An Asynchronous Viterbi Decoder

L. Brackenbury, M. Cumpstey, S. Furber, P. Riocreux
AMULET Group, Deptartment of Computer Science, University of Manchester, Oxford Road, M13 9PL

1 Introduction

1.1 Modelling

To aid the design and assist with our understanding of the Viterbi algorithm, a `C' simulator which modelled the decoder operation was developed. Parameters such as the backtrace length, the metric count limit, and the way the comparison between received and hypothesised uncorrupted transmitted data is computed could be varied so that their effect on decoder behaviour could be observed.

The simulator proved to be an invaluable aid to the design process. In particular, it confirmed that the branch metrics, which represent the difference between a received XY pair from the encoder and the four possible transmitted XY pairs, could be represented by linear metrics rather than a quadratic measure. Further experiments showed that the following simplification steps can be incorporated in the BMU while still meeting the desired bit error rates:

- the linear branch metrics can be decremented so that the smallest numbers are zero
- a non-linear scaling on the pre-decremented number n (by (n-1)/2) can be performed
- numbers can be capped at 6.

These measures have all been adopted in the asynchronous design. The need to deal with only small numbers where many of the numbers are zero minimises the amount switching activity when the numbers are manipulated.

Similarly, the simulator showed that large state metrics counts in the PMU were unnecessary if per-cycle normalisation to zero occurred when (occasionally) required and if numbers were capped at seven. Again, this has been adopted in the design. Comparable manipulations on the branch and state metrics have also been described by Clark and Cain [1]. The use of small numbers in the PMU with the use of zero in the typical case of no errors enables global and local winners to be identified in a simple way and these are then passed to the History Unit (HU).

The `C' simulator has also revealed that significant simplification is possible in the History Unit. The HU traditionally has been subject to far less investigation than the (physically) much larger PMU. Here, the conventional approach is to have multiple RAMs whose function rotates and which perform full backtrace on the RAM blocks. Our simulations show that the great majority of winners emanate from the last winner and hence if this winning path 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 the requirement for a full backtrace a rarity. Our results indicate that with a constraint length of seven, a RAM length of 40 timeslots is all that is required when the overall winner is known and therefore the starting point for any backtrace is identified. Reducing the storage requirement and the amount of memory accessing not only saves significant area and power but also reduces the latency through the system.

1.2 Applying Asynchronous Techniques to the Viterbi Design

Clearly while an asynchronous Viterbi design needs to conform to the external clock which supplies input and removes output data, internally, the operation can be timed by using the events that occur within a system rather than the supplied clock. Such clockless or asynchronous

design seeks to reduce power consumption by reducing the switched capacitance and switching activity compared with that in a synchronous system using a clock.

In a large clocked design, there is a high power cost associated with the distribution of the clock throughout the system. In addition, synchronous systems where the workload is variable inevitably have some overhead in switching between activity and idleness; in a synchronous design, incorporating clock gating or clock deactivation to save power when the system is idle increases system complexity and incurs a significant time delay in restarting the system, while allowing the system clock to run regardless of the system activity wastes power during inactive periods due to redundant switching activity.

In asynchronous or self-timed design on the other hand, the timing is governed by local handshakes between logic blocks. Since these blocks are normally adjacent, the associated capacitive load is small. Moreover, the system can be deactivated or restarted simply by halting or completing the handshake at an appropriate point. The ability to switch between idle and full activity without any sort of power or time penalty is an important, inherent characteristic of asynchronous systems. It is of particular significance to a Viterbi design which would expect to be incorporated in applications such as digital mobile phone handsets which do operate in these two modes.

An asynchronous approach leads to a modular design style. The protocol adopted for communication between blocks in this work is that of the bundled data interface. Here, the data (bundle) is accompanied by only two control wires which indicate the readiness of the source's data (request) and its acceptance by the receiving circuit (acknowledge) [2]. This design style enables the blocks to operate independently, at their natural speed, and where possible in parallel.

In the Viterbi decoder, the design partitions naturally into the BMU, PMU and HU blocks and the bundled interface is used to communicate between these blocks, see figure 1. Within these units the various parts are also free to operate asynchronously. This means that the nodes in the PMU can operate independently, and the backtrace process in the HU can run independently of and in parallel with the process of adding new items to the HU.

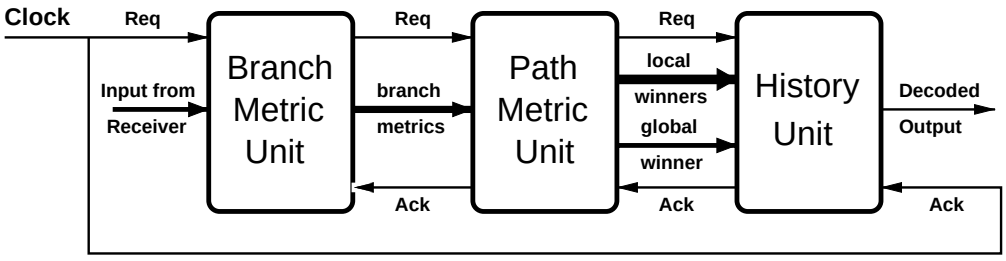

Figure1: Asynchronous Viterbi System

The processing delay within a block is indicated either by an explicit completion signal from the datapath of the block or by a matched delay in the control path. This form of timing means that delays are only dependent on the local maximum block delay rather than the global worst case delay needed in a synchronous system.

Finally, a further significant advantage of an asynchronous system is that the switching activity is spread over the period of activity. In a synchronous system, switching is concentrated around the clock edge leading to large current changes around the edge; this causes large radiation at the clock frequency and its multiples. However, in an asynchronous system, the levels of electromagnetic emission are far lower. The small levels of radiation from asynchronous designs is of particular relevance in systems, such as the Viterbi decoder, which deal with broadcast data.

The AMULET group's experience of designing large, complex, self-timed processor systems [3-5] has revealed that synchronous designs do not translate into an asynchronous structure in an obvious fashion; this arises from the fine-grained activity control required in the clockless design. Thus in undertaking an asynchronous Viterbi decoder design, there was a need to start afresh.

2. The Branch Metric and Path Metric Units

2.1 Serial Unary Arithmetic and the Incdec Units

Bearing in mind the simplicity of the arithmetic requirements and the opportunity for a radical approach to the problem we chose to use serial unary arithmetic. The arithmetic is unary because a number is stored as a string of events the value being the length of the string. It is serial because the numbers are moved and processed by shifting the events serially. The events are stored in specially designed registers where they are represented as a change in level between adjacent elements. For example in a register with 4 elements a value of 0 could be stored as 0000, a 1 could be 1110 and a 2 could be 1101. Since the absolute level is not important but merely the change, the inverted and/or reversed respresentations are equally valid, so the following are also all valid representations of 2: 1011, 0100, 0010 and 0110.

We call the basic storage register an 'incdec' unit derived from its two operations of incrementing and decrementing. It is a simple event FIFO (first in, first out) which accepts increment requests at the input end and decrement requests at the output. The incdec unit has a very simple implementation being constructed of 6 Muller C-gates so can hold a maximum value of 7 when the state of the request acknowledge lines are taken into account. The value held by the incdec unit is equal to the number of transitions on the outputs of the C-gates. Following an increment request the transition ripples through the C-gates and queues at the output end where it can be removed by a decrement request. Connecting the incdec output to the decrement request allows all the held transitions to be removed and the incdec unit will become empty. In addition to the input request lines the incdec units have output lines which indicate whether the unit is empty or full.

2.2 Branch Metric Unit

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 branch metrics. The soft values may be thought of as perturbations away from `perfect' values by the noise introduced by the channel.

Half rate (1/2) coding is achieved by transmitting two bits for every one source bit. The two bits are treated together and their branch metrics added before passing them on to the PMU. For higher coding rates (2/3 to 7/8) the transmitted data is punctured. The depuncturing operation is carried out before the data arrives at the BMU but this means that sometimes one of the pair of bits received carries no information so should make no contribution to the branch metric values.

The three soft bits represent the receiver's 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 either' to `almost certainly a 0'. The BMU converts the soft bit values to actual branch metrics of which two are required representing the probability that the input data originated from either a one or zero. The actual metric values has similar characteristics to -log(probability of correctness) so a zero value is used for the best estimate with positive values respresenting errors.

In some decoders, the Euclidian distance (proportional to the square of the difference between the soft value received and the ideal value, maximum value 49) is used as the branch metric. This can be simplified by performing a partial renormalisation and some scaling. Since it is only the difference in the metric values that is of significance, where both branch metrics are non-zero the BMU subtracts the smaller value from both. This allows linear values to be used instead of the more cumbersome quadratic values often used in other decoders. The values may then be scaled down linearly without loss of accuracy reducing the maximum value from 49 to 7. Our system introduces a small approximation to allow a greater scaling down so the maximum value is 4. Simulation studies have shown that the this approximation has a very small effect on decoding performance but allows a significant improvement in speed. Thus the two values required by the PMU are actually zero and the branch metric difference.

Since two bits are dealt with at the same time, the BMU adds the values for each of the two bits before passing the four sums to the PMU. It should be noted that because of the pre-normalisation carried out by the BMU at least one of the four sums will be zero. The PMU needs the branch metric values to be in a form suitable for the incdec units. Thus each value is first converted to a pattern of bits whose transition count is the value required. The BMU operation is summarised in Figure 2. The PMU parallel-loads the transition patterns produced by the BMU into the appropriate incdec units.

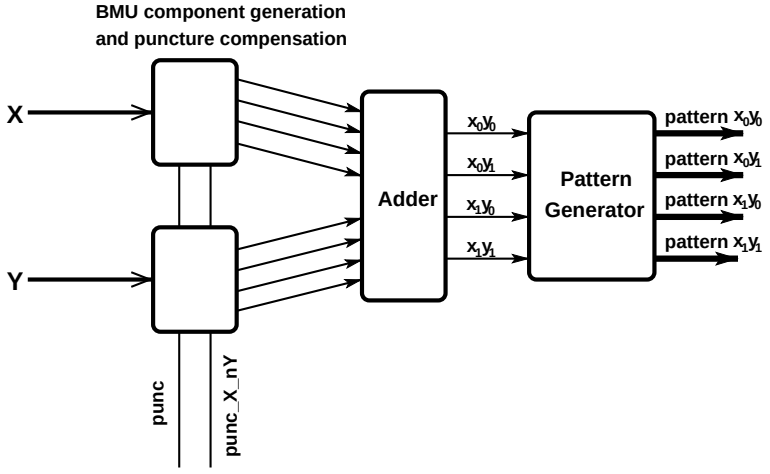

Figure 2: The Branch Metric Unit

2.3 Path Metric Unit

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 X 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 3. The fundamental component in the asynchronous design are the up-down counters which we call incdec units used to hold the system weights or state metrics. One of the attractions of using serial unary arithmetic is the simplicity of making the compare-select unit required by the PMU. This comprises a two input C-gate (labelled 'select' in Figure 2) whose inputs are fed by the two incdec units holding values to be compared. The C-gate produces an output transition when an input transition has been received on both its inputs. When the output transition is fed back to drive the decrement requests for both incdec units they both decrement continuously until one of them is empty then the process stops automatically. The number of output pulses produced is equal to the smaller of the two initial values and the empty indicator shows which was the smaller.

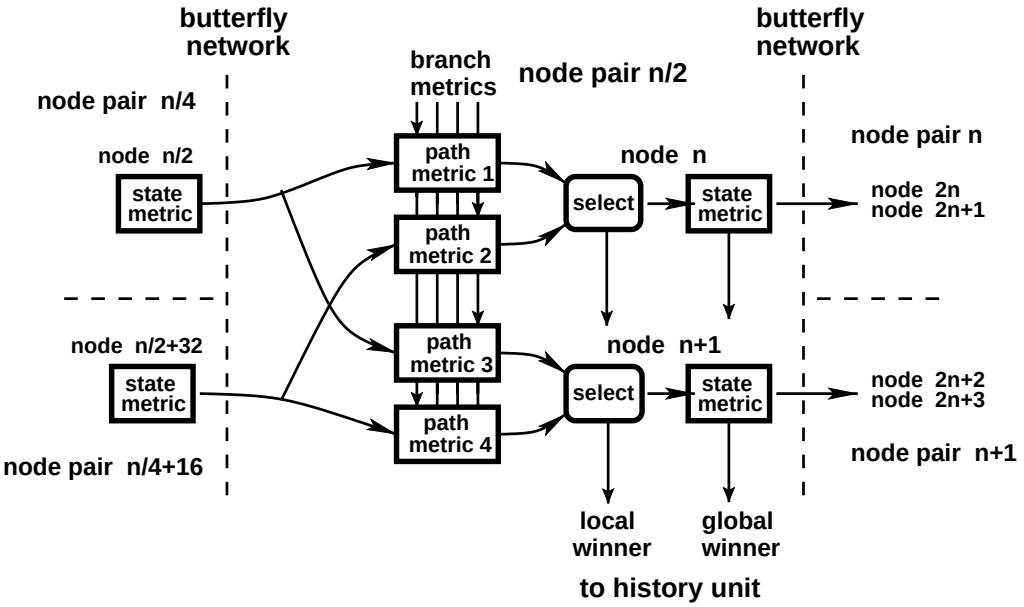

Figure 3. Asynchronous Node Pair

The node pair is organised as a set of serial communications between incdec units. The four incdecs on the left hold the path metrics associated with a `0' and `1' data input for the two input nodes. The incdecs on the right hold the new state metric count for the two output nodes. At the start of a time slot or cycle, the branch incdecs are parallel loaded with the branch metric 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 branch metric is input to the incdecs 1 and 4 and this will have a value between 0 and 8 but is limited to a maximum of 6 by the small arithmetic range adopted. The other two FIFOs are similarly loaded with another branch metric.

The state metrics connected across the butterfly are now transferred serially into the path metric 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 metric values over around 6 as high scoring nodes win extremely rarely. Thus a limit of 7 is imposed

on all the metrics and incdec size so any further events that arrive once this limit is reached are discarded.

Once the state metrics have been transferred and the receiving state metric incdec in the node pair is empty, then the determination of the new state metric for the output node commences. Events in the two upper and two lower path incdecs are paired and transferred as a single input event to their state metric incdec. This continues until one path metric incdec of the pair is empty, at which time the transfer is complete and the winning branch of the pair is identified. This local winner identity is sent to the history unit at the end of the cycle. If the new state metric is zero the node becomes a candidate to be the global winner over all the 64 nodes.
In the presence of very little noise, just one of the state metrics will be zero and the corresponding node is termed the global winner and its binary code represents the decoded signal.

If the received signal has a significant noise level it is possible that all state metrics are non-zero so no global winner exists. The PMU then renormalises the state metrics by finding the lowest value and subtracting it from all the state metrics. Thus the node with the minimum state metric is destined to become the global winner. In our current implementation the renormalisation is spread out over several cycles by limiting the metric decrementation to one per cycle. In this case it is possible that no global winner is found for a few cycles, but a global winner will be found when the renormalisation is complete. It was initially thought that this global decrement limit of one would simplify the design of the PMU and speed up its operation. We now believe that completing the full renormalisation in one cycle would be a better approach since the asynchronous design can handle varying cycle time well and it would lead to a simpler design of the HU and may improve the error rate performance.

After a noise burst and the subsequent renormalisation it is possible that more than one node can have a zero state metric. In this case it does not matter which one of the zero states is chosen as the global winner so the circuit choses one of them by means of a priority encoder.

Global synchronisation occurs once each cycle when all the local winners have been found and the state metric incdecs are holding their new values. A request for new branch metrics is sent to the BMU. The global winner, encoded as a 6-bit value corresponding to which node produced it, and the local winner information from the branches are passed on to the history unit for further processing.

3 Backtrace

In a conventional Viterbi design, backtrace is performed either by the very power- and area-costly register exchange method (where a complete path history is stored at each node and all these path histories are updated in each timeslot) or by backtrace. Because of the expense of register exchange, backtrace is normally preferred, but in its normal implementation this is also a fairly expensive method.

In conventional backtrace systems three (or more) blocks of memory are used. One is being used to store the most recently produced winning path data from the PMU, one is being used to construct the backtraced path and the third is having this backtraced path read from it for output. Because these actions are mutually exclusive, there must be separate spaces for them. The functions of the blocks are rotated at the end of each block of data, with the readout block having fresh data written to it, the fresh data block becomes the backtrace block and the backtrace block being used for readout.

In practice a fourth block is also often used and the backtrace path construction is performed in two stages. In the first stage the path is traced but not recorded. The location where the primary backtrace ends is then used as the starting point for the secondary backtrace, which is recorded. This is needed as, in the absence of overall winner information, the primary backtrace can only start at either a random or arbitrary, fixed location. This location is almost certainly not on the true path and so until the path that is being followed converges with the true path (usually within four or five constraint lengths), the backtraced path cannot be relied upon to be correct. By the end of the primary backtrace however, it is fairly certain that the two paths will have converged, so using the end point of the primary as the starting point of the secondary gives a high degree of confidence in the whole of the recorded path stored in the secondary. Figure 4 indicates this conventional method of backtracing.

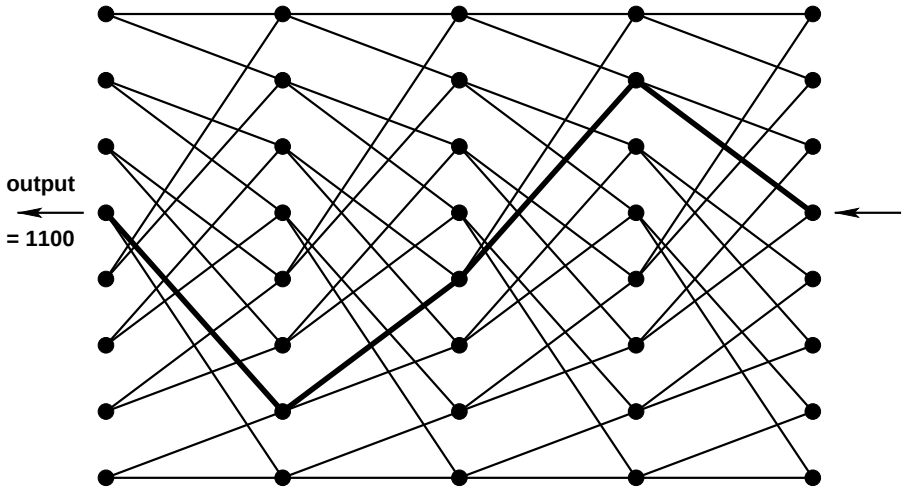

Figure 4: Conventional backtracing

This is however rather energetically wasteful. As the primary backtrace has, in practice, converged within four or five constraint lengths - a length typically much shorter than the size of the memory blocks - most of the path calculated in the primary is in fact correct, and the parts that represent the oldest data are the most likely to be correct. This oldest data is also that closest to being output and that which is most likely to be wrong (the most recently recorded data) is furthest from being output. Where the starting location for the backtrace is known, these first few errors are also in general avoidable, for exactly the reason that the first few bits of the secondary backtrace path are reliable where those of the primary are not.

This suggests a new architecture for the backtrace unit based on a single, revolving block of memory that is logically connected in a loop with a breakage that rotates around the loop - effectively a sliding window. The newest data are written to one side of the breakage and the oldest data read from the other side of the breakage. After each read/write cycle the breakage moves one step, making what was the oldest slot the newest and all others one timeslot 'older'. In addition to the winning path data that are conventionally recorded, the overall winner is also stored. The output data is simply the MSB of this overall winner and so need not be recorded explicitly. This method allows the first and third stages of the backtrace process to be merged into one block of memory, albeit slightly larger ({n+log_2(n)}/n times) than those in the conventional architecture.

The backtrace path construction is also merged in simply. In an error-free data stream, each overall winning path in the PMU will have a simple-to-compute relationship with the overall winner in the previous cycle. If this relationship between them holds then the MSBs of the overall winners represent the backtrace path that would be constructed if backtrace were to be performed. If this relationship continues to hold then no backtrace at all is necessary.

If an error occurs, then this relationship will not hold and backtrace will be required. In essence the backtrace path construction process is very similar to that in a conventional architecture. The overall winner is used to look up which overall winner should have preceded it (as opposed to the one that did) based on the path winner information and this is recorded as the overall winner for the preceding timestep, overwriting what was originally there. The backtrace process then moves its point of reference to the preceding timestep and the process is repeated.

The big difference compared to a conventional architecture is that once the current (as seen by the backtrace process) overall winner is related to its predecessor, the backtrace process will cease (called backtrace retirement). This can be done because, as in the error-free example above, any further backtrace will not have any effect and so is redundant. This mechanism means that the whole backtrace process is still confined to a single memory block, and that backtrace is confined to the times when performing it would actually achieve something. In addition, this mechanism also results in a large reduction in the latency of the system as for the same level of confidence in the backtrace results (a factor of three or more compared to our reference system).

The backtrace process, being divorced from either the writing of new data or the reading of old data once it has been launched, can proceed at whatever speed is desired. If a slow backtrace is used then in the event of an error erroneous data will be placed in the overall winner section of the memory at a slower rate. The total number of erroneous steps will be unchanged however, as all the backtraces run at the same rate and even if later backtraces did catch up with earlier ones, there is no mechanism for their merging.

If a long sequence of errors occurs so that a large proportion of the sliding window contains erroneous data, a slow backtrace would reduce the amount of the sliding window that could be in error but still correctable before the erroneous slots are read for output. In practice this is unlikely to occur, but in the absence of a penalty for fast backtrace it should be preferred as it will reduce the activity overall by minimising interaction between backtraces. I should however be remembered that the speed will have little impact on the error-correcting performance or energy consumed by the backtrace.

When multiple successive errors occur the behaviour becomes a little more complicated. In a simple scheme, all backtrace that is to occur would be completed before the arrival of the data for a new timestep, so that if that data required a backtrace it could be carried out. This is unrealistic however, as backtrace can run for several tens of timeslots and most timeslots do not see the initiation of a backtrace, even with very low SNRs. Instead one of several more complex strategies may be adopted.

In the simplest of these strategies, a record is kept of where in the historical data the backtrace has reached. If the data in a new timeslot require a backtrace to be initiated, the location of the existing backtrace is recorded and it is abandoned. The new backtrace then begins, but cannot be retired until it has passed the point at which the previous one was abandoned. If this backtrace is itself interrupted, the abandonment point is only recorded if it has already passed the previous

abandonment point. This ensures that data which may have been corrupted by the backtrace process is corrected. This approach has its own problems - if a series of errors occurs, then the abandonment point may become so far from the new-data end of the memory that there is little likelihood of any backtrace proceeding as far as the abandonment marker (and this likelihood decreasing all the time).

Instead it is possible to record the location of abandonments in another memory, and when no backtrace is either to be initiated or in progress, an abandonment location may be drawn from this memory and used to resume an abandoned backtrace. This overcomes the problem of the length of backtrace growing to a point where it is unlikely to complete, but adds a new problem of managing this memory, and ensuring that abandonment locations that have been overtaken by events (the break in the circular memory has passed over them) are not used.

A third option, used in this self-timed implementation is to allow all backtraces to continue in all circumstances (other than legitimate retirement). This removes the problems of both of the previous approaches, at the expense of having multiple, concurrent backtraces in operation. At first sight this approach appears to be extremely difficult to manage, but this is overcome simply by implementing a local control strategy, with each location in the rolling window having its own control system. This is illustrated in Figure 5.

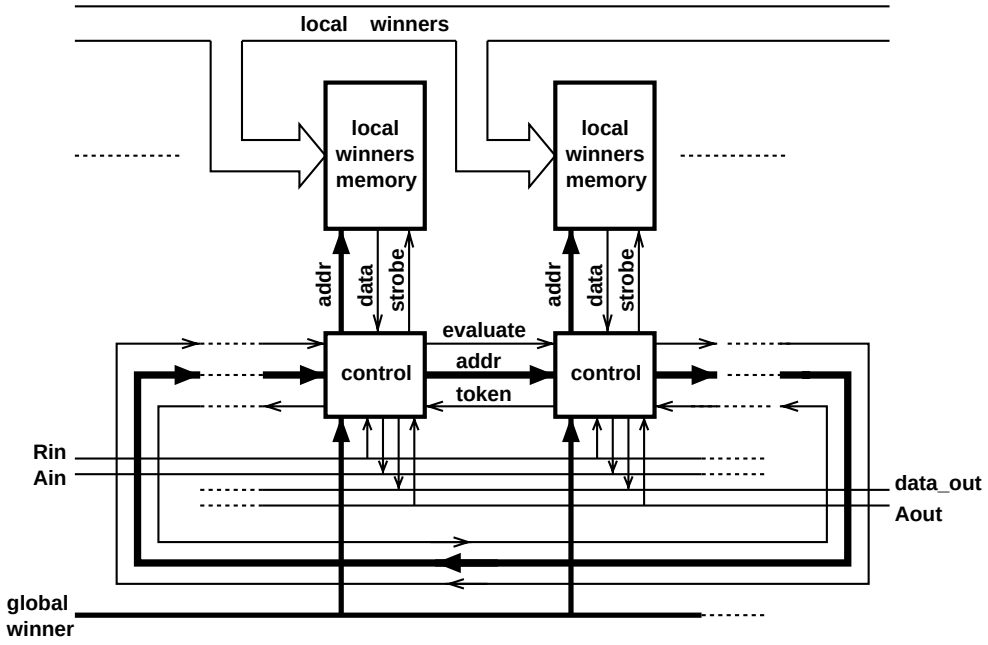

Figure 5: The History Unit

When the PMU indicates that it has a set of local and overall winners to pass to the HU, the timeslot in the HU that possesses the timeslot token stores that data in its local and overall winner memories and outputs the MSB of the overall winner memory from its upstream neighbour as the output data (not illustrated). Then the control that is local to that timeslot decides whether to initiate a backtrace or not. If not, then it simply passes on to its upstream neighbour the 'token' that indicates that it is to receive the next timeslot. If a backtrace is required, then it passes a backtrace token on to its downstream neighbour and then passes its timeslot token to its upstream neighbour. When a timeslot without the timeslot token receives a backtrace token, it decides if the backtrace should be retired and either does so or evaluates the

backtrace and passes the token on to its downstream neighbour. Thus it can be seen that the backtrace process is occurring in the same manner as with the other strategies.

This method has its own drawback: the memory in this system cannot be a single block of conventional RAM, as all of the timeslots could in theory be accessing the stored data simultaneously if each of them were performing a backtrace. In our design an array of latches is used, but a small RAM local to each timeslot could be used instead. This disadvantage is however outweighed by the relative simplicity of managing the backtrace process itself. It also has the advantage that as long as one step of the backtrace requires less time than the average number of timeslots interval between errors the system will not become blocked by uncompleted backtraces.

Another factor may be taken into account when deciding whether to initiate a backtrace: whether the overall winner had a zero SM or not. If the overall winner had a non-zero metric, then there must have been an error in the channel, as this is the only way that the overall winner may have a non-zero SM. In this case, it may be chosen to avoid initiating a backtrace as there is a chance that the wrong overall winner has been chosen. This has the advantage of setting up fewer partially incorrect path segments, but adds to the complexity of the system. In the current design, with only a single global decrement of SMs, this system is needed as when a small error forces there to be no node with a zero SM the overall winner identity that is encoded is fixed and has no relationship with the smallest SM. In a proposed modified design a correct encoding of the overall winner is guaranteed and this mechanism is unnecessary.

Using these types of backtrace, the latency can be reduced further, as backtrace on the oldest timeslots has occurred when they were the newest timeslots, so the number of timeslots that must be stored is the sum of the longest backtrace that it is wished to cater for plus the number of timeslots that would elapse during the performance of that backtrace.

4. Conclusions

The simulator has enabled many simplifications to be made to the Viterbi design, without affecting error performance. These simplifications should have a dramatic effect on the power consumed and the area occupied. Power savings in the BMU and PMU are principally due to the very small numbers handled with normalisation to zero performed whenever possible and the capping of numbers together with the use of scaled linear metrics. The power savings in the HU result from the use of a much smaller memory and only doing backtrace operations when necessary; additionally, any backtrace which is done is only performed as far as required. It should be noted that these design features for less power identified by the simulator are applicable to Viterbi decoders, regardless of their timing model.

The choice of an asynchronous design style can also be expected to contribute to a low power Viterbi design. In particular, the choice of serial unary arithmetic for the values in the BMU and PMU should yield power- and area-efficient arithmetic since less transitions and load will arise in the datapath. The ability for asynchronous circuits to operate independently is exploited in the design to enable as much of the system as possible to operate concurrently. This is of especial consequence in the HU, where the decoupling of the backtrace from the forward path enables the backtrace to run as fast as the system will allow while the rate of adding new history is still essentially governed by the clock rate. In a synchronous system, any backtrace would inevitably be tied to the system clock making it difficult to backtrace at more than 1 bit per clock.

Finally, the inherent ability of asynchronous systems to switch almost instantaneously between full and negligible activity plus the low radiation emitted should demonstrate further significant advantages of opting for an asynchronous approach for this design.

5. References

1. G. C. Clark Jr. & J. B. Cain, `Error-Correcting Coding for Digital Communication', Plenum, New York, 1981.

2. I. E. Sutherland, `Micropipelines', Communications of the ACM, Vol. 32, No. 6, pp. 720-738, 1989.

3. J. V. Woods, P. Day, S. B. Furber, J. D. Garside, N. C. Paver, S. Temple, `AMULET1:An Asynchronous ARM Microprocessor', IEEE Transactions on Computers, Vol. 46, No. 4, pp. 385-398, 1997.

4. S. B. Furber, J. D. Garside, P. Riocreux, S. Temple, P. Day, J. Liu, N. C. Paver, `AMULET2e:An Asynchronous Embedded Controller, Proceedings of the IEEE, Vol. 87, No. 2, pp.243-256, 1999

5. J. D. Garside, W. J. Bainbridge, A. Bardsley, D. M. Clark, D. A. Edwards, S. B. Furber, J. Liu, D. W. Lloyd, S. Mohammadi, J. S. Pepper, O. Petlin, S. Temple, J. V. Woods, `AMULET3i - an Asynchronous System-on-Chip, Proceedings Async2000, pp. 162-175, IEEE Computer Society Press, April 2000.