A FPGA Implementation of a Parallel Viterbi Decoder for Block Cyclic and Convolution Codes.

J.S.Reeve and K. Amarasinghe
School of Electronics and Computer Science,
University of Southampton, Southampton SO17 1BJ, UK
email: jsr@soton.ac.uk

***Abstract*— We present a parallel version of Viterbi's decoding procedure, for which we are able to demonstrate that the resultant task graph has restricted complexity in that the number of communications to or from and processor cannot exceed 4 for BCH codes. The resulting algorithm works in lock step making it suitable for implementation on a systolic processor array, which we have implemented on a field programmable gate array and demonstrate the perfect scaling of the algorithm for two exemplar BCH codes. The parallelisation strategy is applicable to all cyclic codes and convolution codes. We also present a novel method for generating the state transition diagrams for these codes.**

I. Introduction

The Viterbi algorithm[10] was developed as an asymptotically optimal decoding algorithm for convolution codes. It is nowadays commonly also used for decoding block codes since the usual[2], [3] algebraic decoding methods are not always readily adaptable for soft decoding. It is also used in soft decision decoding in which the demodulator returns two numbers that relate to the likelihood that the data bit is a 0 or a 1. In these circumstances Viterbi decoding of Bose-Chaudhuri-Hocquenghem (BCH)[4], [6] and convolution codes is found to be efficient and robust. In this paper we describe a parallel Viterbi algorithm for hard decision decoding, in which data bits are delivered as either 0 or 1 only, as the adaption to soft decision decoding is trivial.

Although the Viterbi algorithm is simple it requires $O(2^{n-k})$ words of memory, where n is the length of the code words and k is the message length, so that $n-k$ is the number of appended parity bits. In practical situations it is desirable to select codes with the highest minimum Hamming distance that can be decoded within a specified time and an increased minimum Hamming distance d_{min} implies an increased number of parity bits. Our parallel Viterbi decoder necessarily distributes the memory required evenly among processing elements.

We describe our method for BCH codes as these are slightly more complex than the convolution decoding, principally because of the presence of feedback in the generating shift register. Convolution codes can be treated in the same way. This work generalises and implements work first reported in [8]. In that paper we express the Viterbi algorithm as a matrix-vector reduction in which multiplication is replaced by addition and addition by minimisation. This resulted in an algorithm closely related to the parallelisation reported by Kumar *et al.* [7] of Floyd's [5] minimum cost path algorithm. Our new approach allows parallel algorithms to be designed for all the block cyclic codes and proves that the valence of each node of the resulting task graph is restricted, thus limiting the connectivity complexity of the resulting hardware. This is accomplished by the observation that the state transition diagram can be generated from a simple rule, described in the next section, that we use to construct the parallel algorithm in such a way as to manifestly restrict the task graphs connectivity.

II. The Sequential Viterbi Algorithm as a State Transition Machine

BCH codes are a class of cyclic codes that append $n-k$ parity bits to a message of k bits so that each code word is n bits long. The code parameters (n, k, d_{min}) are of the form $n = 2^m - 1$, $n-k \leq mt$ and the minimum Hamming distance is $d_{min} \leq 2t+1$. The codes are specified by their generator polynomials in $GF(2)$ which has the general form $G(D) = g_0 + g_1 D + \cdots + g_{n-k} D^{n-k}$. The parity bits appended to the message in the systematic generation of the codeword corresponding to the message polynomial $M(D)$ are the coefficients of the remainder of $\frac{D^k M(D)}{G(D)}$. This encoding process is usually implemented by a shift register. The general setup is shown in figure 1 in which switches s_0 and s_1 are closed and s_2 open for the first k cycles while the message m_k of length k is input. For the next $n-k$ cycles switch s_2 is closed and switches s_0 and s_1 are open.

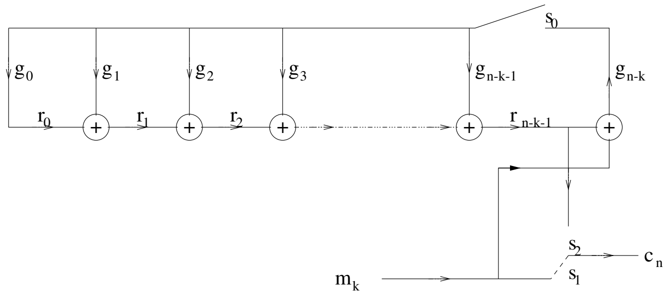

Fig. 1. Shift Register Encoding using BCH Codes

In our implementation of the Viterbi decoding scheme we regard the shift register encoder as a state transition machine in which the state R, is the base 10 number that represents the bit pattern $\{r_0 r_1 \ldots r_{n-k-1}\}$, and the number of possible states is $|R| = 2^{n-k}$. In the following description, the G is the generator polynomial represented as a binary number, $Q = 2^{n-k-1}$, and $g = G \oplus Q$ is the generator with the top bit set to 0. The machine changes to a particular state depending on the top bit of the register, that is, on whether R is greater or less than $Q = 2^{n-k-1}$, and on the value of the next input bit of the message. If the register is in state R^i after the ith message bit has been input, then we have $R^{i+1} = 2R^i$ if either $R^i < Q$ and the message bit is 0, or $R \geq Q$ and the message bit is 1. Conversely, $R^{i+1} = (2R^i) \oplus g$ if either $R^i < Q$ and the message bit is 1, or $R \geq Q$ and the message bit is 0. This state transition template is represented in figure 2, where the input bits are marked against the arrows.

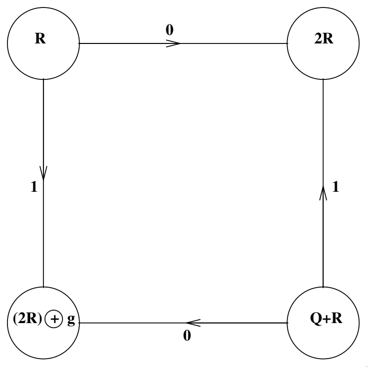

Fig. 2. The building block for the state transition diagram for all cyclic codes.

Using this we can generate the state transition diagram for the BCH(7,4,3) code, which has generator $G = 013$ in octal, corresponding to a generator polynomial of $G(D) = 1 + D + D^3$, and $Q = 2^{n-k-1} = 4$, so that $g = G \oplus Q = 03$. We begin the process by starting with the state $R = 0$ in diagram 2 and observing that this induces the transition to state $R = 0$ when a 0 is input and to state $(2R) \oplus g = 0 \oplus 3 = 3$ when a 1 is input. We can then put $R = 1$ in the generator diagram and follow that to state 2 when a 0 is input and to state $2 \oplus 3 = 1$ when a 1 is input. Progressing in this way we find the resulting state transition diagram as is shown in figure 3.

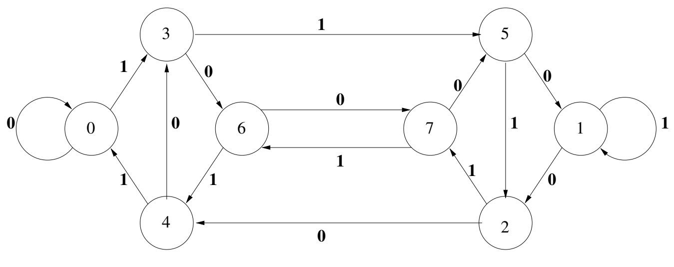

Fig. 3. The state transition diagram for the BCH(7,4,3) code.

As an example, for the message sequence $\{1101\}$, the state progresses in the sequence $\{0 \xrightarrow{1} 3 \xrightarrow{1} 5 \xrightarrow{0} 1 \xrightarrow{1} 1\}$ from state 0 to state 1. The parity bits, which are the binary representation of the state after the message has been input, are thus $\{001\}$.

The sequential Viterbi decoding algorithm is best illustrated by considering the state transition diagram for code word generation. For each possible state R we keep a weight ω_R^i that is the least Hamming distance of the codeword that is legitimately generated via state R for the ith input bit, from the actual input sequence. For example if we traverse the state transition diagram with the input sequence $\{1101001\}$ we arrive back at state 0 with $\omega_0^7 = 0$ and the input sequence is a codeword. Error correction, however, requires that we record all paths through the diagram that have a Hamming distance less than or equal to the minimum code separation, d_{min}. It is clear from figure 2, that we can find the Hamming weight of an input sequence by iterating the relations:-

$$\omega_{2R}^i = \min\{\omega_R^{i-1} + m_i, \omega_{Q+R}^{i-1} + 1 - m_i\} \tag{1}$$

$$\omega_{(2R)\oplus g}^i = \min\{\omega_R^{i-1} + 1 - m_i, \omega_{Q+R}^{i-1} + m_i\} \tag{2}$$

where $m_i \in \{0, 1\}$ is the value of the ith message bit, for $i = 0, 1, \ldots n - 1$. If there isn't a clear minimum, then an arbitrary choice of correct input bit is made.
By successively applying equations 1 and 2 for $i = 0, 1, \ldots n - 1$, and using the initial values $\omega_0^{-1} = 0$ and $\omega_R^{-1} = d_{min}$, for $R = 1, 2, \ldots 2^{n-k}$ we arrive at a value for ω_0^{n-1}. If ω_0^{n-1} is less than half the minimum codeword separation then the message can be corrected.

III. The Parallel Viterbi Algorithm

Our parallelisation strategy is to distribute the states evenly among the available processing elements in such a way that the resultant task graph, that is the directed graph of the processing elements as nodes and the data paths as arcs, has in-degree and out-degree both restricted to 4. This, together with the fact that the work done on each processor is the same means that our algorithm can be implemented in hardware clocked on the incoming bit stream.

We partition the states among 2^m processors so that there are $P = 2^{n-k-m}$ states on each processor. From the generator graph of figure 2 for the state transition diagram, we see that states R and $R + Q$ are required to update states $2R$ and $(2R) \oplus g$, so by placing states R and $R + Q$ on the same processor we need to send only one message (containing the Hamming weights and paths from states R and $R + Q$) to each of the processors handling the states $2R$ and $(2R) \oplus g$. Now let $L = 2^{n-m-k-1}$, the number of states on each processor with state number $s < Q$ and assuming for now that $L \geq 2$ and assign processor p the states s_p and $s_p + Q$ such that $s_p = pL \ldots (p+1)L - 1$. To update an even state, $2s$, requires the previous weights of states $e = s$ and $e + Q$, and to update an odd state, $2s + 1$, requires the previous weights of states $o = \frac{(2s+1)\oplus g}{2}$ and $o + Q$. State s is on processor $\lfloor \frac{s}{L} \rfloor$, when $s < Q$.

The even states, with state numbers less than Q, on processor p, require the previous weights of the states $e_{<Q}$, given

by

$$e_p = \frac{pL}{2}, \frac{pL}{2}+1, \frac{pL}{2}+2, \ldots, \frac{(p+1)L}{2}-1$$

and $e_p + Q$, which all lie on processor $\lfloor \frac{p}{2} \rfloor$. The even states, with state numbers greater than Q, require the previous weights of the states $e_{>Q}$ given by

$$e_{p+Q/2} = \frac{pL+Q}{2}, \frac{pL+Q}{2}+1, \ldots, \frac{(p+1)L+Q}{2}-1$$

and $e_{p+Q/2} + Q$, which all lie on processor $\lfloor \frac{p}{2} \rfloor + \frac{P}{2}$.

The odd states, with state numbers less than Q, on processor p require the previous weights of the states $o_<$ given by

$$o_p = \frac{(pL+1)\oplus g}{2}, \frac{(pL+3)\oplus g}{2}, \ldots, \frac{((p+1)L-1)\oplus g}{2}$$

and $o_p + Q$, which all lie on processor $\lfloor \frac{(pL+1)\oplus g}{2L} \rfloor$. The odd states, with state numbers greater than Q, require the previous weights of the states $o_>$ given by

$$o_{p+Q/2} = \frac{(pL+Q+1)\oplus g}{2}, \frac{(pL+Q+3)\oplus g}{2}, \ldots, \frac{((p+1)L+Q-1)\oplus g}{2}$$

and $o_{p+Q/2} + Q$, which all lie on processor $\lfloor \frac{(pL+Q+1)\oplus g}{2L} \rfloor$.

The above result, that the states o_p (or the states $o_{p+Q/2}$) all lie on the same processor, depends on the fact that for a continuous set of numbers $s = \{0, 1, \ldots, n\}$, $s \oplus g$ covers a set of contiguous numbers. For instance if $s = \{0, 1, 2, 3, 4, 5, 6, 7\}$ and $g = 5$, then $s \oplus g = \{5, 4, 7, 6, 1, 0, 3, 2\}$, the same set of numbers in a different order.

Hence when $L > 2$ each processor requires at most inputs from 4 different processors. Similarly the number of processors requiring information from any processor cannot exceed 4. When $L = 1$ there is only a single state with state number $< Q$ on each processor and the in and out degree of the task graph are 2. When the number of processors is the same as the number of states then our algorithm reverts to a direct implementation of the state transition diagram for that code.

As an example consider the partitioning of the 16 states of BCH $(15, 11, 3)$ code among P=4 processors. The state transition machine for this code is shown in figure 4.

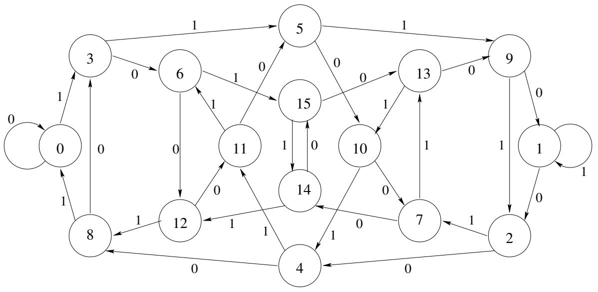

Fig. 4. The state transition diagram for the BCH(15,11,3) code.

For this code, Q=8 and L=2. The partitioning of states among processors is shown in tableI where we have used the notation $p(e_{<Q})$ to designate the processor that has the set of states $e_{<Q}$.

p	s_p	$e_{<Q}$	$p(e_{<Q})$	$e_{>Q}$	$p(e_{>Q})$
0	0,8,1,9	0,8	0	4,12	2
1	2,10,3,11	1,9	0	5,13	2
2	4,12,5,13	2,10	1	6,14	3
3	6,14,7,15	3,11	1	7,15	3
p	s_p	$o_{<Q}$	$p(o_{<Q})$	$o_{>Q}$	$p(o_{>Q})$
0	0,8,1,9	1,9	0	5,13	2
1	2,10,3,11	0,8	0,	4,12	2
2	4,12,5,13	3,11	1	7,15	3
3	6,14,7,15	2,10	1	6,14	3

TABLE I

THE STATE PARTITIONING TABLE FOR THE BCH (15,11,3) CODE.

The task graph for the $(31, 16, 7)$ BCH code on 8 processors is shown in figure 6. In the task graph, circles represent processors and the arcs represent the transition of the paths and their Hamming weights. This parallel decoding machine operates in lock step for n cycles for a (n, k, d_{min}) code. For BCH codes the in-degree (and out-degree) of the task graph never exceeds 4, whereas for convolution codes the valence depends on the number, k, of input bits.

For convolution codes with $k = 1$ input bits and any constraint length has a state transition generator of the same form as the cyclic codes as shown in figure5.

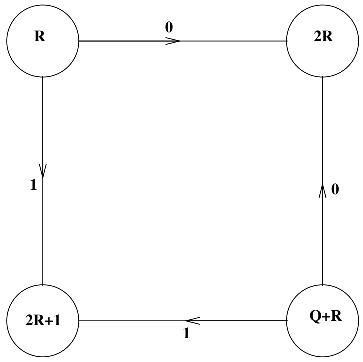

Fig. 5. The building block for the state transition diagram for k=1 convolution codes.

Consequently we can follow a very similar analysis to show that the in-degree and out-degree for these $k = 1$ codes is restricted to 2, and in fact the task graph for all $k = 1$ convolution codes of any constraint length has in-degree and out-degree of 2 and in particular the task graph for these codes on 8 processors is exactly that of figure 6.

Thus our parallel Viterbi decoding algorithm for a (n, k, d_{min}) BCH code consists of n iterations of the weight update equations 1 and 2, each of which takes time proportional to $|R|/P$. This gives an overall time complexity of $O(n|R|/P)$. The memory complexity of our method is $O(|R|)$ because the paths and their weights must be stored for each state.

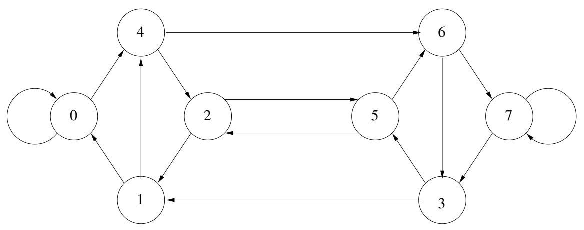

Fig. 6. Task Graph for the BCH (31,26,3) Code on 8 Processors

IV. The Field Programmable Gate Array(FPGA) Implementation

The implementation of our decoder is best described by referring to the particular task graph for the BCH (31,26,3) code shown in figure6. Each processor is assigned to a different block of the FPGA and operates independently. The arcs which represent the transmission of the weights and paths for each state to another processor are implemented by a shared register between the processors. All systems were implemented in behavioural VHDL [11]. A synthesis tool was used to construct the RTL level VHDL for the decoders. This synthesised unit was then simulated using a commercial simulation tool [1] for VHDL. In VHDL the initial conditions such as the location of the weights and paths needed to update a state are readily coded and so don't need to be calculated for each cycle of the decoding process. The received message is fanned out into all the processes a bit at a time and this is the logical clock for the machine. On receiving each input bit, each processor reads the shared registers, updates the weights and paths and writes the results to the shared registers.

V. Results and Conclusion

No. of Processors	1	2	4	8	32	$\lvert R\rvert = 2^{n-k}$
BCH(7,4,3)	63	35	21	14		8
BCH(31,26,3)	1023	527	279	155	62	32

TABLE II

Number of Cycles taken to decode BCH(7,4,3) and BCH(31,26,3) codewords.

As can be seen from Table II, our parallelisation scheme exhibits perfect scaling as the number P of processors is increased because the number of execution cycles $C = (\frac{|R|}{P} + 1)n$ as predicted. The factor $\frac{|R|}{P}n$ is the processing time and the additional factor n is the communications time, which persists when $P = 1$ because we continue to use the "shared" registers to store the weights and paths. Our parallel algorithm has the same structure an consequently the same time and memory complexity for soft as well as hard decision Viterbi decoders applied to cyclic as well as convolution decoders. Because of its perfect scaling nature we can apply our algorithm to very large codes by implementing the communication between processors by blocks of shared memory between processors. Our parallelisation technique is somewhat more efficient than trellis optimisation methods[9] since we can gain perfect speed up simply by adding more processors. This is readily achieved as all the component comparators are the same and can be replicated. This is not the case with trellis optimisation methods as these result, both in limited reduction in complexity of the decoder (a factor of 8 only for the Golay codes) and in trellis components that are not self similar and hence not able to be replicated.

References

[1] Modelsim se/ee plus 5.4c, model technology incorporated, portland, oregon, usa.

[2] E.R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill Inc, 1968.

[3] R.E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley., 1983.

[4] R.C. Bose and D.K. Ray-Chaudhuri. On a class of error-correcting binary group codes. *Information and Control*, 3:68–79, 1960.

[5] R.W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[6] A. Hocquenghem. Codes correcteurs d'erreurs. *Chiffres (paris)*, 2:147–156, September 1959.

[7] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing, Design and Analysis of Algorithms*. Benjamin-Cummings., 1994.

[8] J.S. Reeve. A parallel Viterbi decoding algorithm. *Concurrency and Computation*, 13:95–102, 2001.

[9] A. Vardy. *Handbook of Coding Theory*, chapter 24, pages 1989–2118. Elsevier Science BV, 1998.

[10] A.J. Viterbi. Error bounds for convolution codes and an asymptotically optimum decoding algorithm. *IEEE. Transactions on Information Theory*, 13:260–269, 1967.

[11] A. Williams. *A Behavioural VHDL Synthesis System using Data Path Optimisation*. PhD thesis, The Department of Electronics and Computer Science, The University of Southampton, 1998.