International Journal of Computing and Digital Systems

http://dx.doi.org/10.12785/ijcds/020206

A Low Power Parallel Sequential Decoder for Convolutional Codes

Adil EL Bourichi

National School of Applied Sciences, Ibn Tofail University, Kenitra, Morocco

e-mail: adilelbourichi@gmail.com

Received 8 Apr. 2012, Revised 31 Oct. 2012, Accepted 15 Dec. 2012

Abstract: A novel decoding algorithm having a simple hardware realization is proposed for convolutional codes. The proposed decoder accepts a simple implementation in hardware in terms of area occupancy and power consumption compared to other decoders for convolutional codes such as those based on the Viterbi algorithm (VA). Furthermore, the processing delays due to looking back and forward in a trellis as in sequential decoding algorithms are avoided, which makes the proposed decoder suitable for fast high data rates wireless communication systems. Simulation results show a comparable bit error rate (BER) performance to optimal decoders with a reduction of power consumption of 60% compared to Viterbi decoders.

Keywords: Convolutional codes, low power decoders, sequential decoding, wireless communication.

I. Introduction

Error control coding (ECC) is a classic approach to increase link reliability and lower the required transmitted power. However, lowered power at the transmitter comes at the cost of increasing power consumption of the receiver because strong and efficient codes require complex decoders with high power consumption. Convolutional codes are one of the most important ECC techniques.

Decoding of convolutional codes (CC) is generally classified in two categories: Maximum Likelihood (ML) decoding of which the Viterbi Algorithm (VA) [1] is a well known example and sub-optimal algorithms such as sequential algorithms. The Viterbi decoder has been proved to be a maximum-likelihood decoder [2]. However, it has been reported that more than 30% of power consumption in a wireless system is due to the Viterbi decoder [3][4][5][9]; moreover Viterbi decoding is impractical for constraint lengths >=7 required for high data rate applications [6][7][8][10].

Sequential decoding algorithms, on the other hand, try heuristically to find the most probable path in a tree or trellis structure. While these algorithms have lower complexity than VA, they are sub-optimal and suffer from variable decoding time which makes them non suitable for fast high data rate wireless applications.

In this paper, we propose a parallel sequential decoding algorithm where all probable paths in a tree are built in parallel and the best path is decided for at the end of the algorithm. Low power consumption of this algorithm, its fixed processing time and its suitability for hardware implementation make it a strong candidate for fast and high data rate decoding in next generation wireless networks.

II. Related Work

A. Convolutional Codes

A convolutional code is an error correcting code in which each k bit information symbol to be encoded is transformed into an n bit symbol, where $\frac{k}{n}$ is the code rate ($n \geq k$), and the transformation is a function of the last l information symbols, where l is the constraint length or memory of the code. Such code is denoted by the three-tuple (n,k,l).

A convolutional encoder is described as a mechanism of shift registers and modulo-2 adders, where the output bits are modular-2 additions of selective shift register contents and present input bits. Figure 1 shows the encoder of binary (2,1,2) code as a one shift register consisting of two delay elements and two outputs:

$$v_1(t) = u(t) + u(t-1) + u(t-2)$$

$$v_2(t) = u(t) + u(t-2)$$

During the encoding process, the contents of shift registers in the encoder are initially set to zero. The k input bits are then fed into the encoder in parallel forming the input message u consisting of k bits, to generate n output bits according to the shift-register framework, these n outputs are interleaved to obtain the final codeword.

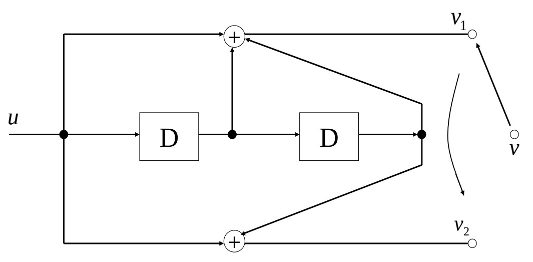


Figure 1. Encoder for the (2,1,2) convolutional code with generators g1=(111) and g2=(101)

The state diagram of this system is depicted in Figure 2. The states are defined as $u(t-1), u(t-2)$ pairs and the state transitions' outputs are defined as $v_1(t), v_2(t)$. A transition shown as dotted arrow in the figure corresponds to a bit input of 1 and a bold transition corresponds to a bit input of 0. The output bits of the encoder are shown for each transition. The state diagram can also be represented as a state transition table, shown in figure 3.

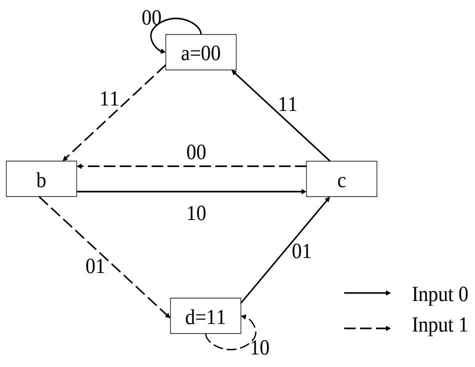


Figure 2. FSM for the encoder of figure 1

State Transition Table

Codeword	State Transition	Message bit
00	a ➔ a	0
	c ➔ b	1
10	b ➔ c	0
	d ➔ d	1
01	d ➔ c	0
	b ➔ d	1
11	c ➔ a	0
	a ➔ b	1

Figure 3. State transition table for the encoder of figure 1

The state diagram offers a complete description of the system. However, it shows only the instantaneous transitions. It does not illustrate how the states change in time. To include time in state transitions, a trellis diagram is used (Figure 4). Each node in the trellis diagram denotes a state at a point in time. The branches connecting the nodes denote the state transitions.

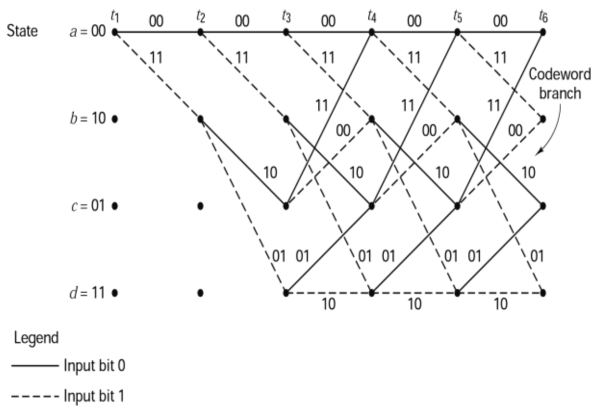


Figure 4. Trellis for the code of figure 1

In theory, code sequences of convolutional codes are of half infinite length. But for practical applications, usually finite sequences are used. There are three different methods to obtain finite code sequences:

Truncation: We stop encoding after a certain number of bits without any additional efforts. This leads to high error probabilities for the last bits in a sequence.

Termination: We add some tail bits to the code sequence in order to ensure a predefined end state, which leads to low error probabilities for the last bits in a sequence.

Tail biting: We choose a starting state which ensures that starting and end state are the same. This leads to equal error protection.

In general we prefer termination or tail biting, where tail biting increases the decoding complexity and for termination additional redundancy is required. In this paper, we consider only terminated code sequences, where we start encoding in the all-zero encoder state and we ensure that after the encoding process all memory elements contains zeros again; this can be done by adding kl zero bits to the information sequence of length N .

B. The Viterbi decoding algorithm

The Viterbi decoder [1] examines an entire received sequence of a given length. The decoder computes a metric for each path and makes a decision based on this metric. All paths are followed until two paths converge on one node. Then the path with the higher metric is kept and the one with lower metric is discarded. The paths selected are called the survivors.

The most common metric used is the Hamming distance metric. This is just the dot product between the received codeword and the allowable codeword. Other metrics are also used. These metrics are cumulative so that the path with the largest total metric is the final winner. Given a code vector **Z**, the Viterbi algorithm's objective is to find a path through the trellis starting at the all-zero state and ending at the all-zero state so that the distance measure between **Z** and a sequence **U** corresponding to the desired path is minimized.

The Viterbi algorithm relies on the observation that of two paths entering a certain state in a trellis in a given time instant, only one of them is good. Therefore, the basic idea of the algorithm is to identify which path should be erased. This is done using the follwing procedure:

If a certain state s(t+1) at time t+1 can be reached from two states s'(t) and s''(t) via branches v'(t) and v''(t) respectively, then:

A best path to s(t+1) = a best of
(best path to s'(t) extended by v'(t),
best path to s''(t) extended by v''(t)).

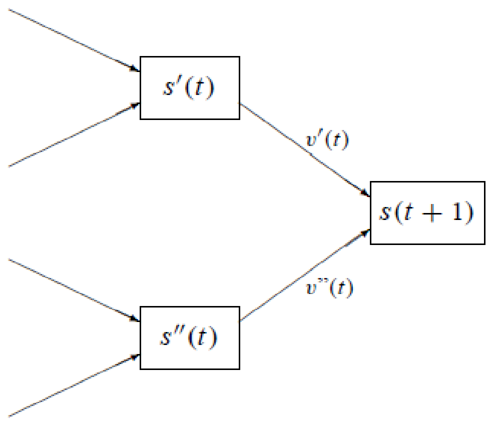


Figure 5. Path elimination in VA

C. Sequential decoding algorithms

A convolutional code with an arbitrarily long constraint length may be decoded by a recursive tree-search technique called sequential decoding. There exist various sequential decoding algorithms, of which the fastest is probably the Fano algorithm [9]. In general, sequential algorithms follow the best code path through the code trellis (which becomes a tree for long constraint lengths) as long as the path metric exceeds its expected value for the correct path. When a wrong branch is taken, the path begins to look bad and the algorithm then backtracks and tries alternative paths until it again finds a good one.

Sequential decoding achieves asymptotically the same error probability as maximum likelihood decoding but without searching all possible states. In fact, with sequential de coding the number of states searched is essentially independent of constraint length, thus making it possible to use very large (K = 41) constraint lengths. This is an important factor in providing such low error probabilities. The major drawback of sequential decoding is that the number of state metrics searched is a random variable. For sequential decoding, the expected number of poor hypotheses and backward searches is a function of the channel SNR. With a low SNR, more hypotheses must be tried than with a high SNR. Because of this variability in computational load, buffers must be provided to store the arriving sequences. The large variations in the required decoding effort of conventional sequential decoders have made them considered to be unsuitable for applications that include periodic, hard deadlines such as real-time applications.

III. PROPOSED DECODER

Supposing that messages were encoded using the termination technique, i.e. zeros are added at the end of the message to flush the encoder's contents, a convolutional decoder decides that a given codeword is valid when the parsing in a trellis or tree corresponding to that codeword starts and finishes at the all zero state. When such path is unique, the decoder is a hard decision decoder.

The proposed decoder in this paper is a hard decision decoder that aims at reducing the hardware complexity related to metric calculations and comparisons in VA while avoiding the delays in decoding time due to looking back and forward in a trellis or tree as in sequential decoders.

A. Decoding algorithm

The proposed algorithm builds a tree of states having as root the all-zero state. At the start of the algorithm, the tree consists of a single node that is the root node equal to the all-zero state. Upon reading the first codeword, the decoder looks at the transition table to find a transition that has the state at the root as starting state and that outputs the current codeword; if such transition exists, the decoder extracts the final state from it and adds it as a child to the root node. The corresponding message bit is also extracted and stored appropriately. Now, the tree consists of the root node and one leaf node that represents the next state of a transition that has the all-zero state as originating state and the first codeword of the received code vector as output; the leaf node will be used as the starting state of a possible transition that outputs the next codeword; if such transition exists, then its final state is added to the tree as a child state of the previous leaf state. If all the next codewords are correct, the algorithm outputs a tree that consists of a single path that is the correct path that would have been output by a classic sequential algorithm.

Now, let's suppose that an error had occurred and that the first codeword has been erroneous; that means there is no transition starting at the root state that outputs that codeword. The decoder then looks at all n-bit codewords and finds those that are output by a transition starting at the root state. For binary codes, there should be two of these codewords and for each one the final state is extracted from the corresponding transition and added as a child to the root state. Now, the tree consists of a root node and two leave nodes. On reading the next codeword, the decoder looks at transitions starting at each of the two new leaves that output that codeword and for each leave appropriate children are created. After all codewords have been read, the decoder output is tree consisting of multiple path, each path corresponding to a sequence of state transitions that define a possible codeword sequence. The path whose corresponding final state is the all zero state is decided to be the correct one and the correct message bit is decoded appropriately.

The next example illustrates the working of the proposed decoder for the (2,1,2) code discussed earlier. Let's suppose the message $m = 101100$ is encoded into code vector U=111000010111which is transmitted through the channel that induces an error into its 4th bit and the code vector received at the decoder is Z=1111000101111.The decoder builds a tree having as root the all zero state $a = 00$. The decoder reads the first codeword 11 and, according to the transition table in figure 3, finds that it is output to transition $a->b$ corresponding to a message bit of 1; therefore the first branch of the tree is created having as root state a and as single child state b; the decoded message bit of 1 is equally stored appropriately. It is important to note here that the decoder does not store the tree entirely but stores only the leaves (the new final states) and their corresponding decoded messages. For this purpose, a set of 2-bit registers are created (in implementation, the set of 2-bit registers can be replaced by a long register consisting of 2-bit blocks) and updated each time a 2-bit codeword is read. That is, in the example above, on reading the first 2-bit codeword 11 and determining its corresponding transition ($a->b$), the new state b is stored in the same place where state a was stored before, and a decoded bit of 1 is stored in an appropriate register to store decoded messages. At the next clock cycle, the 2-bit codeword 11 is read. Looking at the transition table, there is no transition that outputs 11 and has b as a starting state meaning that an error has occurred and correction is necessary. The decoder looks at all possible codewords that could have been originally sent: two words with Hamming distance of 1 to erroneous word 11, these are 10 and 01, and one word with Hamming distance of 2 to erroneous 11 that is 00. This latter does not correspond to any transition starting at state b but 01 and 10 do correspond respectively to transitions $b->d$ with message bit 1 and $b->c$ with message bit 0. The new final states are therefore d and c, and they correspond respectively to decoded message 11 and 10. One of these states (say, state d) will be stored in the same block where original state b used to be stored before (in implementation, the first two bits from the left of the final states' register) while the other final state (here, state c) will be stored in the subsequent block (the 3rd and 4th bit of the final states register). This same procedure is repeated for every 2-bit codeword. After all codewords have been read, i.e. after 6 clock cycles in this example, it is time for the decoder to make the hard decision on the sent message, this is done simply by taking the message corresponding to the final state equal to the all zero state. In this particular example, the correct message is the 9th message since the 9th state is the only one that is equal to the all-zero state. More specifically, the decoder outputs a number of final states, 9 states in the example: b, d, c, b, d, c, c, d and a and a number of message bits (9) corresponding to these final states: 101001, 101011… and 101100. These outputs are the two registers final_states_register and decoded_messages whose contents after the 6th clock cycle are:

$$final\_states\_register = 101101101001011100 = bdcbdccda$$
$$decoded\_messages = 101001101011 ...... 101100 = m_1 m_2 .... m_9$$

where,

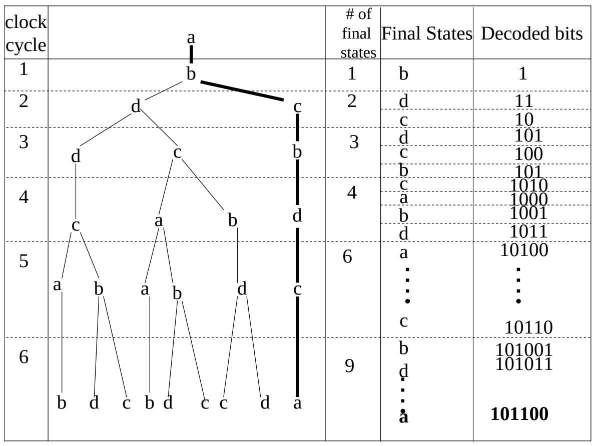
$m_1 = 101001$

$m_2 = 101011$

....

$m_9 = 101100$

The 9th final state is equal to the all zero state, and therefore the 9th message is the correct one. The correct path and correct message are shown in bold in the tree in figure 6 showing the execution trace of the algorithm for the studied example.



clock cycle	# of final states	Final States	Decoded bits
1	1	b	1
2	2	d	11
		c	10
3	3	d	101
		c	100
		b	101
4	4	c	1010
		a	1000
		b	1001
		d	1011
5	6	a	10100
		⋮	⋮
		c	10110
6	9	b	101001
		d	101011
		⋮	
		a	**101100**

Figure 6. Execution trace for the example

The main working of the algorithm is shown in figure 7.

```
Input $V$ : codevector,
      $N$ : number of n bits codewords in $V$
at each clock cyle{
$w <- read\_next\_nbit\_codeword$
for each final state $S$ in $final\_states\_register${
if($\exists Transition(S -> S', w)$) then $S <- S'$
else for each $k-bit\ w_i \neq w${
if($\exists Transition(S -> S', w)$)
  then add $S'$ to an appropriate place in $final\_states\_register$
    (with appropriate shifting to preserve old content)
}
}
if this clock cycle is the last one then{
for I = 1 to $number\_of\_final\_states$
if $final\_states\_register[I : I + l - 1] == 0...0$
  then $final\_decoded\_message <- decoded\_messages[I : I + N - 1]$
}
}
```

Figure 7. Proposed decoding algorithm

B. Circuit for the proposed decoder

The notion of maintaining a list of all possible final states at each point of time is essential to the proposed algorithm, since at the end of the processing when all codewords have been read, only one final state in this list will equal the all-zero state and therefore give the final decoded message. This list of final states can be represented by a single register consisting of blocks of l bits (l being the memory or constraint length of the code), each block of l bits representing a particular state. Likewise, the decoder should maintain a list of decoded messages, each message corresponding to one particular final state. Therefore, it is essential to use an appropriate indexing mechanism to match each final state with its corresponding decoded message. For reasons of brevity, the mechanism of updating the decoded messages and the indexing mechanism are not shown in the functional description of the circuit in figure 8 below because the understanding of their existence and functionality is intuitive. The circuit shown in figure 8 represents the main part of the proposed decoder that is the mechanism of building the tree of states as in the example of figure 5.

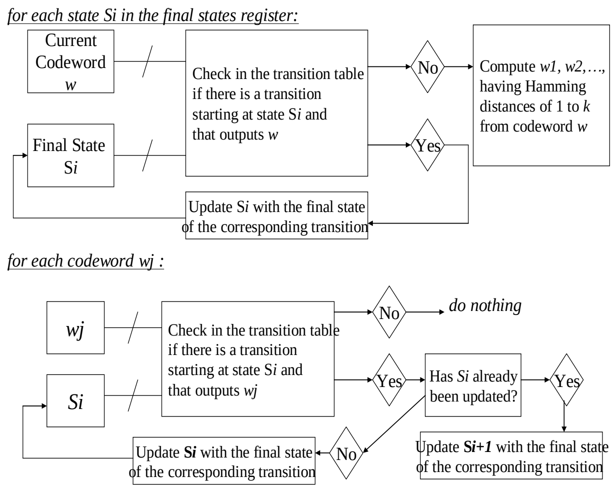

for each state Si in the final states register:
Current Codeword w
Final State Si
Check in the transition table if there is a transition starting at state Si and that outputs w
No
Yes
Compute w1, w2,…, having Hamming distances of 1 to k from codeword w
Update Si with the final state of the corresponding transition
for each codeword wj :
wj
Si
Check in the transition table if there is a transition starting at state Si and that outputs wj
No
do nothing
Yes
Has Si already been updated?
Yes
No
Update Si with the final state of the corresponding transition
Update Si+1 with the final state of the corresponding transition

Figure 8. Functional description of the decoder's circuitry

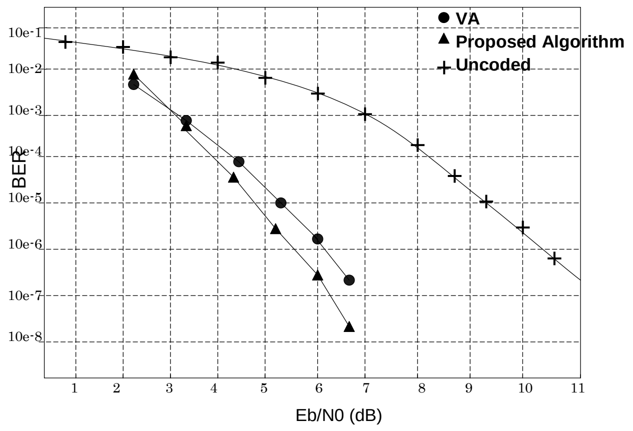


Figure 9. BER versus Eb/N0

The circuitry shown in figure 8 consists of two parts that are shown together because they happen in the same clock cycle, i.e. for one n -bit codeword. Every time a codeword is read, for all final states being presently maintained by the decoder, the latter needs to look at the transition table to find if there is a transition that outputs this codeword and has as starting state that particular final state. If there is one then no correction needs to be done and the new final state simply replaces the old one. If there is no transition starting from that particular state and outputting the present codeword, then the functionality shown in the lower part of figure 8 is performed: the codeword is assumed to be erroneous and corrections are necessary; first, all possible codewords having Hamming distances from 1 to n are stored (that is all n -bit words except the erroneous codeword) and then for each of these codewords the existence of a valid transition is checked and appropriate determination of the next final state and its storage as well as updating the decoded message are done appropriately.

IV. Simulation and Results

The proposed decoding algorithm has been simulated using C language and compared to VA in terms of bit error rate (BER) performance and average power consumption. Constraint lengths were chosen between 7 and 10 and an additive white Gaussian noise (AWGN) channel was assumed with BPSK (Binary Phase Shift Keying) modulation. Results were averaged over 200 simulation runs to obtain the BER performance shown in figure 9.

Results specific to power consumption were obtained for the (2,1,2) code presented earlier in the paper . Design Compiler from Synopsis was used to obtain a gate level circuit from a RT level description in Verilog. The technology used in our experiments is CMOS 90nm with supply voltage 1.0V. The proposed decoder dissipates 421 micro-Watts compared to 1023 micro-Watts dissipated by a register exchange (RE) implementation of the Viterbi decoder, resulting in about 60% improvement in power efficiency.

V. Conclusion

Decoding of convolutional codes using parallel sequential algorithms is a promising alternative to computationally expensive Viterbi algorithm and to sub-optimal delay incurring classic sequential algorithms especially because next generation wireless networks are expected to have strong low power and processing speed requirements. To the best of our knowledge, there hasn't been enough work on efficient implementations of parallel sequential decoders for convolutional codes and we believe that improvements of the proposed decoder in this paper are possible to allow more speed and lower power consumption.

References

[1] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," IEEE Transactions on Information Processing, 13:260-269, 1967.

[2] D. Forney, "The Viterbi algorithm", Proceedings of the IEEE, vol. 61, pp. 268-278, March 1973.

[3] I. Kang and A. N. Wilson, "Low power Viterbi decoder for CDMA mobile terminals," IEEE Journal of Solid State Circuits, vol. 33, No. 3, pp. 473-482, March 1998.

[4] S.J. Li, T. L. Brandon, D. G. Elliott, and V. C. Gaudet, "Power Characterization of a Gbit/s FPGA Convolutional LDPC Decoder," in Signal Processing Systems (SiPS), 2012 IEEE Workshop on, 2012, pp. 294-299.

[5] H. Shen-Rei and C. Sau-Gee, "A novel pipelined CCK decoder for IEEE 802.11b system," in Solid-State and Integrated-Circuit Technology, 2008. ICSICT 2008. 9th International Conference on, 2008, pp. 1621-1624.

[6] Sun, Yang, and Joseph R. Cavallaro. "A low-power 1-Gbps reconfigurable LDPC decoder design for multiple 4G wireless standards."*SOC Conference, 2008 IEEE International*. IEEE, 2008.

[7] S. Yang, J. R. Cavallaro, and L. Tai, "Scalable and low power LDPC decoder design using high level algorithmic synthesis," in SOC Conference, 2009. SOCC 2009. IEEE International, 2009, pp. 267-270

[8] H. L. Lou, "Implementing the Viterbi algorithm, fundamental and real time issues for processor designers," IEEE Signal Processing Magazine, pp. 42-52, September 1975.

[9] R. Henning and C. Chakbarati, "Low power approach for decoding convolutional codes with adaptive Viterbi algorithm approximations" , ISPLED'02, August 12-14, 2002, Monterey, California, USA.

[10] X. Wang, Y. Zhang and H. Chen, "Design of Viterbi Decoder based on FPGA", 2012 International Conference on Applied Physics and Industrial Engineering, published by Elsevier in Physics Procedia 24, pp. 1243-1247.