Fast soft-output Viterbi decoding for duo-binary turbo codes

Yannick Saouter, Claude Berrou

FAST SOFT-OUTPUT VITERBI DECODING FOR DUO-BINARY TURBO CODES

Yannick Saouter and Claude Berrou

Ecole Nationale Supérieure de Télécommunications de Bretagne,
Technopole Brest-Iroise, BP832, 29285 Brest Cédex, France.

ABSTRACT

In this article, the implementation of a Soft-Output Viterbi decoder for an eight-state duo-binary code is described. This decoder is to be used in a turbo-decoder for convolutional codes. The implementation was made with the SYNOPSYS environment and targeted a 0.18 μm technology. The layout obtained has a working frequency of 150 MHz and thus an output data rate of 300 Mbits/s.

1. INTRODUCTION

ETSI (European Telecommunications Standards Institute) is in the process of adopting new channel coding for interactivity in Digital Video Broadcasting (DVB) for satellite (RCS: Return Channel over Satellite) and terrestrial (RCT: Return Channel over Terrestrial) transmission. The error-correcting code is a duo-binary 8-state convolutional turbo code, specified for various block sizes (from 12 to 216 bytes) and coding rates (from 1/3 to 6/7). In this paper, we contemplate the possibility of extending the use of this code to continuous coded streams with high throughputs ($\geq$ 300 Mbits/s). To achieve this, we consider the Soft-Output Viterbi Algorithm (SOVA) [1, 2] as the component decoder in the turbo decoding iterative process [3]. This paper describes the design of the device, with the prospect of integration into 0.18 μm technology.

2. DUO-BINARY CODES

Duobinary codes were introduced in the domain of turbo codes by Berrou et al. [4]. These codes are in fact binary codes with an unpunctured rate of 2/3: each pair of data bits produces one redundancy bit. These codes exhibit better correcting performances and do not suffer from severe flattening at low error rates, unlike classical binary turbo codes. For this reason, they have been used in many communication protocols. In our implementation, we used the 8-state duo-binary recommended by the DVB datasheet standard. Figure 1 describes the layout of the encoder as well as its transition graph.

Yannick.Saouter@enst-bretagne.fr
Claude.Berrou@enst-bretagne.fr

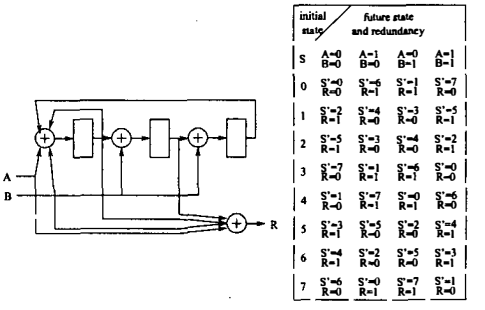

Fig. 1. Architecture and state lattice of the duo-binary convolutional encoder

3. ALGORITHM OVERVIEW

The SOVA is an extension of the classical Viterbi algorithm: given sampled data from the channel, it computes four values (one for each symbol of the alphabet of the code) at each clock period. These values are minimal when the probabilities of the corresponding symbol in the sequence are maximal. These values will be called *weights* in the following. The Viterbi algorithm is explained in the original paper [5] and in many other publications. The weighting algorithm is derived from Battail's unsimplified revision formula which is to be found in [6]. In the case of binary codes, a simplified algorithm has been proposed in [1] and in the general case, including duo-binary codes, the derivation is given in [2]. The reader should refer to these articles for full details and especially to [2] which describes the exact weighting procedure of the circuit.

The algorithm is made up of two parts. The first one, given the sampled data of the input channel, computes at any time the most likely state of the encoder (classical Viterbi algorithm). The second one, given the sampled data and the estimated state determined by the latter operation, computes the opposite of the logarithm of the probabilities that the emitted symbol is one of the four possible symbols (the most probable symbol will be called the *principal decision*). This second part makes a first estimate of the weights by computing the differences of metrics for the state decided by the Viterbi algorithm and then, the initial values are up-

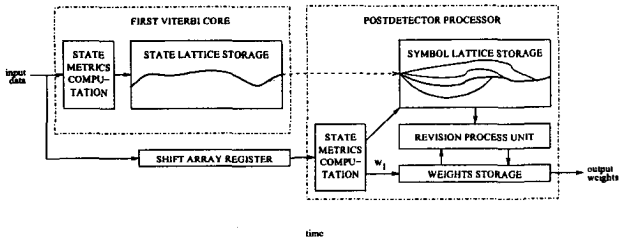

Fig. 2. Overall architecture of the duo-binary soft-output decoder.

dated according to the three paths which were in competition with the survivor path. Thus there are two distinct Viterbi devices in the circuit, each with its own memory. In the following, unless otherwise specified, the first one will be referred to as the Viterbi device, while the second one will be called the *postdetector processor* or even simply the processor. The global architecture is illustrated in Figure 2.

The Viterbi algorithm, for any of the eight states, at any step computes one value called the *state metric*. This value is the minimum of the four incoming metrics of this state (one for each possible symbol). Any incoming metric is the result of the addition of the state metric of a predecessor of the state in the state graph plus the *branch metric* associated with the transition in the graph. This branch metric is equal to $\pm X_1 \pm X_2 \pm Y$ where X_1 (X_2 and Y resp.) is the value read from the channel for the least significant bit of the emitted symbol (the most significant bit and associated redundancy resp.). The $\pm$ operator is equal to $+$ or $-$ depending on whether the expected value is 0 or 1. For instance according to Fig. 1, from state 4 it is possible to reach state 7 if the data are $A = 1$ and $B = 0$ and the associated redundancy is $R = 1$. Thus the branch metric associated with this transition will be equal to $-X_1 + X_2 - Y$ and state 7 will have $SM_4 - X_1 + X_2 - Y$ as one of its four incoming metrics, if SM_4 is the state metric of state 4. Thus for any state, at any moment, we can have the history of the choices that led to this particular step. The output of the Viterbi algorithm will be the oldest state in the history of the most likely state (i.e. the one which has the smallest state metric at a given time). If d_V is the length of stored histories for any state, then at time t, the output value of the Viterbi algorithm will be the most likely state at time $t - d_V$. The value d_V is called the *depth of the Viterbi algorithm.*

The postdetector processor also computes the state metrics at any time. Given the state decided by the Viterbi algorithm, it makes a first estimate of the probabilities. If m is the state predicted by the Viterbi algorithm, the initial weight for symbol 0 (1, 2 and 3 resp.) will be the difference of the incoming state metric for state 0 (1, 2 and 3 resp.), i.e. the sum of the state metric of the predecessor of m for a 0 (1, 2 and 3 resp.) received symbol plus the associated branch metric, minus the least of the four incoming state metrics.

This initial value is then updated according to previous values and according to the histories of symbol choice leading to state m. In the history of m, we select the four histories corresponding to any possible choice for the received symbol. One of these paths is called the *survivor path* and corresponds to the least incoming metric for m. Along this path, the weights of all the symbols are null. The other three paths are called the *concurrent paths*. The updating process is the following one: let s be the symbol associated with a concurrent path; if the weight associated with a symbol of the concurrent path is greater than p_s, the initial probability for s, then this value has to be lowered to p_s.

4. ARCHITECTURE OF THE VITERBI DECODER

The architecture of the Viterbi decoder is depicted in Figure 3. Each of the modules has a specific task:

- ADDCOMP computes the branch metrics $\pm X_1 \pm X_2 \pm Y$ in a pipeline way (any of the eight possible branch metrics is computed in a binary tree fashion in three clock periods and the computation of the state metrics is delayed by the same time in order to preserve synchronization between the modules ADDCOMP and METCOMP);
- METCOMP computes all the incoming state metrics, given the initial state metrics stored in the STATE METRICS REGISTER module and the branch metrics given by the ADDCOMP module (the METCOMP module contains the topology of the state lattice and dispatches any branch metric to all the transitions where it is required);
- MIN4S compares the incoming state metrics and keeps the smallest one for any state (it is just a minimum out of four values performed in a binary tree fashion);
- OVERFLOW deals with the overflow hazards: when all the most significant bits of the state metrics are equal to 1, they are all set to 0.
- STATE METRICS REGISTER stores the state metrics;
- STATE SELECTION selects for any state one of the four possible branches for the history;
- REGISTER EXCHANGE ARCHITECTURE stores the history for any of the states;
- MIN8PIPE computes the most likely state (that has the lowest metric) in a pipeline way;
- POSTAMBLE is a simple shift array register which has as many pipeline levels as MIN8PIPE so that decisions of the latter module are synchronized with the output of the POSTAMBLE module.

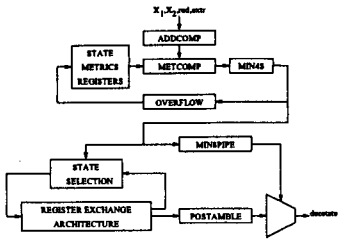

Fig. 3. Architecture of the Viterbi decoder.

The samples from the channels were assumed to be signed and 4 bits wide. Simulations on a turbo-decoder then established that 8 bits were enough for state metrics, and that a depth equal to 16 gives quite acceptable performance. With these parameters, Table 1 gives the complexity of any of the modules of the architecture, expressed in terms of 2-input NAND equivalent gates.

Module	Gates	Module	Gates
ADDCOMP	1403	STATE SEL.	1241
METCOMP	1521	MIN4S	1099
OVERFLOW	14	STORING	1995
MIN8PIPE	727		
TOTAL	8000		

Table 1. Number of gates in the Viterbi device: 1 gate $\simeq$ 1 2-input NAND gate.

5. ARCHITECTURE OF THE POSTDETECTOR PROCESSOR

The postdetector processor is subdivided into two parts. The first one consists of a second Viterbi device. It has two tasks: first, it again computes the state metrics whose values are required for the computation of the initial weights. Second, it computes again the history of symbol choices for any of the eight states. Histories are required in order to apply the updating rule and this is the second part of the processor, which is effectively in charge of computing the updated values for the weights according to the updating rule.

The weight for the principal decision (i.e. the decision which would be predicted by a conventional Viterbi decoder and which corresponds to the least incoming state metrics amongst the 32) is always equal to zero. We then use a technique to avoid storing them: the symbol s of the principal decision is stored in a shift array and the other three weights are permuted in such a way that the weight of another symbol s' is stored in the shift array in row $s \oplus s'$. At the output of the shift array, weights are rearranged in their natural order, that is, the outgoing weight at time t of symbol s' will be the one stored in row $s' \oplus s(t - d_R)$, where $s(t - d_R)$ is

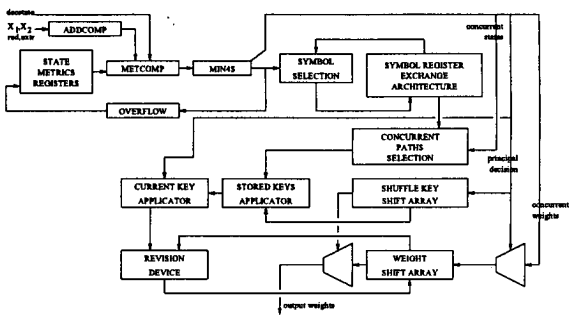

Fig. 4. Architecture of the postdetector processor.

the principal decision given by the Viterbi detector at time $t - d_R$ and d_R is the depth of the revision process. This technique has two advantages: first, it saves a quarter of the area of weight storage and second an updated value becomes at most the minimum out of four values and not five values as previously. Indeed, the revision process involves for any of the four possible symbols, lowering the weights along the corresponding history: if a symbol s has a weight equal to W_s, then any of the symbols with weight W in the history of the path corresponding to s will have its new weight set to $min(W, W_s)$. Thus, in the classical revision scheme the updating value will be $min(W, W'_{00}, W'_{01}, W'_{10}, W'_{11})$ where W'_{ij} is equal to W_{ij} if the revised symbol belongs to the history of the concurrent path corresponding to symbol (i, j) and will be equal to $+\infty$ in the other case. Amongst the four values W'_{ij}, one corresponds to the survivor path. In this case we will have $W'_{ij} = 0$ if the revised symbol belongs to the history of the survivor path and $W'_{ij} = +\infty$ in the other case. In the first case, the new value for the weight will be 0 whatever the other values involved in the minimum are, and in the second case, the weight of the survivor path has no effect on the final value. When we shuffle the weights by the value of the principal decision, there are only three weight values to store and the new weighting value will be $min(W, W''_1, W''_2, W''_3)$ if the revised symbol does not belong to the survivor path and 0 otherwise, where W''_j is the weight associated with the j-th concurrent path different from the survivor path. The addressing of W'' has to take into account the shuffle of the current principal decision as well as the shuffle of the former principal decision which shuffles W. Thus this technique allows a quarter of the comparators used to be saved and, in addition, this minimum requires only two stages of comparators and not three, if we neglect the small additional delay of the multiplexer corresponding to the survivor path. Thus, the critical path of the system is shortened.

Figure 4 depicts the architecture of the postdetector processor. The modules have the following tasks:

ADDCOMP, METCOMP, OVERFLOW and STATE METRICS REGISTERS have the same behaviour as previously and perform the kernel computation of the second Viterbi algorithm;

MIN4S as previously computes the future metric for each state, but given the decided state *decstate*, it also provides the decided symbol, the concurrent states and the concurrent weights (i.e. corresponding to the three other decisions);

SYMBOL SELECTION selects for any state one of the four possible branches for the symbol history;

SYMBOL REGISTER EXCHANGE ARCHITECTURE stores the history of the symbol choices for any state;

CONCURRENT PATH SELECTION makes the selection of the three alternate paths in the history of symbols leading to the decided state *decstate*;

SHUFFLE KEY SHIFT ARRAY stores the keys used to avoid the storage of null weights corresponding to the principal decision in the WEIGHT SHIFT ARRAY (in fact the keys are simply the principal decision symbol);

STORED KEYS APPLICATOR applies the shuffle keys stored in the SHUFFLE KEY SHIFT ARRAY to the paths selected by CONCURRENT PATH SELECTION;

CURRENT KEY APPLICATOR applies the current key to the latter shuffled paths so that paths are coherent with the shuffled concurrent weights;

REVISION DEVICE performs the updating process computations;

WEIGHT SHIFT ARRAYS store weights shuffled by the symbol corresponding to successive principal decisions.

In the implementation, the same representations were used for the postdetector processor as for the Viterbi device but the weights in the WEIGHT SHIFT ARRAYS were stored as unsigned 6-bit wide numbers. The depth of the updating process was set to 10, after simulations. Table 2 gives the complexity of any module of the processor, expressed in equivalent 2-input NAND gates.

Module	Gates	Module	Gates
ADDCOMP	1403	STORED KEYS APPL.	99
METCOMP	1521	CURRENT KEYS APPL.	70
OVERFLOW	14	REVISION	11268
MIN4S	1099	STORAGE	2896
SYMB. SEL.	773		
TOTAL	19143		

Table 2. Number of gates in the postdetector processor: 1 gate $\simeq$ 1 2-input NAND gate.

6. PERFORMANCE AND PERSPECTIVES

The description of the circuit was written in VHDL description language and was synthesized by the SYNOPSYS development environment for ASIC 0.18 μm technology library provided by Austria Microsystems. The description of the circuit was validated by simulation before and after synthesis. The chip has a total area of 0.4 mm^2 (0.1 mm^2 for the Viterbi device and 0.3 mm^2 for the postdetector processor) and is expected to work up to 150 MHz (250 MHz for the Viterbi device and 150 MHz for the postdetector processor) in the typical case of a temperature which gives an output data rate of 300 Mbits per second. In our future work, we plan to improve and optimize the architecture in order to reach a working frequency of 300 MHz (i.e. an output data rate of 600 Mbits/s).

7. REFERENCES

[1] C. Berrou, P. Adde, E. Angui, and S. Faudeil, "A low-complexity Soft-Output Viterbi Decoder Architecture," in *1993 Intl. Conf. on Communications*, Geneva, Switzerland, 1993, vol. 2, pp. 737–740.

[2] D. Kwan and S. Kallel, "A Rate-k/n Heuristic Soft-Output Viterbi Algorithm (SOVA) that is Postdetector-Compatible," *IEEE Trans. on Comm.*, vol. COM-46, no. 5, pp. 621–626, May 1998.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error correcting coding and decoding: Turbo-codes," in *Intl. Conf. on Communications*, Geneva, Switzerland, May 1993, vol. 2, pp. 1064–1070.

[4] C. Berrou and M. Jezequel, "Non-binary convolutional codes for turbo coding," *Electronics Letters*, vol. 35, no. 1, pp. 39, January 1999.

[5] A.J. Viterbi, "Error Bounds for Convolutionnal Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. on Inf. Theory*, vol. IT-13, pp. 260–269, April 1967.

[6] G. Battail, "Pondération des symboles décodés par l'algorithme de Viterbi," *Annales des Télécommunications*, vol. 42, no. 1-2, pp. 31–38, January 1987.