Efficient Hardware Implementation of Advanced Soft Information-Set Decoders in FPGAs

R. P. JASINSKI[1], W. GODOY JR.[1], A. GORTAN[1], S. B. L. FRANÇA[1], V. A. PEDRONI[1]
Dept. of Electronics
UTFPR
Curitiba, Paraná
Brazil
pedroni@utfpr.edu.brwww.lme.citec.ct.utfpr.edu.br

Abstract: - This paper has four main goals: (i) to describe in detail a new architecture to implement soft-decision, information-set-based decoders purely in hardware; (ii) to investigate the effects of quantization of the received word on the decoder performance, calculating the minimum number of bits that should be adopted; (iii) to present a strategy for optimizing the choice of candidate codewords, allowing the selection of a small set with a very high probability of containing the best candidate; and (iv) to present a new acceptance criterion that is both highly efficient and well-suited for hardware implementations.The proposed architecture can be used to implement any linear block decoder and is shown to be highly area-efficient, with the $C(48,24,12)$ code occupying only 20% of the smallest FPGA in the Stratix IV family. It is also shown that there is very little to gain by using more than 3 quantization bits, and that sets as small as 3% of all possible values suffice to obtain essentially the same results as true MLD. The presented acceptance criterion reduces in 96.8% the number of candidates that must be evaluated for the $C(24,12,8)$ code, with performance difference relative to the Taipale-Pursley criterion never larger than 12%.

Key-Words: -Block codes, error correcting codes, information set, decoder, hardware, FPGA.

1 Introduction

No matter how well a decoding algorithm for error correcting codes performs, it is of little use if its complexity prevents it from being implemented. A notable example is LDPC codes [1], proposed by Gallager in the 1960s, whose implementation only became viable in recent years, making it now one of the most intensely studied classes of error-correcting codes [2].

Most decoding algorithms for error-correcting block codes use only hard decisions to demodulate received symbols [3]. Although this approach simplifies the decoding procedure, the use of analog information may improve decoding performance up to 3 dB in the case of Gaussian channels, and up to 10 dB on Rayleigh fading channels [4]. On the downside, it increases the demand for computational resources. Examples of algorithms that exploit this extra information are Generalized Minimum Distance (GMD) [5] and its variants [3].

Especially in iterative algorithms, there is usually a tradeoff between throughput, error correction performance, and computational resources. In order to improve error correction, one can increase the number of iterations or use additional information (e.g., analog levels of the received symbols). In order to improve throughput, one can accept a lower error correcting performance or use more computational power. When both error correction and throughput must be improved, one possible solution is to implement the algorithms directly in hardware. Arithmetic operations that require a large number of CPU cycles when implemented in software can be performed in a single cycle when dedicated hardware is available.

If computational power is not a limiting resource, maximum-likelihood decoding (MLD) is an optimal decoding procedure. However, since it involves iterating through all possible codewords, its computational cost is prohibitive. In order to make this approach practical, the number of candidates evaluated need to be reduced, at the expense of an inferior decoding performance.

One approach to reduce the number of computations is the use of an*acceptance criterion*.

The basic idea is to check the decoding efficiency for each processed candidate, interrupting the search when a predefined performance level has been achieved.

This paper describes a design strategy that makes viable hardware implementations of information-set-based decoders with near-MLD performance. This is achieved through four main developments: (1) a criterion that reduces the number of candidate codewords, without significant performance loss; (2) a modified, hardware-friendlier version of the Dorsch decoding algorithm; (3) detailed circuit analysis and optimization in all critical parts (matrix manipulators, data sorter, data memory, and data path) that comprise the final circuit; and (4) a hardware-oriented acceptance criterion, optimized for physical implementation and requiring only a simple test in the final stage to check whether a given candidate is the maximum-likelihood output of the decoder. The final hardware implementation is described in detail, enabling the adoption of these techniques in similar high-performance decoders.

The paper organization is as follows. Section 2 introduces the fundamentals of information-setdecoding. Section 3 introduces a basic hardware architecture that implementsthe original Dorsch decoding algorithm. The next three sections discuss alternative ways to improve the decoder performance while still maintaining it suitable for a hardware implementation. Section 4 analyses the effects of quantizing the received word in the decoder performance. Section 5 presents a method to choose a small number of candidate codewords that yield a performance close to MLD. Section 6 presents a new acceptance criterion that is suited for a hardware implementation. Section 7 presents the final hardware architecture, encompassing all three improvements. Section 8 presents the results obtained by synthesizing both implementations, and section 9 summarizes the conclusions.

2 Information-Set Decoding

Consider a $C(n, k, d_{Hmin})$ linear block code with codewords $\mathbf{c}_i$ ($i = 0$ to 2^k-1), minimum Hamming distance d_{Hmin}, and generator matrix $\mathbf{G}$ (of size $k \times n$). The encoding procedure consists in multiplying a message vector $\mathbf{u}$ (with k bits) by $\mathbf{G}$ to produce a corresponding codeword $\mathbf{c} \in C$ (with n bits). However, the decoder receives $\mathbf{x}$, a possibly corrupted version of $\mathbf{c}$, from which it extracts a hard-decoded sequence $\mathbf{r}$, along with a reliability measure (based on the actual analog value) of each symbol. The latter is needed in order to rank the symbols in $\mathbf{r}$ according to their reliabilities, originating the sorting sequence $\mathbf{s}$, which allows the use of soft decision.

$\mathbf{G}$ consists of k linearly independent (LI) columns (usually, the identity matrix $\mathbf{I}_k$) plus $n - k$ columns (linearly dependent on the previous ones) responsible for adding the redundancy. In block form, $\mathbf{G}$ can be represented as $\mathbf{G} = [\mathbf{I} \mid \mathbf{P}]$, where $\mathbf{I}$ is a $k \times k$ identity matrix and $\mathbf{P}$ is a $k \times (n - k)$ parity matrix.

An *information set* (IS) is defined as any set of k LI columns in $\mathbf{G}$ [6]. Because of the redundancy added by the encoder, it is not necessary to consider all n bits to decode a received sequence. The core principle of soft information-set decoding is to consider only the k most reliable LI symbols in $\mathbf{x}$ to reconstruct the original source message. Such algorithm can be roughly summarized as follows.

a) Extract from the received codeword $\mathbf{x}$ the hard-decoded sequence $\mathbf{r}$ and the corresponding reliability sequence $\mathbf{s}$.

b) Based on $\mathbf{s}$, select the k most reliable symbols in $\mathbf{r}$ and disregard the remaining $n - k$ symbols.

c) Re-encode the k most reliable symbols using a new $\mathbf{G}_n$ matrix, derived from the original $\mathbf{G}$ and equivalent to it, but with unit columns in the k most reliable positions.

One way of obtaining $\mathbf{G}_n$ is by inverting the matrix formed by the k elected columns of the original $\mathbf{G}$, then multiplying the result by $\mathbf{G}$. A major problem in this procedure is that not all sets of k columns from $\mathbf{G}$ are LI, so inversion might not be possible. In such a case, another set of k symbols must be chosen and the process repeated until k LI columns (an IS) are found. Additionally, matrix inversion is a very costly operation, in both software and hardware.

Another approach, partially based on [7], but with much simpler computations and for which a guaranteed small search space is demonstrated, was introduced in [8]. The generator matrix $\mathbf{G}$ is manipulated using Gauss-Jordan transformations, which can reduce any row or column to a unit vector. The algorithm is summarized in Fig. 1 and briefly described below, with a (7, 4) code used as an example, whose generator matrix is shown in Fig. 1(a). It is assumed that the ranked reliability values ($\mathbf{s}$) are those marked at the top of each block in Fig. 1, which range from 1 (most reliable) to 7 (least reliable).

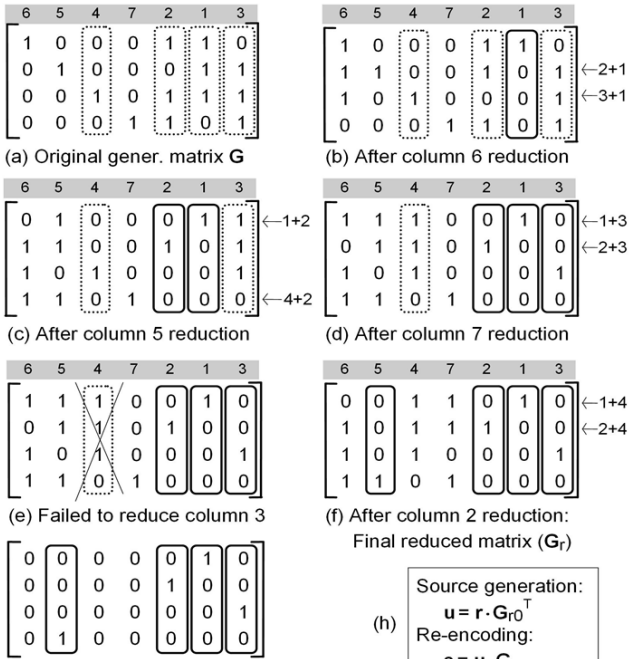

Fig. 1. Construction of the G_r and G_{r0} matrices.

a) Extract from the received codeword the hard decoded sequence **r** and the corresponding reliability sequence **s**. The values of **s** are marked at the top of Fig. 1(a).

b) Using Gauss-Jordan transformations, reduce the k most reliable columns (MRCs) of **G** to unit vectors. Even though there is no guarantee that the k MRCs are LI, the process does not need to be restarted when an LD columns is found; just replace the least reliable among the MRCs with the next MRC and proceed from there. This is illustrated in Figs. 1(b)-(f).Column 6 (the MRC) was reduced in Fig. 1(b), column 5 (the next MRC), in Fig. 1(c), then column 7, in Fig. 1(d). In Fig. 1(e), the algorithm failed to reduce column 3, indicating that the set is not LI. Column 3 was then replaced with column 2 (the next MRC), which was successfully reduced in Fig. 1(f), resulting in a fully reduced matrix $\mathbf{G}_r$.

c) Create the matrix $\mathbf{G}_{r0}$, which is simply $\mathbf{G}_r$ with all unselected columns zeroed. This is shown in Fig. 1(g).

d) Multiply **r** by $\mathbf{G}_{r0}$ to obtain the initial source message, $\mathbf{u}_0$.

e) Construct the set of all candidate messages by simply flipping one bit of $\mathbf{u}_0$ at a time. Thus the total number of candidate messages in this version of the algorithm is $k + 1$ (represented by $\mathbf{u}_i$, $i = 0$ to k).

f) Finally, re-encode each candidate message ($\mathbf{u}_i$) using $\mathbf{c}_i = \mathbf{u}_i \times \mathbf{G}_r$ to get the candidate codewords ($\mathbf{c}_i$), and measure the Euclidean distance between each of these codewords and**r** in order to decide the winner.

3 Fundamental Hardware-OrientedInformation-Set Decoder

This section introduces the hardware architecture for the basic IS decoder. As will be explained later, the number of bits used to encode each analog symbol in **x** and the maximum number of candidate codewords have a great influence on the overall decoder performance. In the descriptions that follow, we adopt 3 bits for the former and a generic value m ($\geq k + 1$) for the latter.

The hardware architecture of the fundamental IS decoder is depicted in Fig. 2. It consists of 5 main blocks: (1) input sorting and demodulation, (2) modified Gauss elimination of the **G** matrix, (3) candidate messages generation, (4) candidate codewords generation, and (5) best candidate selection.

The overall decoding process proceeds as follows. Block 1 receives a digitized version **x** of the analog received word, from which it produces the

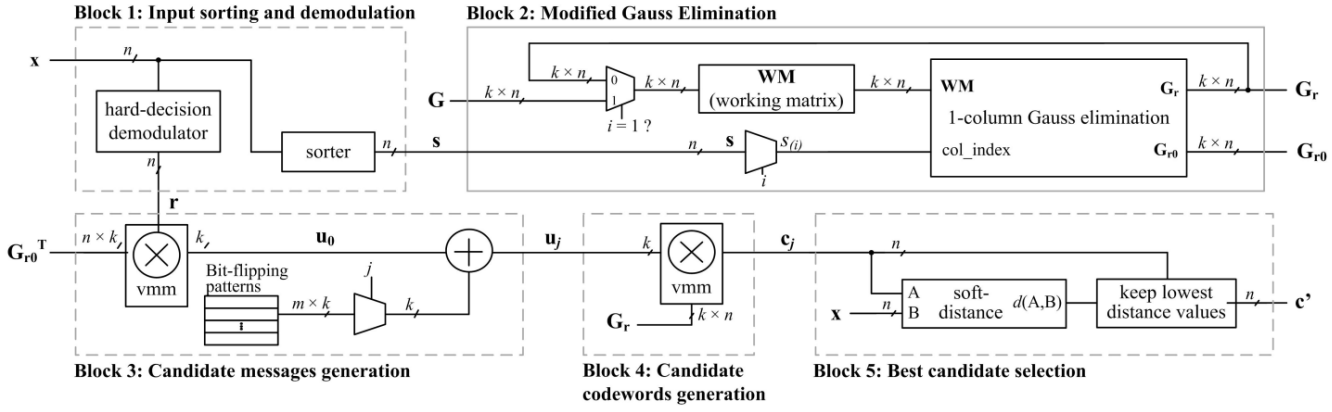

Fig. 2. Decoder hardware diagram.

reliabilities vector **s** and the received word demodulated in a hard-decision fashion **r**. Block 2 receives the original generator matrix **G** and the reliabilities vector**s**, and performs a modified Gauss elimination on G, producing two new matrices called $\mathbf{G}_r$ (reduced matrix) and $\mathbf{G}_{r0}$ (reduced transformation matrix). Block 3 receives $\mathbf{G}_{r0}$ (from block 2) and **r** (from block 1), and produces sequentially a set of *m* candidate messages $\mathbf{u}_j$. Block 4 receives $\mathbf{G}_r$ (from block 2) and the candidate messages $\mathbf{u}_j$ (from block 3), producing, for each, a corresponding candidate codeword $\mathbf{c}_j$. Block 5 receives $\mathbf{c}_j$ (from block 4) and **x**, and evaluates each of the *m* candidate codewords in order to produce the final output **c'**, which is the candidate with the smallest soft-distance with respect to **x**.

The implementation details and computational performance of each block are described next. In the detailed diagrams, clock and reset signals are omitted for better clarity.

3.1Block 1: Input Sorting and Demodulation

Block 1 produces the hard-decoded version **r** from the received word **x**, a simple operation that consists in checking the most significant bit (i.e., the sign bit) of each digitized input symbol. As shown in Fig. 3, a simple≥4 comparator suffices, since 3 bits (and therefore 8 possible levels) were employed to digitize the input analog values. In this case, values of **x** in the 0-to-3 range are decoded as zeros, while those in the 4-to-7 range are decoded as ones.

Vector **s** is a list of integers from 1 to *n*, indicating the position of each bit from**x**in order of decreasing reliability. The first value in **s** indicates the position of the most reliable bit in **x**. Because the received analog values are normalized in the −1 to +1 range, the closer a bit value is to −1 or +1, the more reliable it is; conversely, the closer it is to 0, the least reliable it is. This vector is generated by a linear insertion sorter, based on the architecture described in [9]. Since the analog values are ordered as they are shifted into the sorter, block 1 outputs become available after *n* clock cycles.

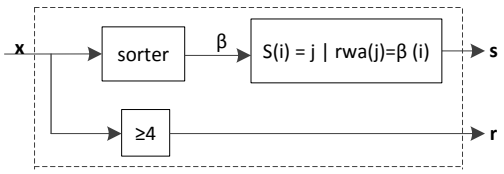

Fig. 3. Block 1: Input Sorting and Demodulation.

Sorting algorithms usually operate iteratively on an array of elements, and therefore the computational complexity is a function of the problem size, *n*. For example, the *bubblesort* algorithm presents $O(n^2)$ complexity, while *quicksort* has $O(n \log n)$. If the software runs in a single-processor CPU, no matter how optimized, this timing characteristic cannot be helped. On a custom hardware implementation, other choices are available, such as in the *insertion sorting* algorithm. This algorithm has an average and worst-case complexity of $O(n^2)$ [10], meaning that the average number of computations to sort a randomly ordered list is proportional to the square of the number of elements *n* in the list.

For a single-processor machine, this dictates that the time to sort a list grows quadratically with its size. However, although the number of computations must be the same in either hardware or software, the timing complexity can be greatly reduced. By using replication, a hardware implementation can perform many computations in parallel. For example, Fig. 4 shows a parallel implementation of a linear sorter with$O(n)$ time complexity, employed in our implementation. Because sorting is the first operation performed on the received word, it has a significant impact in the throughput of the decoder. A constant goal in out hardware implementation was to keep the number of clock cycles to decode a word as linear as possible, with respect to the size of the received word.

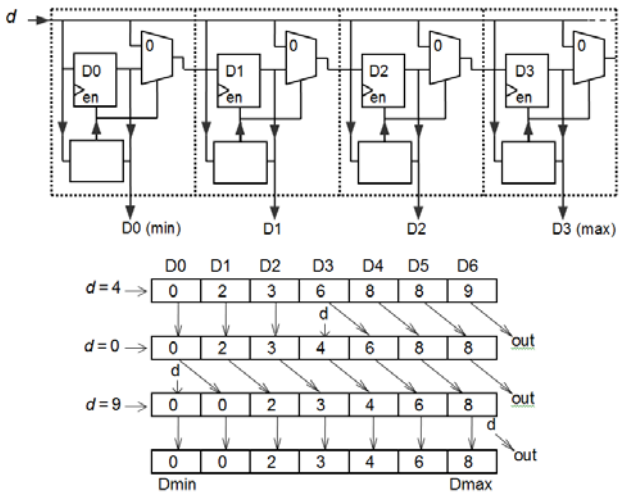

Fig. 4. Hardware implementation of a linear insertion sorter.

3.2 Block 2: Modified Gaussian Elimination

Block 2 receives the reliabilities vector **s** produced by block 1 and the code's original generator matrix **G**, from which it produces the matrices $\mathbf{G}_r$ (reduced **G** matrix) and $\mathbf{G}_{r0}$ (reduced transformation matrix) described in Section 2. As shown in Fig. 5, the

ordering of the reduced columns is dictated by **s**.The elimination proceeds until *k* LI columns are found.

In the first iteration, a temporary data structure called the *working matrix* (**WM**) is loaded with **G**. In subsequent iterations, **WM** is transformed via Gauss-Jordan operations, until an identity matrix is found. Because it may take a variable number of cycles to obtain *k* LI columns, the outputs will be available somewhere between *k* and $n - d_{min} + 1$ clock cycles, as demonstrated in Theorem 1.4.15 of [11]. $\mathbf{G}_{r0}$ will be used to extract from the received word only those bits in the positions indicated by the IS. $\mathbf{G}_r$ will be used to re-encode the *k*-bit candidate messages, generating the series of *n*-bit candidate codewords.

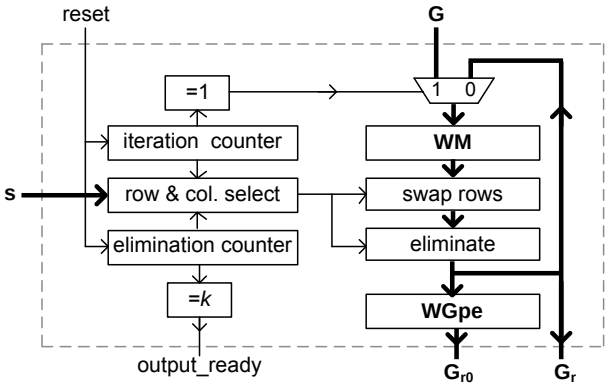

Fig. 5. Block 2: Modified Gaussian elimination.

3.3 Block 3: Candidate Messages Generation

Block 3 is responsible for generating a series of *m* candidate messages, one per clock cycle. In the subsequent stages, these candidates are evaluated to select the one that most closely resembles the received word. This is an iterative process, controlled by the iteration counter present in Fig. 6.

In order to obtain the original candidate message $\mathbf{u}_0$, the bits from **r** corresponding to the selected information set must be extracted and rearranged. This is done by multiplying the hard-decoded sequence **r** by matrix $\mathbf{Gr_0}^{\mathbf{T}}$, using a hardware structure called a *vector-matrix multiplier* (VMM), which was shown as a symbol in Fig. 2. In Fig. 6, this VMM is shown in more detail, at the circuit level. No that even though matrix operations using real numbers tend to be complex, this is not the case with boolean matrices; as can be seen in Fig. 6, this operation is performed with simple AND and XOR gates.

The fundamental version of the decoder generates *k* candidate messages, by flipping one bit at a time from the original message $\mathbf{u_0}$. For example, for a code with 4 information bits (*k*=4), the corresponding flipping patterns would be 0001, 0010, 0100, and 1000. In total, $m = k + 1$ messages are evaluated, because the original message with no bits flipped is also a valid candidate. In Sec. III, this number of candidates will be extended to improve the performance of the decoder. Additionally, more complex bit patterns will be used.

The bit-flipping patterns are stored in a ROM, and are summed with the original candidate message $\mathbf{u}_0$ to produce the series of candidate messages. This operation (a modulo-2 addition) is very simple in hardware, and is equivalent to a bitwise XOR of the original message $\mathbf{u}_0$ with each bit-flipping pattern.

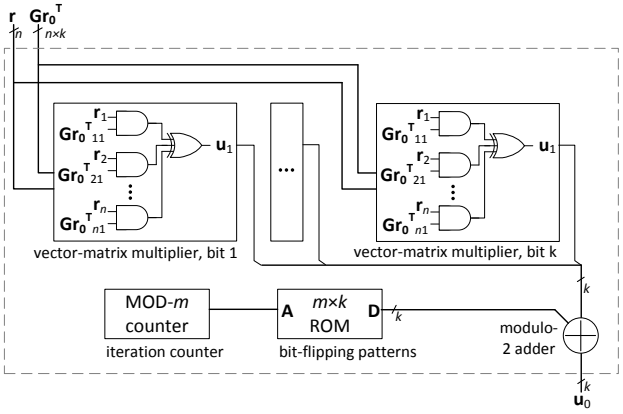

Fig. 6. Block 3: Candidate messages generation.

3.4 Block 4: Candidate Codewords Generation

Block 4 is responsible for producing a candidate codeword $\mathbf{c}_j$ from each candidate message $\mathbf{u}_j$ generated in block 3. This operation consists in re-encoding each message using the rearranged generator matrix **Gr**. As in block 3, a vector-matrix multiplier is used (Fig. 7). This operation is entirely sequential, and the outputs are available at the next clock cycle.

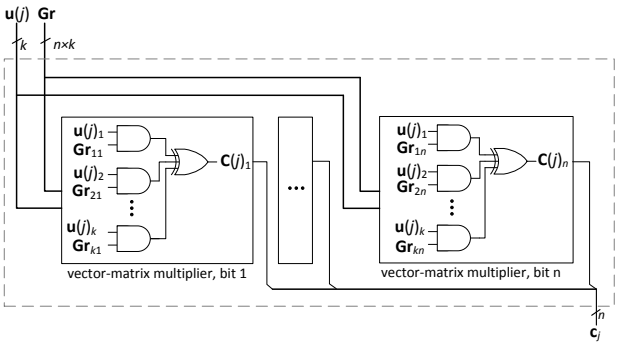

Fig. 7. Block 4: Candidate codewords generation.

3.5 Block 5: Candidates Evaluation and Selection

Block 5 is responsible for ultimately selecting the best candidate (i.e., the one that most closely resembles the analog received word) and producing the output of the decoder. In maximum-likelihood decoding (MLD), the decoding algorithm consists in selecting the codeword with the shortest Euclidean distance to the received word. In the presented architecture, a simpler soft-distance between each candidate $\mathbf{c}_{cj}$ and the received analog word $\mathbf{x}$ is calculated as described in [12]: for each received symbol represented as a 3-bit quantized level x, the bit distance is 7-x to a code bit value '1', and x to a code bit value '0'. The total distance between these two words is the sum of all individual bit distances. After calculating this metric for each candidate, the codeword with the smallest distance to the received analog word is selected as the output of the decoder. Each candidate word is evaluated in one clock cycle; therefore, if a number of candidates greater than k+1 is used, it can have a significant impact on the decoder performance.

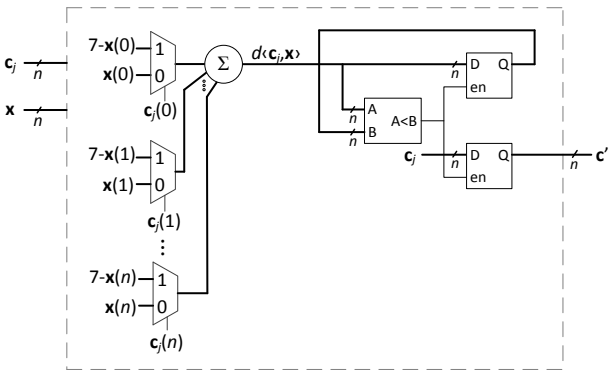

Fig. 8. Block 5: Candidates evaluation and selection.

4 Quantization Effects and Number of Bits

A crucial decision in any digital hardware implementation is the minimum number of bits needed to represent the involved signals, which should be kept as low as possible in order to save resources (area, power consumption, etc.). The purpose of this section is to investigate the effect of quantization on the IS decoder performance, and from it to derive the minimum number of bits that should be adopted.

In order to conduct such analysis, it is necessary that the signal characteristics be either well known or, at least, consistently modeled. In soft decision decoding of block codes, the received signal generally undergoes a normalization process, so the signal's mean and standard deviation get divided by a constant. Even though such normalization does not affect the standard normal variable and the bit error rate, it does change the signal's sensitivity to the number of discretization intervals.

We start the analysis by examining the influence of the number of quantization intervals on the SNR, from which we obtain analytical expressions that can be used to fine tune the number of levels used in the decoder. Next, the results are validated both by determining the convolution of the signal distribution with the quantization noise distribution, and by showing the results from Matlab simulations. Finally, the minimum number of bits is obtained and conclusions are presented.

4.1 Influence of received signal normalization on its standard deviation

For a code of length n and message length k, with BPSK modulated components of amplitude ±A, under the influence of an AWGN channel with a signal-to-noise ratio E_b/N_0, the standard deviation is [13]

$$\sigma_r = A/\sqrt{\left(2\cdot(k/n)\cdot 10^{\frac{Eb/N0}{10}}\right)} \qquad (1)$$

where the subscript r denotes *real* standard deviation (as opposed to that after signal normalization).

In order to preserve the relative symbol reliabilities, the receiver/demodulator executes signal normalization for every block of n symbols by dividing all n values by the greatest signal modulus of all n symbols. This way, symbol amplitudes become normalized between ±1. Truncation of outliers, although a common practice, will be disregarded. Such normalization produces a signal compression given by

$$\sigma_n = \frac{\sigma_r}{\mu_{ext}}, \qquad (2)$$

where the subscript n means *normalized* and μ_{ext} is the average of the *extreme* (greatest) values in each block.

The value of μ_{ext} can be derived with the help of the theory of order statistics. Using the theory in [14], for example, one can obtain the distribution for the extreme values and, from it, the mean value μ_{ext}. Fig. 9(a) shows the resulting theoretical distribution of extreme values for the $\mathcal{C}(48,24,12)$ code, for several signal-to-noise ratios. To validate these

results, simulations were performed with 10^5randomly generated codewords, and their extreme values were plotted as histograms in Fig. 9(b). As can be seen, the theoretical and the experimental values are virtually the same.

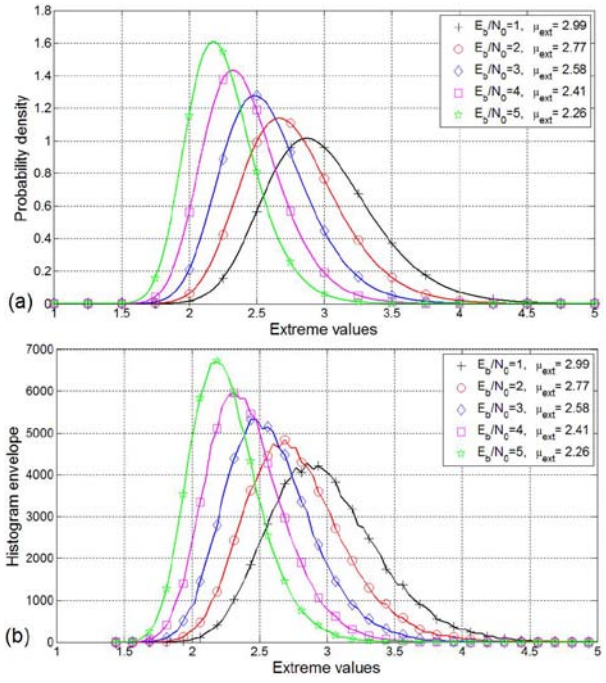

Fig. 9. (a) Theoretical extreme values distribution for *C*(48,24,12) and (b) corresponding simulation histograms.

Additional simulation results are presented in Table 1, for three codes. As can be seen in the last two columns, the theoretical and the experimental values are again very close.

4.2 Influence of number of quantization intervals

The discretization of a signal with a finite number of quantization levels, where each level has an amplitude q, introduces a random error of amplitude $\pm q/2$. In general, simulation models equate quantization in levels of amplitude q to a uniformly distributed noise between $+q/2$ and $-q/2$, with zero mean and variance $q^2/12$. However, Widrow [15] showed that such a model, called *pseudoquantizationnoise* (PQN) model, is only an approximation, valid under certain conditions. The original goal of Widrow's analysis was to investigate the conditions under which the probability distribution and other statistics of the original signal could be recovered without errors from the quantized signal. However, his results can be applied here to validate the applicability of the PQN model to our investigation.

According to Widrow, the PQN model performs well with quantization intervals up to one standard deviation, with precision better than one part in 10^7 for standard deviation estimates of Gaussian distributed signals. Dorsch [7] suggested using a quantization interval value of 0.35σ for computer simulations of quantized signals. Our study will assess the consequences of discretizing a normalized signal with from 3 to 8 quantization bits (thus with $2^3 = 8$ to $2^8 = 256$ quantization intervals). For a normalized signal in the ±1 interval, this means amplitude intervals q from $2/2^3 = 0.25$ to $2/2^8 = 0.0078127$. When compared to the σ_n values in the last columns of Table 1, $q<\sigma_n$ in all cases, warranting the application of the PQN model without further restrictions.

Table 1. Standard deviation of normalized values.

Eb / No (dB)	σ_r	σ_{ext}	μ_{ext}	$\sigma_n = \sigma_r / \mu_{ext}$	σ_n simulated
Code (15,7,5)					
1	0,9225	0,5063	2,6014	0,3546	0,3575
2	0,8222	0,4512	2,4273	0,3387	0,3412
3	0,7328	0,4022	2,2720	0,3225	0,3245
4	0,6531	0,3585	2,1337	0,3061	0,3077
5	0,5821	0,3195	2,0104	0,2895	0,2910
Code (24,12,8)					
1	0,8913	0,4558	2,7359	0,3258	0,3306
2	0,7943	0,4062	2,5471	0,3119	0,3160
3	0,7079	0,3620	2,3788	0,2976	0,3016
4	0,6310	0,3227	2,2289	0,2831	0,2866
5	0,5623	0,2876	2,0953	0,2684	0,2722
Code (48,24,12)					
1	0,8913	0,4160	2,9903	0,2981	0,3036
2	0,7943	0,3708	2,7738	0,2864	0,2915
3	0,7079	0,3304	2,5809	0,2743	0,2790
4	0,6310	0,2945	2,4090	0,2619	0,2668
5	0,5623	0,2625	2,2558	0,2493	0,2539

By applying the PQN model, the quantized signal can be represented as a random variable resulting from the sum of two independent random variables (Fig. 10); one is the Gaussian distributed normalized signal with mean $\mu_n = 1/\mu_{ext}$ and standard deviation $\sigma_n^2 = (\sigma_r \mu_{ext})^2$, and the other is a uniformly distributed variable due to noise in the $\pm q/2$ interval with mean $\mu_u = 0$ and variance $\sigma_u^2 = q^2/12$.

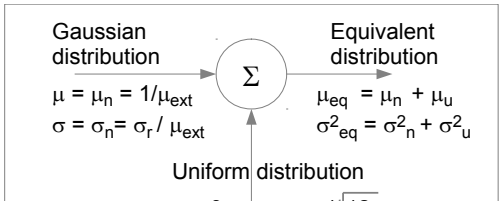

Fig. 10: PQN model for evaluation of quantization noise effects.

Since the random variables involved in the sum are statistically independent, the resulting mean will be the sum of their means, and the resulting variance the sum of their variances. By denoting the quantization interval as a fraction α of the resulting standard deviation, we can write:

$$\alpha = q/\sigma_n \rightarrow q = \alpha.\sigma_n \tag{3}$$

Therefore:

$$\sigma_{eq} = \sqrt{\sigma_n^2 + \frac{q^2}{12}} = \sigma_n\sqrt{1 + \frac{\alpha^2}{12}}, \tag{4}$$

where the subscript *eq* stands for *equivalent*, as indicated in Fig. 10.

Eq. 4 shows that the normalized signal standard deviation grows by a factor $(1 + \alpha^2/12)^{1/2}$. Furthermore, because the model is linear, the standard deviation $\sigma_{r\ eq}$ of the equivalent Gaussian distributed noise can be estimated by multiplying both members of equation (4) by μ_{ext}:

$$\sigma_{req} = \mu_{ext} \times \sigma_{eq} = \mu_{ext} \times \sigma_n\sqrt{1 + \frac{\alpha^2}{12}} = \sigma_r \times \sqrt{1 + \frac{\alpha^2}{12}} \tag{5}$$

Using equation (5), we can determine the SNR degradation due to noise. By rewriting equation (1):

$$(E_b/N_0)_r = 10\log_{10}\left(\frac{A^2}{2R\sigma_r^2}\right) \tag{5}$$

Then setting $\sigma_r = \sigma_{r\ eq}$ and using equation (5):

$$(E_b/N_0)_{req} = 10\log_{10}\left(\frac{A^2}{2R\sigma_{req}^2}\right) = 10\log_{10}\left(\frac{A^2}{2R\sigma_r^2\left(1 + \frac{\alpha^2}{12}\right)}\right) \tag{6}$$

the following results:

$$\Delta(E_b/N_0) = (E_b/N_0)_r - (E_b/N_0)_{req} = 10\log_{10}\left(1 + \frac{\alpha^2}{12}\right) \tag{7}$$

Table 2 and Fig. 11 summarize the resulting degradation for three codes. An important conclusion is that there is very little to gain by using more than 3 quantization bits, and essentially no gain at all by using 5 bits or more. For example, for the *C*(48,24,12) code operating in a channel with E_b/N_0 = 5 dB, Table 2 shows that the effect of using only 3 bits (instead of 5 or more) is the same as having a channel with E_b/N_0 reduced to 4.67 dB; or reduced to 0.77 dB if operating in a channel with E_b/N_0= 1 dB. Whether such (small) degradation is acceptable or not depends on the application, for which the quantification presented in this analysis provides the needed information. For the reasons presented above, 3 bits were employed in the hardware implementations described in this paper.

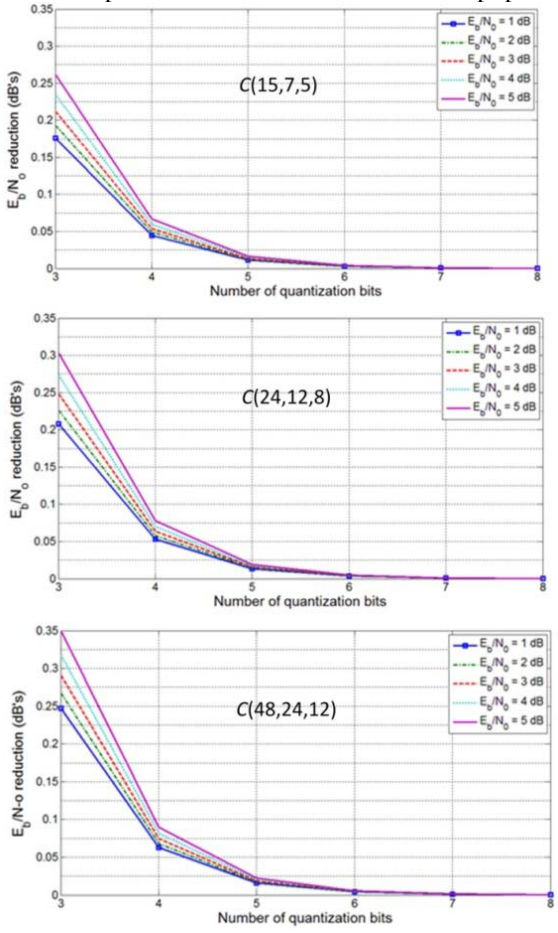

Fig. 11: SNR degradation due to quantization.

Table 2. SNR degradation due to quantization.

# of bits	E_b/N_0 = 1 dB	E_b/N_0 = 2 dB	E_b/N_0 = 3 dB	E_b/N_0 = 4 dB	E_b/N_0 = 5 dB
Code (15,7,5)					
3	0,1763	0,1928	0,2122	0,2349	0,2618
4	0,0447	0,0490	0,0540	0,0599	0,0670
5	0,0112	0,0123	0,0136	0,0151	0,0168
6	0,0028	0,0031	0,0034	0,0038	0,0042
7	0,0007	0,0008	0,0008	0,0009	0,0011
Code (24,12,8)					
3	0,2080	0,2265	0,2482	0,2734	0,3032
4	0,0530	0,0577	0,0634	0,0700	0,0778
5	0,0133	0,0145	0,0159	0,0176	0,0196
6	0,0033	0,0036	0,0040	0,0044	0,0049
7	0,0008	0,0009	0,0010	0,0011	0,0012
Code (48,24,12)					
3	0,2474	0,2674	0,2907	0,3179	0,3495
4	0,0632	0,0684	0,0745	0,0817	0,0900
5	0,0159	0,0172	0,0187	0,0206	0,0227
6	0,0040	0,0043	0,0047	0,0051	0,0057
7	0,0010	0,0011	0,0012	0,0013	0,0014

4.3 Validation of the PQN model

The previous section showed how to determine the equivalent standard deviation and corresponding degradation in SNR of the quantized signal. Because the reference model to evaluate decoding performance is that of a channel under additive white Gaussian noise, a natural question that arises is how far the quantized signal departs from this model.

The employed PQN model consists of a sum of random variables with different probability density functions (PDFs). Such sum is also a random variable, and its PDF is the convolution of the PDFs of the summed variables. We want to examine how much the result of this convolution might differ from a true Gaussian distribution.

Results from such convolution are depicted in Fig. 12. In Fig. 12(a), a ratio $\alpha = q/\sigma n = 4$ was employed, only to further highlight the difference between the convolution results a Gaussian variable with the same standard deviation. Nevertheless, even in this extreme scenario, the difference between the two curves is less than 2.5%. In the scenarios considered in our study, $\alpha < 1$, and the resulting difference is never greater than 0.4%. Fig. 12(b) shows the case for $\alpha = 1$; note that the normal curve and the convolution result are indistinguishable. This figure was vertically truncated in order to better highlight the compared curves.

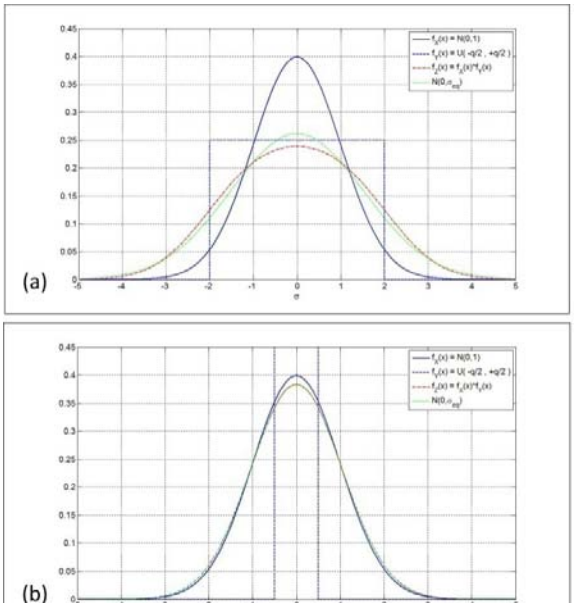

Fig. 12: Convolution of signal and quantization noise for (a) $\alpha = 4$ (β)$\alpha = 1$.

4.4 Validation of quantization effects

The previous sections showedthat the discretization of the normalized signal is equivalent to a degradation of the SNR, and provided the means to quantitatively determine this degradation.

However, the ability to estimate an equivalent SNR is no guarantee that a decoding algorithm will perform equivalently with a quantized signal as it would perform with a non-quantized signal in a correspondingly reduced signal-to-noise ratio scenario. The reasoning behind this is that the real PDF is not Gaussian in nature but, according to Widrow [15], it consists of a series of Dirac impulses of area equal to the corresponding area under the equivalent normal curve, as shown in Fig. 13.

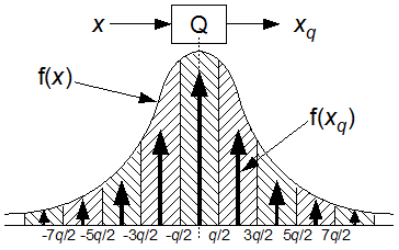

Fig. 13: Signal distributions before (x) and after (x_q) quantization.

In order to understand the distributions shown in Fig. 13, we must consider that the quantization process rounds off all values in one interval to its central value. In other words, the whole interval area

under the normal curve becomes concentrated into a single point at the center of the interval. Therefore, the impulse areas correspond to the sample values, taken at the center of the interval, of the convolution of the non-quantized signal with a rectangular pulse corresponding to the uniformly distributed PQN noise.

Given the real nature of the PDF of the quantized signal, it is reasonable to inquire whether the decoder performance is the same in both situations: with the quantized signal, or with a non-quantized signal and the correspondingly reduced signal-to-noise ratio, as derived above. In order to clear this last point, simulations were performed with 10^5 randomly generated codewords, comparing the decoder performance using both the quantized and non-quantized signals against the non-quantized version, always with the correspondingly reduced signal-to-noise ratio. As an example, Fig. 14 illustrates the assignment of discrete values to the quantization intervals used in the case of 8 quantization intervals. Fig. 15 shows simulation results for this case (8 intervals), where it can be seen that the curves for the quantized signal and for the non-quantized signal with SNR reduction practically coincide, confirming the theoretical predictions made above.

An important conclusion of this section is that the curves, tables, methods, and equations presented are a valid tool to enable a hardware designer to balance resource usage against the performance degradation introduced through discretization of the analog received signals.

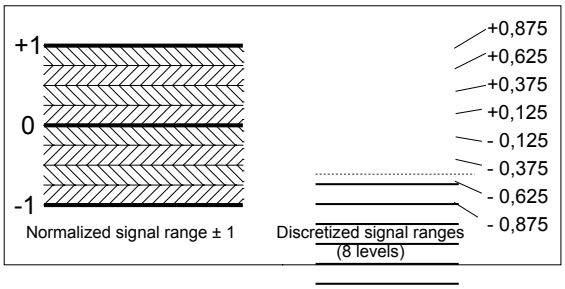

Fig. 14: Discrete values assignment for 8 quantization intervals

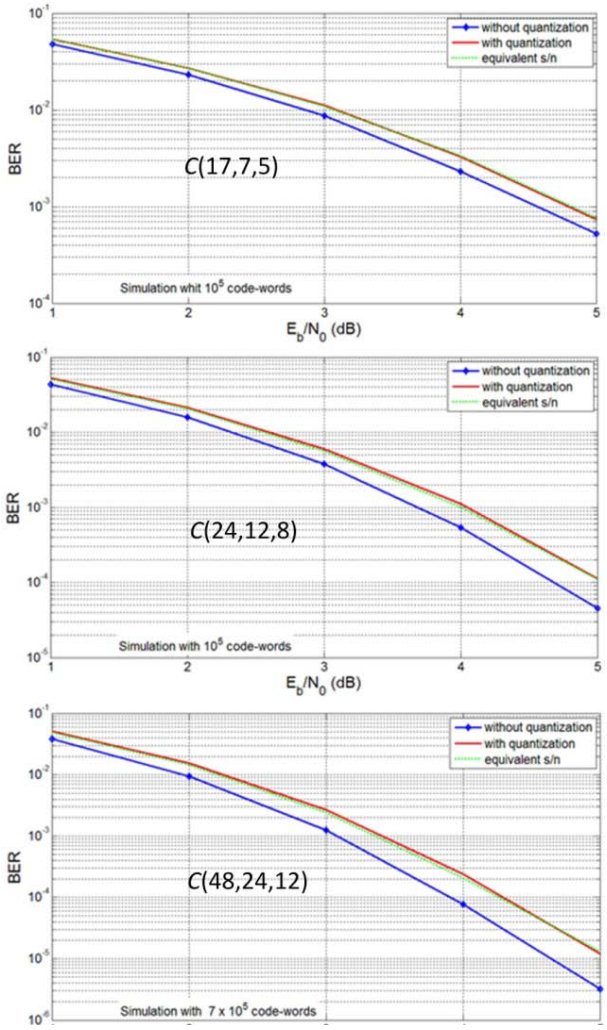

Fig. 15: Validation of the quantization effects (8 intervals) for several codes.

5 Number of Codewords

Soft maximum-likelihood decoding (MLD) algorithms consist in computing the Euclidean distance between a received sequence and all possible 2^k codewords, then choosing the candidate with the smallest distance. Longer codes offer better error-correction performance; however, the number of comparisons grows exponentially with k. Nevertheless, we demonstrate in this section that it is possible to select a relatively small number of codewords with a very high probability of containing the best candidate. This greatly reduces the number of comparisons and speeds up the decoding process.

The strategy to obtain the most likely candidates for a given block code $C(n, k, d)$ consists in the following steps:

i) Randomly generate a large number of codewords $\mathbf{c} \in C$, subjected to additive white Gaussian noise.

ii) Estimate the probability of occurrence for every possible error pattern, based on the measured number of occurrences of each error pattern.

iii) Sort the error patterns according with the measured probability.

From this, a fixed set of error patterns (called *bit-flipping patterns*, in Section 3) for a given code is obtained, which are sorted by decreasing probability of the corresponding error occurrence. The number of bit-flipping patterns (m) can be chosen according to the desired level of decoding performance. Simulations have shown that choosing m from 3% to 5% of 2^k produces results which are practically indistinguishable from true MLD results.

Once the bit-flipping patterns for the code are found, a set of highly probable candidates for each received sequence $\mathbf{x}$ can be determined as follows.

i) The n symbols of the received sequence $\mathbf{x}$ are sorted in order of decreasing reliability.

ii) The code generator matrix $\mathbf{G}$ is then manipulated via Gauss-Jordan transformations, in order to determine the positions of the k most reliable *and* linearly independent symbols of $\mathbf{x}$. These will be the positions which correspond to the first k unitary columns of the transformed matrix $\mathbf{G_r}$, as explained in Section 2.

iii) The k most reliable and linearly independent symbols from $\mathbf{x}$ are extracted, forming the most likely message $\mathbf{u}_0$ (hereafter called the *reference candidate message*), from which $\mathbf{c}$ could have been originated by multiplying $\mathbf{u_0}$ by $\mathbf{G_r}$.

iv) The bit-flipping patterns are then applied to $\mathbf{u_0}$, generating a set of alternative messages $\{\mathbf{u_0}, \mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_m}\}$. This set of messages $\mathbf{u_i}$ is then multiplied by $\mathbf{G_r}$, producing the set of candidate codewords $\{\mathbf{c_0}, \mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_m}\}$.

In general, the most likely candidates are those resulting from message words with just one bit flipped, followed by those with two bits flipped, and so on. However, this is not the case for all patterns. Moreover, it is necessary to determine which, among all candidates with the same number of bits flipped, is the most likely, so exhaustive simulations are indeed needed to rank the bit-flipping patterns.

The most relevant conclusion is that after inspecting just the candidates with one or two bits flipped the decoder already reaches a performance very near (>99%) that of MLD; this means that more than 99% of the bit errors that would be corrected with MLD decoding are corrected by the proposed approach. In other words, instead of inspecting all 2^k words (as in MLD), only very few are indeed needed to reach practically the same performance.An example is shown in Table 3, for the $C(24,12,8)$ code.

Table 3.Decoder performance relative to MLD as a function of the number of candidate codewords, for the C(24,12,8) code.

m (for k=12)	Bit-flipping pattern	Proposed / MLD coverage
1 (only $\mathbf{u}_0$)	000000000000	71.89 %
...	...	...
25	000000100010	97.97 %
...	...	...
100	000001100001	99.76 %
...	...	...

Note that after inspecting only m=25 (properly chosen) candidates, nearly 98% of the performance that would be attained with all 4,096 candidates is achieved. Observe also that with m=100 candidates (that is, just 2.44% of the possible cases), the performance is already near 99.8%. This selection procedure, which allows m to be small, is crucial to make the hardware implementation viable.

Additional results are presented in Fig. 16, showing the performance degradation for several codes with respect to MLD as a function of the number of candidates, under several noise conditions.

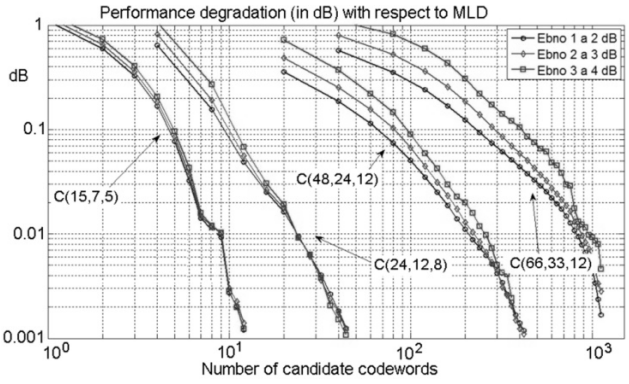

Fig. 16. Performance degradation (in dB) with respect to MLD, as a function of the number of candidate codewords.

This graphgives the number of candidates that must be evaluated to achieve the desired decoding performance. For example, to operate with at most 0.1 dB of degradation with respect to MLD, we take the corresponding horizontal line in Fig. 16, from which the following results, for the worst case (E_b/N_0 = 5 dB):

- For $C(15,7,5)$: 5 candidates, out of 128 words.

-For $C(24,12,8)$: 12 candidates, out of 4,096 words.

-For $C(48,24,12)$: 100 candidates, out of 16.8×10^6 words.

-For $C(66,33,12)$: 400 candidates, out of 8.6×10^9 words.

6 Acceptance Criteria

The original soft maximum-likelihood decoding (MLD) algorithm computes the Euclidean distance between a received sequence and all codewords in a given code, and then chooses the candidate with the smallest distance to the received sequence. On the other hand, soft information-set decoding allows the number of comparisons to be greatly reduced, while being still very likely to contain the winning codeword.

The number of comparisons can be reduced even further with the inclusion of acceptance criteria. An *acceptance criterion* (also called a *stop criterion*, *early termination rule*, or *stop rule*) is a test which can immediately identify a given codeword as the best possible candidate, which is then declared the winner, abbreviating the decoding procedure.

As a trivial example for a stop rule, consider a check of the Euclidean distance between the received sequence and a given codeword. If this distance is less than the minimum Euclidean distance of the code, the candidate can be instantly declared as the correct output of the decoder. However, although this criterion is easily applicable, it has a low efficiency because it can only identify a small fraction of the actual maximum-likelihood candidates, having therefore little impact on the average number of computations.

There are, however, other acceptance criteria, with different degrees of efficiency and computational demands. This section reviews the main stop criteria and describes a new one (HO-BGW), which is hardware-oriented and is used in the final implementation of our IS decoder.

Before we start, it is important to mention that the efficiency of a stop rule can be immensely enhanced by ordering the candidates according to some probability measure (so the most likely are tested earlier), which indeed occurs in the proposed method.

6.1 Generalized Minimum Distance (GMD) Criterion

The GMD acceptance criterion [5] states that for a given code of length n and minimum Hamming distance d_{Hmin}, given a received sequence $\mathbf{x}$ and a candidate codeword $\mathbf{c}$ modulated in BPSK, if the condition

$$\langle \mathbf{x}, \mathbf{c} \rangle > n - d_{Hmin} \tag{9}$$

is satisfied, where $\langle \mathbf{x},\mathbf{c} \rangle$ denotes the inner product between $\mathbf{x}$ and $\mathbf{c}$, then $\mathbf{c}$ is unique, $\langle \mathbf{x},\mathbf{c} \rangle = \langle \mathbf{x},\mathbf{c} \rangle_{\max}$, and therefore the Euclidean distance between $\mathbf{x}$ and $\mathbf{c}$ is minimum.

Although the GMD criterion has a simple implementation, it is not used in practice because of its low efficiency. As will be shown later, simulations of GMD with a $C(15,7,5)$ code indicate an average reduction in the number computations of only 1.77%, for an E_b/N_0 ratio of 5 dB.

6.2 Hypercone Rule

The hypercone rule [16][17] is based on the principle that if the angle between the received codeword and a candidate is less than half the smallest angle between any two codewords, then the received word is within the Voronoi region of the candidate. This criterion is valid for all cases where all codewords have the same module. It can be expressed mathematically as follows:

$$\cos(\alpha) = \frac{\langle \mathbf{c}, \mathbf{v} \rangle}{\sqrt{n}\|\mathbf{v}\|} > \cos\left(\frac{\delta_{min}}{2}\right) = \sqrt{1 - d_{Hmin}/n} \tag{10}$$

or

$$\langle \mathbf{c}, \mathbf{v} \rangle > \|\mathbf{v}\|\sqrt{1 - d_{Hmin}/n}, \tag{11}$$

where the radical on the right side of equation (11) is always a constant, dependent only on the code parameters d_{Hmin} and n.

Later, Godoy et al [17] demonstrated that a received vector $\mathbf{v}$ belongs to the Voronoi region of a candidate $\mathbf{c}$ if and only if the hybrid sum of $\mathbf{v}$and$\mathbf{c}$ belongs to the Voronoi region of the null codeword $\mathbf{c_0}$, that is:

$$\mathbf{v} \in V(\mathbf{c}) \Leftrightarrow (\mathbf{v}[+]\mathbf{c}) \in V(\mathbf{c}_0), \tag{12}$$

where $V(\mathbf{c})$ stands for the Voronoi region of $\mathbf{c}$. On the other hand, it is also possible to set $\mathbf{c}$ to the null codeword $\mathbf{c_0}$ in equation (11). If all components are BPSK encoded, equation (11) is reduced to:

$$\langle \mathbf{c}_0, \mathbf{v} \rangle = -\sum_{i=1}^{n} v_i > \|\mathbf{v}\|\sqrt{1 - d_{Hmin}/n} \tag{13}$$

Denoting the hybrid sum of **v** and **c** by **vs**, and its components by vs_i, the hypercone acceptance criterion can be rewritten as:

$$\sum_{i=1}^{n} vs_i < -\|\mathbf{v}\|\sqrt{1 - d_{Hmin}/n}, \qquad \mathbf{vs} = \mathbf{v}[+]\mathbf{c} \tag{14}$$

This last form is more efficient when the hybrid sum has already been computed by another algorithm, or when computing the hybrid sum is simpler than computing the inner product.

6.3 Taipale-Pursley Criterion

The acceptance criterion proposed by Taipale and Pursley [3] can be summarized as follows. Consider a code *C* with minimum Hamming distance d_{Hmin}, a received sequence **x** (whose symbols $x_j$delivered by the demodulator have a reliability measure β_j, $0< \beta_j< 1$), and a candidate codeword **c**. The latter is the best candidate if the following condition is satisfied:

$$\sum_{j\in S} \beta_j \le \sum_{j\in T} \beta_j, \tag{15}$$

where *S* is the set of indices *j* such that the components c_j (of **c**) and x_j (of **x**) have opposite signs, i.e., $S = \{j \mid \text{sgn}(c_j) \neq \text{sgn}(x_j),\ 0 \le j \le n\}$ (this set has cardinality $|S|$). *T*is the set of $\delta = d_{Hmin} - |S|$ indexes in the complementary set of *S* that have the smallest values for β_j.

6.4 Barros-Godoy-Wille (BGW) Criterion

It was demonstrated in [16] that the BGW criterion has a better performance than GMD or Hypercone. It was also demonstrated independently in [18] that its performance is equivalent to that of Taipale-Pursley; however, the BGW criterion employs a different approach, much simpler to implement in hardware. To better understand its derivation, we will initially review the definitions of the Voronoi region and hybrid sum [18].

Definition: The Voronoi region $V(\mathbf{c_i})$ of a sequence $\mathbf{c_i} \in C$ is defined as the set of vectors $x \in \mathbb{R}^n$ that are close to $\mathbf{c_i}$, obeying the following relationship:

$$V(c_i) = \{x \in \mathbb{R}^n | d_E^2(\mathbf{x}, \mathbf{c_i}) \le d_E^2(\mathbf{x}, \mathbf{c_k}), \forall \mathbf{c_k} \in C\} \tag{16}$$

Definition: The hybrid sum operation [+] between a sequence $\mathbf{x} \in \mathbb{R}^n$ and a sequence $\mathbf{c_i} \in C$ is defined by the equation below, where $\mathbf{x}.\mathbf{c_i}$ is the term-by-term product of **x** and $\mathbf{c_i}$:

$$\text{x}[+]\text{c}_\text{i} = -(\text{x}.\,\text{c}_\text{i}) \tag{17}$$

Property 1: The operation [+] simply changes the sign of the components of **x** if the correspondent components of $\mathbf{c_i}$ are equal to one. The reliabilities of the symbols in the sequence $(\mathbf{x}[+]\mathbf{c_i})$ are the same as in the sequence **x**.

Theorem: Consider a code *C* with minimum Hamming distance d_{Hmin}, a codeword $\mathbf{c_i}$ (with components c_{ij}), and a sequence **x**, (with components x_j). The codeword $\mathbf{c_i}$ is considered the best candidate codeword for **x** if the following condition is satisfied:

$$\sum_{j\in Q}(x_i[+]c_{ij}) \le 0, \tag{18}$$

where *Q* is the set of d_{Hmin} indexes *j* such that the components in the sequence $(\mathbf{x}[+]\mathbf{c_i})$ have the highest numerical values (most positive or least negative).

Proof: Suppose that$\mathbf{x} \in V(\mathbf{c_i})$; then, by Definition 1:

$$d_E^2(\mathbf{x}, \mathbf{c_i}) \le d_E^2(\mathbf{x}, \mathbf{c_k}), \forall \mathbf{c_k} \in C \tag{19}$$

Writing the Euclidean distance in terms of inner product:

$$d_E^2(\mathbf{x}, \mathbf{c_i}) = |\mathbf{x}|^2 - 2(\mathbf{x}.\,\mathbf{c_i}) + |\mathbf{c_i}|^2 \tag{20}$$

Inequality (20) can be rewritten as:

$$-2.\sum_j (x, c_{ij}) \le -2.\sum_j (x_j, c_{kj}), \forall \mathbf{c_k} \in C \tag{21}$$

Assuming that $W = \{j: c_{ij} \neq c_{kj,} 0 \le j < n\}$, with $\bar{W}$ as its complementary set, we have:

$$-\sum_{j\in W}(x.c_{ij}) - \sum_{j\in \bar{W}}(x_j.c_{ij}) \le -\sum_{j\in W}(x_j.c_{kj}) - \sum_{j\in \bar{W}}(x_j.c_{kj}), \forall \mathbf{c_k} \in C \tag{22}$$

Observing that $c_{ij} = -c_{kj}, \forall j \in W$ and $c_{ij} = c_{kj}, \forall j \in \bar{W}$, the second sum in both sides cancel out and we obtain:

$$-\sum_{j\in W}(x_j.c_{ij}) \le 0, \forall \mathbf{c_k} \in C \tag{23}$$

Using Definition 2:

$$\sum_{j\in W}(x_j[+]c_{ij}) \le 0, \forall \mathbf{c_k} \in C \tag{24}$$

It is sufficient to consider in the equation above only the closest codewords, such that $|W|= d_{Hmin}$.

Finally, we consider the following property: if the sum of the d_{Hmin} components of highest numerical values in a given sequence is less or equal to zero, than the sum considering any other d_{Hmin} components is also less or equal to zero. Therefore, the equation above is satisfied if:

$$\sum_{j\in Q}(x[+]c_{ij}) \leq 0, \tag{25}$$

which concludes the proof.

Corolary 1: The sequence **x** belongs to the Voronoi region of the zero codeword $\mathbf{c_0}$ if:

$$\sum_{j\in Q} x_j \leq 0 \tag{26}$$

This criterion was called Sum Rule, and its performance was demonstrated to be superior (less stringent) than that obtained with the GMD or the hypercone criterion [16][17][18].

6.5 Hardware-Oriented BGW (HO-BGW)

The fundamental motivation for the *HO-BGW*(*Hardware-Oriented BGW*) criterion was to produce a feasible hardware implementation, while keeping a performance level close to the original BGW.

Acceptance tests are one of the final steps in the decoding process. Because they must be executed once for each candidate, they should not take longer than generating such a codeword, otherwise the test would represent a bottleneck for the entire decoding process.

Despite being a major constraint, this problem exists in all of the previous criteria. Moreover, such criteria employ complex operations, such as summations, sorting, and other iterative procedures, and require several clock cycles to be evaluated. The HO-BGWcriterion overcomes this limitation by making use of the already available symbol reliability ordering. Recall that the BGW requires the symbols to be ordered according to their signed values, in order to isolate the d_{Hmin} most positive ones. While symbol reliability ordering is based on absolute values instead of signed values, this ordering can still be used if sign information for each symbol is available. Fortunately, as will be shown, in soft IS decoding the candidates generation is such that sign information will always be available for the *k* most reliable symbols. To make this statement clear, an example is provided below.

Consider a *C*(15,7,5) code and a received sequence **x**. Applying the information set decoding algorithm, a set of *q* candidates $\{\mathbf{c}_0, \mathbf{c}_1, \ldots \mathbf{c}_{q-1}\}$ can be obtained, whose hybrid sum with **x** generates the set $\{\mathbf{x'}_0, \mathbf{x'}_1, \ldots \mathbf{x'}_{q-1}\}$. Fig. 17(a) illustrates the reliabilities-based ordering of $\mathbf{x'}_0$.

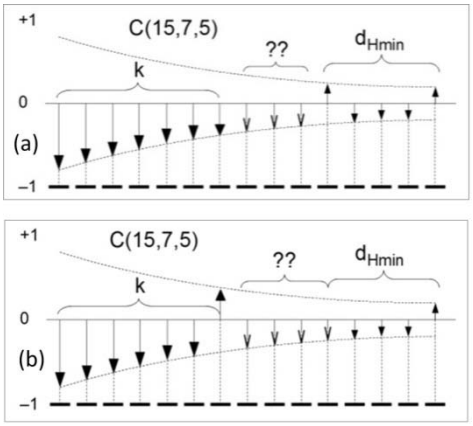

Fig. 17. Reliability-based ordering of (a) $\mathbf{x'}_0$ and (b) $\mathbf{x'}_1$ components.

The dashed line indicates the absolute value ordering, obtainedwithy standard IS decoding. The *k* leftmost (and most reliable) symbols were obtained by abrupt decision. Since these symbols were used to reconstruct the codeword $\mathbf{c}_0$, those positions have the same polarity in **x** and $\mathbf{c}_0$. It follows, by definition of the hybrid sum, that the first *k* positions of $\mathbf{x'}_0$ are all negative. For these reasons, the most positive values in $\mathbf{x'}_0$ (which would be used by the original BGW algorithm) will be in the d_{Hmin} rightmost positions. These values must be summed and their sum must result negative. We will refer to this requirement as condition C_1. The values to be summed in order to verify condition C_1 can be selected by a mask m_1. In this example, the value of m_1 for the first candidate is:

$$m_{1,0} = \{0000000\ 000\ 11111\} \tag{27}$$

However, the stop condition will only be valid if the central $k - d_{Hmin}$ positions in $\mathbf{x'}_0$ (labeled with question marks in Fig. 17(a)) are negative. These values can be selected by a mask m_2, which indicates the positions that should not be negative for the rule to be valid:

$$m_{2,0} = \{0000000\ 111\ 00000\} \tag{28}$$

We will refer to this second condition as C_2. In the case of Fig. 17(a), the sum of the five rightmost values of $\mathbf{x'}_0$ is negative (condition C_1) and the

central $k - d_{Hmin}$ positions are all negative (condition C_2), so $\mathbf{c}_0$will be declared as the best possible candidate.

At this point, the HO-BGW algorithm would stop, but, in order to better illustrate the algorithm operation, we proceed to analyze the next iteration. Fig. 17(b) illustrates the reordering of the next hybrid sum in the set, $\mathbf{x'}_1$. This hybrid sum was obtained from $\mathbf{c}_1$, in which one of the symbols (the least reliable among the most reliable k symbols) was flipped, according to the bit-flipping pattern 0000001. In this case, the flipped symbol is not only positive, but also, due to the reliability ordering, more positive than any of the rightmost d_{Hmin} symbols, and so it should be exchanged with one of the d_{Hmin} rightmost symbols.

This is the point where the HO-BGW deviates from the original BGW. The flipped position should replace the most negative of the least reliable d_{Hmin} symbols; however, here only absolute reliability information (and not polarity) is available, so we settle for substituting the most reliable among the least reliable d_{Hmin} symbols, noting that this will be valid only if the replaced symbol was indeed negative. This is the reason why we need to check condition C_2 in addition to C_1, and also the reason why the HO-BGW is slightly less efficient than the original BGW criterion: it will not be able to operate on the (small number of) candidates that have positive valuesin the set of symbols selected by mask m_2 (indicated by question marks in Figs. 17(a)-(b)).

For the hybrid sum $\mathbf{x'}_1$, the values of m_1 and m_2 are:

$$m_{1,1} = \{0000001\ 000\ 01111\} \quad (29)$$

$$m_{2,1} = \{000000011110000\} \quad (30)$$

The generation of masks m_1 and m_2 is simple: the first k bits of m_1 are the same as the bit-flipping patterns used to generate the candidate codewords. The last bit positions will be filled with as many ones as necessary for the mask m_1 to have exactly d_{Hmin} bits one. As for mask m_2, its first k positions will always be zero and its last $n - k$ positions will be the complement of the $n - k$ bit positions of m_1.

6.6 Comparison between BGW and HO-BGW

A direct comparison between these two acceptance criteria is presented below.

BGW

1) Sort $\boldsymbol{x'}_n$ in descending order of signed values.
2) Sum the first (most positive) d_{Hmin} values.
3) If this sum is negative, **c** is the best candidate.

HO-BGW

1) Sort $\boldsymbol{x'}_n$ in descending reliabilities order (absolute values).
2) Sum the values in the positions indicated by mask m_1; condition C_1 is true if this sum is negative.
3) Check the values in the positions indicated by m_2; condition C_2 is true if none is positive.
4) If C_1 and C_2 are true, **c** is the best candidate.

An important conclusion from this comparison is that they are simple enough for the HO-BGW criterion to be incorporated in the hardware, without substantial additional resources. Moreover (and very importantly), the HO-BGW test can be performed in a single clock cycle.

6.7 Comparison of stop criteria efficiency

A numeric performance comparison between BGW and HO-BGW is presented in Table 4. As can be seen, HO-BGW achieves its best result with the *C*(24,12,8) code, with a reduction of 96.8% in the number of candidates that must be evaluated. The performance difference between BGW and HO-BGW varies with the code length and the signal-to-noise-ratio, but it is always under ~12% for the testedcodes. On average, the HO-BGW criterion provided a reduction of 69.6% on the number of candidates that must be evaluated, against 77.9% of the original BGW criterion.

Table 4. Reduction in the number of iterations (in %).

E_b/N_0 (dB)	*C*(15,7,5)		*C*(24,12,8)		*C*(48,24,12)		Average		
	BGW	BGWG	BGW	BGWG	BGW	BGWG	BGW	BGWG	Differ.
1	55.7	55.5	60.7	55.5	27.2	16.0	47.9	42.3	5.5
2	69.2	67.2	77.8	69.6	52.7	31.5	66.6	56.1	10.5
3	81.3	78.7	90.3	82.6	79.1	53.2	83.6	71.5	12.1
4	89.9	87.6	96.8	91.9	94.7	75.0	93.8	84.8	9.0
5	94.4	93.1	98.8	96.8	99.3	90.3	97.5	93.4	4.1
							77.9%	69.6%	8.2%

7 Enhanced Hardware-Oriented Information-Set Decoder

This section introduces the final hardware implementation for ourIS decoder with the HO-BGW acceptance criterion included. The circuit, based on that introduced in Fig. 2, is shown in Fig. 18. Afundamental feature of this implementation is that the acceptance test of each candidate is performed in a single clock cycle. Because the generation of a candidate also requires one clock

cycle, no bottleneck is imposed by the introduction of the stop criterion.

As can be seen in Fig. 18, new hardware structures were added to implement the HO-BGW acceptance criterion (indicated by the lines and blocks in bold). Note that the computations related to the HO-BGW criterion were distributed through all five stages. The modifications performed on each stage of the decoder are summarized below.

Block 1: The permutation matrix $\mathbf{P}_s$ is generated, with its elements defined by $ps_{ij} = \{0 \mid i \neq s_{ij},\ 1 \mid i = s_{ij}\}$. When $\mathbf{P}_s$ is multiplied by the received sequence **x**, the components of **x**are reordered such that the first element is the most reliable one.

Blocks 2 and 3: During Gaussian elimination, it is possible that some symbols fall into LD positions and cannot be used in the decoding procedure. For this reason, matrix $\mathbf{P}_s$ is rearranged into matrix $\mathbf{P}_{sI}$, which will have only LI columns on its left side. In Block 2, two auxiliary matrices are created: $\mathbf{P}_{sLI}$, with only LI columns, and $\mathbf{P}_{sLD}$, with only LD columns. In Block 3, these matrices are concatenated into $\mathbf{P}_{sI}$, which has LI columns on the left, LD columns in the center, and the remaining columns copied directly from $\mathbf{P}_s$.

Block 4: Besides generating the candidate codewords $\mathbf{c}_i$, Block 4 calculates the hybrid sum $\boldsymbol{x}'_n = \boldsymbol{x}\ [+]\ \boldsymbol{c}_n$ and two bitmasks, used to compute conditions C_1 and C_2. These bitmasks are rearranged versions of m_1 and m_2, sorted by reliability, and given by $m_1r(i) = m_1(i)\cdot\mathbf{P}_{sI}^{T}$ and $m_2r(i) = m_2(i)\cdot\mathbf{P}_{sI}^{T}$.

Block 5: Finally, conditions C_1 and C_2 are calculated, as described in Section 6.5. These calculations are simple and can be performed at the same time as the next candidate is being generated in Block 4.

8 Results

8.1 Fundamental Hardware-Oriented IS Decoder

This is the initial version of the decoder shown in Fig. 2, which does not include yet the acceptance criterion. The VHDL description was synthesized to a high-end Altera Stratix III FPGA (EP3SL150F780C2) for several code sizes, as shown in Table 5. A coverage performance of 99.0% relative to MLD was specified for all decoder implementations. The hardware correctness was confirmed by an exhaustive testbench simulation of the $C(7,4,3)$ code, which yielded correct outputs for all 2^{21} possible input values (7 input symbols encoded with 3bits each).

Table 5. Synthesis Results for the Fundamental IS Decoder.

Code	Registers	ALUTs	f_{MAX} (MHz)	Latency (cycles)	ttd_{MAX} (cycles)	Throughput (Mbps)
C(7,4,3)	290	454	155.3	19	5	124,24
C(15,7,5)	763	1,359	127.7	40	12	74,49
C(24,12,8)	2,155	4,019	101.9	84	41	29,82
C(48,24,12)	31,300	61,980	74.4	667	580	3,08

Regarding silicon area usage, it was found that logic resources utilization (look-up tables and registers) increases almost linearly with the $n\times k$ product; however, since the list size m also increases for larger codes (in order to attain the same performance relative to the MLD), the observed growth is in fact more pronounced. It should be noted that even the $C(48,24,12)$ code fits in a mid-range device of the Stratix III family, indicating that the hardware implementation is indeed area-efficient.

8.2 EnhancedHardware-Oriented IS Decoder

This implementation concerns the complete decoder (Fig. 18), with the HO-BGW acceptance test included. It was also implemented in VHDL language, then synthesized to a high-end FPGA of the Altera Stratix IV family (EP4SGX70DF29C2X)

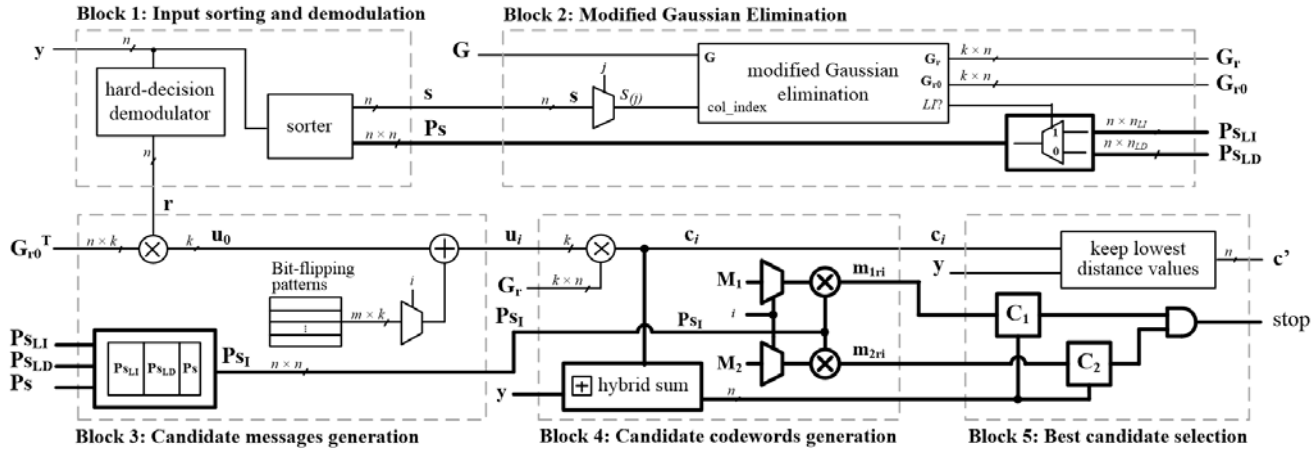

Fig. 18. Hardware implementation of the IS decoder with the HO-BGW acceptance criterion

for several code sizes, as shown in Table 6. The correctness of the hardware implementation was again confirmed by an exhaustive testbench simulation of the *C*(7,4,3) code, which yielded correct outputs for all 2^{21} possible input values 7 input symbols encoded with 3bits each).

Table 6. Synthesis Results for the Enhanced IS Decoder.

	C(7,4,3)	C(15,7,5)	C(24,12,8)	C(48,24,12)	Avg. Increase (*)
LUTs	894	1,688	3,095	9,197	92.0 %
Registers	860	1,729	3,011	7,556	180.0 %
f_{MAX} (MHz)	145.3	127.6	107.9	92.7	6.0 %

(*) Compared to the fundamental decoder implementation of Fig. 2.

As can be seen, the HO-BGW requires 92% more lookup tables (LUTs) and 180% more registers (flip-flops), on average. Roughly speaking, the inclusion of the HO-BGW stop test doubles the decoder's hardware size. On the other hand, it provides a reduction of up to 96.8% in the number of codewords that must be examined. Since the maximum operating frequency stays essentially the same, this represents a total throughput of up to 30 times the original value.

The resulting circuit is also a highly area-efficient implementation, with the *C*(48,24,12) code occupying only 20% of the smallest FPGA in the Stratix IV family.

9 Conclusions

We have presented three improvements that made viable the implementation of information-set decoders purely in hardware. First, we havestudied the effects of quantization in the decoding performance, allowing us to choose a number of bits that is small, and still provides a negligible performance loss. Second, we have introduced a procedure to select a small number of candidate codewords, with a high likelihood of containing the correct MLD output. Third, we have presented a new hardware-oriented acceptance criterion, which is highly efficient and feasible for a hardware implementation.

Those three improvements were combined in the implementation of an enhanced information-set decoder purely in hardware. The proposed architecture can implement any linear block decoder and is highly area-efficient, with the C(48,24,12) code occupying only 20% of the smallest FPGA in the Stratix IV family. The inclusion of the HO-BGW stop test roughly doubles the hardware size;on the other hand, it provides a reduction of up to 96.8% in the number of examined codewords. This represents a total throughput of up to 30 times the original value.

Quantization of the received signal is a procedure common to all soft decoders. Our investigation demonstrated that there is very little to gain by using more than 3 quantization bits for each analog symbol, and essentially no gain at all by using 5 bits or more. The method and results supporting this conclusion are useful for any designer when choosing the number of quantization bits for a decoder.

We have also presented a strategy forselecting a number of candidate codewords that allows achieving any desired performance level relative to MLD. We have shown that choosing sets as small as 3% to 5% of all 2^k possible values still produces results that are practically indistinguishable from true MLD.The most relevant conclusion is that after inspecting just the candidates with one or two bits flipped, the decoder already reaches a performance very close to that of MLD (>99%); this means that more than 99% of the bit errors that would be corrected with MLD decoding are corrected with the proposed approach.

Acceptance tests are one of the final steps in the decoding process. Because they must be executed once for each candidate word, they should not take longer than generating such a codeword, otherwise the test would represent a bottleneck for the entire decoding process. We have demonstrated the efficiency of the new HO-BGW acceptance criterion, which provided a reduction of 96.8% in the number of candidates evaluated for the C(24,12,8) code. The performance difference varies with the code length and the signal-to-noise ratio, but it is always within 12% compared to the common Taipale-Pursley criterion. The HO-BGW criterion is the first acceptance criterion that allows a direct hardware implementation.

References:

[1] R. Gallager, "Low-density parity-check codes," *IRETrans. on Information Theory*, vol. 8, no. 1, pp. 21-28, Jan. 1962.

[2] T. Richardson, R. Urbanke, "The renaissance of Gallager's low-density parity-check codes," *IEEE Communications Magazine*, vol. 41, no. 8, pp. 126- 131, Aug. 2003.

[3] D. J. Taipale, M. B. Pursley, "An improvement to generalized minimum distance decoding," *IEEE TIT*, vol. 37, pp. 167-172, Jan. 1991.

[4] B. Vucetic, V. Ponampalam, J. Vuckovic, "Low complexity soft decision algorithms for Reed-Solomon codes," *IEICETrans. on*

Communications, vol. E84-B, no. 3, March 2001.

[5] G. D. Forney Jr., "Generalized minimum distance decoding," *IEEETrans. on Information Theory*, vol. IT-12, no. 2, April 1966.

[6] E. Prange, "The use of information sets in decoding cyclic codes," *IRE Trans. on Information Theory*, vol. IT-8, pp. 5-9, Sep. 1962.

[7] B. G. Dorsch, "A decoding algorithm for binary block codes and j-ary output channels," *IEEE Trans. on Information Theory*, vol. IT-20, pp. 391-394, May 1974.

[8] A. Gortan, R. P. Jasinski, W. Godoy Jr., V. A. Pedroni, "Achieving near-MLD performance with soft information-set decoders implemented in FPGAs," *APCCAS*, Dec. 2010.

[9] L. Ribas, D. Castells, J. Carrabina, "A linear sorter core based on a programmable register file", *DCIS*, pp. 635-640, France, 2004.

[10] D. Knuth, *The Art of Computer Programming: Sorting and Searching* (vol. 3), Addison-Wesley, 1998.

[11] W. Huffman, V. Pless, *Fundamentals of Error-Correcting Codes*, Cambridge University Press, 2003.

[12] P. Sweeney, *Error Control Coding From Theory to Practice*, Wiley, 2002.

[13] R. Blahut, E. Richard, *Algebraic Codes for Data Transmission*, Cambridge University Press, New York, 2003.

[14] C. Barry, N. Balakrishnam, H. Nagaraja, *A first course in order statistics*, *SIAM Classics in Applied Mathematics*, vol. 14, Philadelphia, 2008.

[15] B. Widrow, I. Kollár, *Quantization Noise – Round off error in Digital Computation, Signal Processing, Control and Communications*, Cambridge University Press, New York, 2008.

[16] D. J. Barros, W. Godoy Jr., E. G. Wille, "A new approach to the information set decoding algorithm," *Computer Communications*, vol. 20, pp. 302-308, 1997.

[17] W. Godoy Jr., E. G. Wille, "Proposal of sub-optimum decoding algorithm with a bound of Voronoi region $V(c_0)$," *Computer Communications*, Vol. 21, 736-740, 1998.

[18] W. Godoy Jr., E. G. Wille, T. Cunha, "Adaptive decoding of binary linear block codes using information sets and erasures," *CTRQ*, Athens, 2010.

[19] W. K. Leung, W. L. Lee, A. Wu, Li Ping, "Efficient implementation technique of LDPC decoder," *Electronics Letters*, vol. 37, no. 20, pp. 1231-1232, Sep. 2001.

[20] Zin-She Yang, *Introduction to Computational Mathematics*, World Scientific, 2008.

[21] D. MacKay, R. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, no. 6, pp. 457-458, March 1997.