty). This continues until one or more metric counts indicates an empty condition at which point the overall winner state is latched. Again, the exact point this occurs in the slot is unknown. Provided the next BMs have been loaded, the node pair operation is ready to restart with the addition of the PM values BMs. Most of the activity in the system occurs in this second stage and it is clearly going to prove the most difficult stage to design within the time constraint

In the backtrace unit, because there is no feedback path, no possibility of exceptions, and a high level of latency may be tolerated (the Mitel design has a latency of ~512 cycles), we can introduce as much pipelining as is convenient. Once the global and local winner data are latched, the global winner encoding, path history writing, global data output, path discontinuity determination and winner history writing may all be separate pipeline stages, with the backtrace being decoupled from this.

The speed of the backtrace unit may affect power consumption and performance. If the backtrace unit runs very quickly, it will be able to quickly build up an erroneous path and both that and the correction of the errors will cost energy, costing more energy the faster it runs. If the backtrace unit runs slowly however, its ability to correct an incorrect path generated by a prolonged error will not be as good as if it were run faster. In this case the tail-end could catch up with the error data in the winner history before the backtrace unit can correct it (once good data start arriving) for a shorter duration of burst error than if the unit ran faster. The desirable speed of the backtrace unit therefore depends critically on what form the errors take, and there should be an optimal point; more simulation and better noise models are required to establish what this point is. It may be possible to dynamically vary the backtrace speed to take account of the differing needs of complete and partial path reconstruction. This would require only the inclusion or exclusion of an additional delay element.

## 3 Power

Although detailed design has yet to be done on the asynchronous decoder, we believe from high level considerations that the proposed Viterbi decoder will consume significantly less power than a conventional decoder in the branch metric unit and in the backtrace block. In the branch metric unit, the reduction in number size together with normalisation has enabled a lot of design simplification in the node pair logic. Principally, the data path in the node pairs, which form the majority of the combinatorial logic required, has been reduced from conventional 8-bit parallel arithmetic processing to a serial-event bit stream. Given that up to 6 events are held, a significant power saving on the Mitel design can be reasonably expected in the BM and PM units.

The backtrace block should also yield a significant power saving compared with a conventional design. Here, backtrace is only undertaken if necessary and terminates as soon as possible. The asynchronous framework particularly supports this in that an idle state, where no power is dissipated, is automatically entered if the local handshakes are not generated. Thus idle states are both easy to enter and exit with no associated power or performance overhead. To give some indication of likely power benefits, assuming a SNR of 1dB, with a coding rate of 3/4, which represents a point fairly close to the worst-case from a specification viewpoint, 89% of the winners generated require no backtrace. In this case, backtrace need only be performed 11% of the time. However, backtrace is only performed until the new backtrace route merges with the old so that only as much computation as is required is performed. In practically all cases, merging occurs after, on average, 28 steps. The average number of backtrace steps per output bit for this case is therefore 11% of 28, i.e. 3 steps. Typical cases would be much lower than this. In the Mitel design each output bit requires 2 backtrace steps per output bit regardless of coding rate or noise level. It is expected that this decoder will, in most cases, be operated at a much greater SNR, a typical case would represent an improvement over the Mitel design.

## 4 Conclusions

An asynchronous design for a very different Viterbi decoder has been given This incorporates a number of novel features compared with a conventional decoder. Some of these features such as the use of a more accurate comparison algorithm, smaller count lengths and only retracing as far as required are applicable to all decoder designs and will save considerable power in any Viterbi decoder that adopts these. In addition, there is additional power saving and performance gain from adopting an asynchronous design approach. This is particularly apparent in the handling of the PMs and BMs which are held as a set of events, and in the backtrace block where a zero power idle state for the tracing back operation is expected to exist most of the time.

## 5 Acknowledgements

We would like to acknowledge the help of several Mitel employees in assisting in our understanding of the operation of their Viterbi decoder, in particular Keith Lyons. We would also like to thank various people from Queens University, Belfast for their attempts to (constructively) find errors with our reasoning regarding improvements to the performance of the Viterbi decoder.