VLSI implementation of SISO arithmetic decoders for joint source channel coding

Simone Zezza and Guido Masera
Politecnico di Torino, Italy
{simone.zezza;guido.masera}@polito.it

Abstract

In this paper we propose an efficient VLSI implementation of a Soft Input Soft Output (SISO) arithmetic code (AC) decoder for joint source channel coding. The addressed application shows a very high level of processing complexity, but, to the best of our knowledge, no papers have been published in the literature on the hardware implementation of the considered joint source channel scheme. First we introduce a simplified algorithm for the SISO AC, which is 1.3 times faster than the standard one. Then an efficient SISO AC architecture is proposed and synthesis results on a 0.13 \( \mu \)m standard cell technology are reported for two different sets of parameters (\( M=128, M=256 \)). The proposed core runs at 338.9 MHz and can decode up to 124.987 kbit/s.

1 Introduction

With the increasing popularity of mobile communications, research in the area of joint source channel coding has received renewed impetus.

In [7] an innovative joint source channel iterative decoding scheme for arithmetic codes (AC) has been proposed. The system executes the iterative soft decoding of arithmetic codes by means of a novel Soft-in Soft-Out (SISO) decoder based on suboptimal search and pruning of a binary tree. In order to improve the performance of the iterative decoder a forbidden symbol [5] has been introduced in the binary and not adaptive arithmetic encoder/decoder.

Several works have been published in the literature on the comparison between separated and joint source channel coding, evaluated in terms of Bit Error Rate (BER), Frame Error Rate (FER), etc., but today, to the best of our knowledge, no papers are available on architectural complexity and hardware implementation of joint source channel decoders either using variable length [1] or arithmetic codes. However the search for efficient implementation architectures is of great interest because of the amazing level of complexity in the addressed application.

The paper is organized as follows. In Section 2 the SISO AC proposed in [7] is briefly reviewed while in Section 3 an efficient implementation of it is shown. Section 4 describes the proposed VLSI architecture and implementation results are given in Section 5 in terms of area and throughput; finally, in Section 6 conclusions are drawn.

2 System overview

At the encoder side, the AC receives as input a sequence of binary symbols \( s = \{s_0, s_1, ..., s_{L-1}\} \) of fixed length \( L \), and it outputs a sequence of symbols \( u = \{u_0, u_1, ..., u_{N(s)-1}\} \) of variable length \( N(s) \). The convolutional coder (CC) receives as input a permuted sequence \( u' \) of symbols \( u \), and it generates the codeword \( c \). This latter is transmitted across an Additive White Gaussian Noise (AWGN) channel by means of a BPSK modulation.

At the decoder side, the SISO CC block implementing the standard BCJR algorithm [6], observes a noisy version \( y = \{y_0, y_1, ..., y_{N(s)-1}\} \) of the encoded sequence \( u \), and it computes the probabilities \( P\{y|u'\} \). The inputs of the SISO CC decoder are the a priori probabilities \( P\{u'\} \) and the a posteriori probabilities (APP) \( P\{y|c\} \). The SISO AC receives \( P\{y|u\} \), and it computes the APP \( P\{u, y\} \), which is used to update the a priori probability \( P\{u'\} \) for the next iteration.

To accomplish the SISO AC, in [7], the standard BCJR algorithm has been modified. Particularly the usual trellis representation was replaced with the binary tree showed in the example of Fig. 1, where all the possible sequences \( u \) of length \( N(s) \) are indicated on a short example sequence. Each node \( \sigma_i \) in the tree represents the state of the arithmetic decoder, whereas the index \( i = 0, 1, ..., N(s) - 1 \) indicates the decoding step. Each transition between two states \( \sigma_i \) and \( \sigma_{i+1} \) is associated to a bit label \( u_i \); depending on \( u_i \), the arithmetic decoder updates the state from \( \sigma_i \) to \( \sigma_{i+1} \), and it decodes the variable number \( L_i \geq 0 \) of decoded source symbols \( s_i \).
The goal of the SISO AC block is to estimate the a posteriori probability (APP) \( P\{u_t = k|y\} = P\{u_t = k, y\}/P\{y\}, k=0,1 \); since given the received sequence \( y \), \( P\{y\} \) is just a scaling factor, in the following we report the equation needed to compute the joint probabilities \( \lambda_t(k) = P\{u_t = k, y\} \). It is well known that, from the implementation point of view, the use of the log-likelihood ratios (LLR) instead of the symbol probability \( \lambda(k) \) is generally preferred [8]. The LLR are defined as follows:

\[
\Lambda_t = \ln\left\{ \sum_{(\sigma_t, \sigma_{t+1}) : u_t = 1} e^{\alpha_t(\sigma_t) + \gamma_t(\sigma_t, \sigma_{t+1}) + \beta_{t+1}(\sigma_{t+1})} \right\} \quad (1)
\]

\[
-\ln\left\{ \sum_{(\sigma_t, \sigma_{t+1}) : u_t = 0} e^{\alpha_t(\sigma_t) + \gamma_t(\sigma_t, \sigma_{t+1}) + \beta_{t+1}(\sigma_{t+1})} \right\}
\]

where \( \alpha_t(\sigma_t) \) and \( \beta_t(\sigma_t) \) can be computed with a forward and backward recursion respectively:

\[
\begin{aligned}
\alpha_{i+1}(\sigma_{i+1}) &= \alpha_i(\sigma_i) + \gamma_i(\sigma_i, \sigma_{i+1}) \\
\alpha_0(0) &= 0 \\
\beta_i(\sigma_i) &= \ln\left\{ \sum_{\sigma_{i+1}} exp[\gamma_i(\sigma_i, \sigma_{i+1}) + \beta_{i+1}(\sigma_{i+1})] \right\} \\
\beta_{N(s)}(\sigma_{N(s)}) &= \sum_{i=0}^{N(s)-1} \ln[P(s_i)]
\end{aligned} \quad (2)
\]

\[
\begin{aligned}
\beta_{i}(\sigma_{i}) = \ln\left[ \sum_{\sigma_{i+1}} exp[\gamma_{i}(\sigma_{i}, \sigma_{i+1}) + \beta_{i+1}(\sigma_{i+1})] \right] \\
\beta_{N(s)}(\sigma_{N(s)}) = \sum_{i=0}^{N(s)-1} \ln[P(s_i)]
\end{aligned} \quad (3)
\]

where \( i = 0, ..., N(s) - 1 \), \( \gamma_i = u_i \cdot \delta_i \) (with \( \delta_i = \ln(P[y_i|u_i=1]) \) and \( \sum_{i=0}^{N(s)-1} \ln[P(s_i)] \)) represents the a priori probability of the sequence that corresponds to the decoding path terminating in the state \( \sigma_{N(s)} \).

In conclusion, the SISO AC decoder goal is the evaluation of (1) through a forward and backward recursion defined by (2) and (3) respectively.

Looking at the forward recursion in (2), it is clear that at each decoding step \( i \), both the number of nodes and paths grow in exponential manner. In order to contrast this phenomenon, Authors of [7] adopt the M-Algorithm (MA) [2].

### 2.1 Previous M-algorithm architecture

In the past, the MA has been employed as a suboptimal alternative to the Viterbi algorithm (VA) [4]. Contrary to the VA, in the MA only a subset of \( M \) paths are retained at every instant. This implies a direct exploration of the trellis where each iteration is composed of three steps: extension, suppression of the merged paths, and selection. During the extension phase, the \( M \) paths at a given time \( t \) are extended to their two successive at time \( t + 1 \). For the \( 2M \) paths thus generated, the new paths metrics are computed. In the selection phase, the \( M \) surviving paths, i.e the \( M \) paths having the smallest paths metrics, are selected. The suppression and selection phases imply the use of sorting circuits. However, the hardware complexity of sorting circuits can be very large. Few hardware implementations of the MA have been reported.

In [9] Mohan and Sood describe a multiprocessor architecture. In the selection phase, it employs a \( 2M - item \) sorter based on a modified Batcher’s sorting network [3] and the complexity of the selection phase can be evaluated using the equation \( \frac{M}{2} \cdot [(\log 2M)^2 - \log 2(2M) + 4] - 1 \). To select the best \( M \) paths the selection procedure requires \( \frac{1}{2} \cdot \log_2(2M) \cdot [(\log 2M) + 1] \) steps.

The Authors of [11] propose a simplified version of the algorithm described in [9] based on a different sorting strategies. In this case the selection phase needs two \( M - item \) odd-even sorting networks and one layer bitonic merge [3]. The complexity of the selection phase can be evaluated using the equation \( \frac{M}{2} \cdot [(\log 2M)^2 - \log 2(2M) + 4] - 2 + M \) and to select the best \( M \) paths it requires \( \frac{1}{2} \cdot \log_2(2M) \cdot [(\log 2M) + 1] + 1 \) steps.

In [10] Simmons proposes a nonsorting VLSI architecture for implementing the MA. In fact, the sorting circuit adopted during the selection phase has been replaced with a simple algorithm, which considers \( 2M \) paths metric represented on \( k \) bits, analyzes each bit starting from the MSB to the LSB, and selects the best \( M \) paths with the highest value of metric. The hardware complexity of this architecture is considerably reduced given the lack of sorting circuits. To select the best \( M \) paths the selection procedure requires \( 2 \cdot k \cdot \log_2(2 \cdot M) \) steps.

### 3 Efficient implementation of a SISO AC

The main differences between the MA adopted in the SISO AC and the MA described above are: the searching MA explores a tree and not a trellis; each path in the tree is associated to an \( \alpha \) value and a \( \beta \) value; the suppression phase is not necessary and consequently a single sorting operation is performed with respect the path metric.

Since in [7] the path metric is computed using (2), in the following we will use the \( \alpha \) values to select the best \( M \)
paths of the tree, those exhibiting the highest values of metric. During the extension phase, in order to select the best \( M \) nodes without ordering of metrics, two simple buffers One-Vector and Zero-Vector of length \( M \) have been used. In the former we load the nodes obtained decoding an encoded bit \( u_i = 1 \) by means of an arithmetic decoder (one_extension), and in the latter those obtained decoding an encoded bit \( u_i = 0 \) (zero_extension). At each sorting step the heads of the One-Vector and Zero-Vector buffers are compared and the one with the highest metric is chosen as one of the \( M \) values. Depending on which buffer provides one of the \( M \) values the content of that buffer is right shifted by one position. The comparison and shifting continues until all the \( M \) values are selected.

As an example, consider the nodes at stage \( i + 1 \) in Fig. 1. If \( M = 2 \) the operation of the algorithm is demonstrated in Fig. 2. First the values of the metric computed using (2) \( \alpha_{i+2} = -40 \) and \( \alpha_{i+2}^{+1} = 55 \) are loaded into the buffers One-Vector and Zero-Vector, respectively. The next two metrics \( \alpha_{i+2}^j = -110 \) and \( \alpha_{i+2}^{+3} = -15 \) are similarly loaded into the next two free locations in One-Vector and Zero-Vector, respectively. The process continues until 2\( M \) values are stored in the two buffers. Next \( \alpha_{i+2}^j = -40 \) and \( \alpha_{i+2}^{+1} = 55 \) are compared and the node with \( \alpha_{i+2}^{+1} = 55 \) is selected as one of the \( M \) nodes. Next Zero-Vector shifts right by one position, \( \alpha_{i+2}^j = -40 \) is compared with \( \alpha_{i+2}^{+3} = -15 \) and the node with \( \alpha_{i+2}^{+2} = -15 \) is selected as the second node.

One important feature of computing the node metrics using (2) is partial ordering of the metric values in One-Vector and Zero-Vector buffers. From Fig. 2 it is possible to see that the value of the metrics in One-Vector and Zero-Vector are always ordered in the ascending order from the left to right. This partial ordering permits the selection of the \( M \) nodes with the highest metric values using a simple comparison and shift operations. In order to select the best \( M \) paths, the selection procedure requires \( M \) steps. We will refer to this algorithm as simplified M-algorithm (SMA).

The SMA was inserted in the C++ model implementing the whole encoder-decoder system and using the same setup parameters used in [7]: the total execution time required to decode 3 \( M \) bit, is equal to 1073 s and 801 s when SISO AC with MA or SMA is used respectively. The speed up achievable with the SISO AC with SMA is around 25.3%.

4 SISO AC decoder VLSI implementation

The block diagram of the proposed implementation architecture is outlined in Fig. 3. The architecture primarily consists of Forward unit and Backward unit. It also includes: two arithmetic decoders (AC Decoders0 and AC Decoders1) inside the Forward unit, one Memory Tree to store all the information required to go through the tree forward and backward, one Memory InPs to store, at the end of each extension, all the information belonging to the arithmetic decoders and a Memory \( \delta \) unit to store all the \( \delta \) associated to each decoding step \( i \). A Control unit handles the interactions between all these blocks.

In order to reduce the hardware complexity of the arithmetic decoder, the effect of finite precision representation of data on global performance has been investigated. The bit error probability \( P_w(\epsilon) \) versus signal to noise ratio \( E_b/N_0 \), is plotted in Figure 4 for 16, 20 and 32 bit fixed-point arithmetic decoder. While a performance loss of more than 1 dB at \( P_w(p) = 5 \cdot 10^{-4} \) is experimented with 16 bit fixed-point representation, 20 bit data representation results in almost negligible performance loss (about 0.1 dB at \( P_w(p) = 1 \cdot 10^{-4} \)) and it is therefore selected for implementation instead of the 32 bit format adopted in [7]. Moreover we have estimated that the total execution time required to decode 3 \( M \) bits is equal to 635 s when the 20 bit fixed-point arithmetic decoder is adopted.

5 Performance evaluation

A straightforward comparison between the SISO AC and the architectures described in [9], [11] and [10] is impractical, due to the difference between the algorithms implemented. However, in Table 1 we have reported a compar-
The comparison between the SMA and the MA proposed in [9], [11] and [10] in terms of number of comparators and steps required to select M paths with the highest value of metric. Observing the results reported in Table 1, it is possible to see that the SMA requires a lower number of hardware resources than [9] and [11] methods, both when M = 128 and M = 256. However, the steps required to sort M numbers represented on 32 bit, using our method, is either greater than the numbers required by the [9] and [11] or lower than the number required by the [10] method. The proposed architecture has been implemented in VHDL and synthesized using 0.13 μm CMOS standard cells technology. When M = 128 the number of equivalent gates required to realize the SISO AC is 1,594,088, whereas the amount of memory is 167kByte. If we chose M = 256 the number of equivalent gates is equal to 3,295,597 and the total amount of memory is 340 kByte. The two considered cases, M = 256 and M = 128, are those yielding the best performance in [7]: a reduced complexity decoder with M = 128 exhibits a maximum penalty limited to 0.12 dB at P_w(e) = 10^{-4}. We have conducted the performance analysis of the proposed architecture using the same setup parameters used in [7] and clock frequency equal to 338.9 MHz. With M = 128 the throughput of the SISO AC is of 114.2 kbit/s, whereas with M = 256 the throughput is of 55.82 kbit/s. If we reduce the number of iterations to three, the throughput achievable with M = 128 is 124.987 kbit/s and with M = 256 is 57.6 kbit/s.

6 Conclusions

In this paper an efficient VLSI implementation of a SISO AC arithmetic decoder has been proposed. The architecture runs at 389.9 MHz and can decode up to 124.987 kbit/s when M=128. In this particular case we have a maximum penalty of about 0.12 dB at P_w(e) = 10^{-4}, but the hardware resources required are about 50% less, if compared with the case M = 256 whose correspond the best performance. As far as the future developments are concerned, we are currently working on a complete ASIC implementation of a parallel extensions SISO AC.

References