www.ietdl.org

Published in IET Communications
Received on 19th October 2011
Revised on 29th February 2012
doi: 10.1049/iet-com.2011.0767

ISSN 1751-8628

Field programmable gate arrays implementations of low complexity soft-input soft-output low-density parity-check decoders

L.J. Arnone[1] J. Castiñeira Moreira[1] P.G. Farrell[2]

[1]Electronics Department, Engineering School, Mar del Plata University, Mar del Plata, Argentina
[2]School of Computing and Communications, Lancaster University, Lancaster, UK
E-mail: casti@fi.mdp.edu.ar

Abstract: Low-density parity-check (LDPC) codes are very efficient error control codes that are being considered for use in many next-generation communication systems. In this study low complexity soft-input, soft-output (SISO) field programmable gate arrays (FPGA) implementations of a novel logarithmic sum-product (LogSP) iterative LDPC decoder and a recently proposed simplified soft Euclidean distance (SSD) iterative LDPC decoder are presented, and their complexities and performance are compared. These implementations operate over any choice of parity check matrix (including those randomly generated, structurally generated and either systematic or non-systematic) and can be parametrically adapted for any code rate. The proposed implementations are both of very low complexity, because they operate using only sums, subtractions, comparisons and look-up tables, which makes them particularly suitable for FPGA realisation. The SSD decoder has a lower implementation complexity than the LogSP LDPC decoder and it also offers the advantage of not requiring knowledge of the channel signal-to-noise ratio, unlike most other LDPC decoders.

1 Introduction

Low-density parity-check (LDPC) codes are receiving particular interest in practical applications because of their impressive bit error rate (BER) performance, which for a large code length n is very close to the Shannon limit [1]. A well-known algorithm for iterative decoding of LDPC codes is the sum-product (SP) belief propagation algorithm originally proposed by Gallager [2], and rediscovered by MacKay and Neal [1]. It has recently been shown by two of the authors of this paper that the SP algorithm can be simplified significantly by using logarithmic processing, leading to a novel logarithmic sum-product (LogSP) algorithm [3, 4].

Another new iterative algorithm for decoding LDPC codes is based on the Euclidean metric, and is called soft-distance (SD) decoding [5, 6]. This algorithm is also based on a bipartite (Tanner) graph, but it utilises the squared Euclidean distance as a metric, and sum-antilog operations for performing belief propagation. A much simpler version of the SD decoder, the simplified soft Euclidean distance (SSD) algorithm, has been created by combining it with the logarithmic processing structure devised for the LogSP algorithm [3, 7]. Monte Carlo simulations show that there is no BER performance degradation for the LogSP and SSD algorithms when compared to the SP algorithm, as we indicate below. Both the SD and SSD algorithms have the advantage of not requiring knowledge of the signal-to-noise ratio (SNR) at the output of the channel, whereas the LogSP algorithm does need the SNR.

Field programmable gate arrays (FPGAs) are widely used in practical implementations of error-control and signal-processing systems, because of their many advantages. As efficient general-purpose logic devices they can implement a great range of coding and processing algorithms, and they are relatively easy to re-structure and re-programme [8]. However, most iterative decoding algorithms for LDPC codes are in general very complex, and involve operations such as products and quotients, which are quite difficult to implement in programmable logic such as FPGA [8]. In this paper, effective FPGA decoder implementations are presented for the low complexity LogSP and SSD algorithms. These generic algorithms are particularly suited to FPGA implementations, because they use only sums, subtractions. comparisons and look-up tables. Unlike other FPGA implementations [9–14], these decoders can operate over any kind of parity check matrix $\boldsymbol{H}$, whether randomly or structurally generated, systematic or non-systematic [15, 16], and also they can easily be set to any code rate k/n.

The rest of this paper is organised as follows. A short outline of LDPC decoding is described in Section 2. The LogSP decoding algorithm and its FPGA decoder structure and parameters are explained in Sections 3 and 4. The SSD decoding algorithm and its FPGA decoder are similarly described in Sections 5 and 6. Complexity aspects are introduced in Section 7, and decoding simulation results are presented in Section 8. Section 9 concludes the paper with a summary of the main characteristics of the algorithms and their advantages.

2 LDPC decoding algorithms

LDPC codes are linear block codes [4] for which there exists a parity-check matrix $\boldsymbol{H}$ such that the number of non-zero elements in the matrix is very much less than the number of zeros. The aim of a decoding algorithm is to find a good estimate $\hat{\boldsymbol{c}}$ of the codeword [1, 4]. In the case of a soft-input, soft-output (SISO) decoder, the estimate $\hat{\boldsymbol{c}}$ may or may not satisfy the following condition

$$\boldsymbol{H} \bullet \boldsymbol{c} = 0 \tag{1}$$

However, the estimates in $\hat{\boldsymbol{c}}$ will be the best a posteriori soft values for each decoded symbol.

Decoding of these codes is based on a bipartite (Tanner) graph that represents the relationships between the symbol nodes $1, 2, \ldots j \ldots n$ and the parity-check nodes $1, 2, \ldots i \ldots n-k$, which are specified by the parity-check equations of the code. The decoding process is characterised by an iterative interchange of information (belief propagation) between symbol nodes (which represent bits of the received vector $\boldsymbol{r}$) and parity-check nodes (which represent the parity-check equations of the code) [2, 4]. This iterative process of interchanging information is stopped if condition (1) is verified, or if a given number of iterations is reached.

Table 1 Summary of the LogSP algorithm

Initialisation
$q_{ij}^0 = f_j^0 \; q_{ij}^1 = f_j^1$
Horizontal step 1
$\delta q_{ij} = \text{máx}(-q_{ij}^0, \; -q_{ij}^1) - f_{-}(q_{ij}^0, q_{ij}^1)$
$s_{ij} = 0$ if $-q_{ij}^0 \geq -q_{ij}^1$ else $s_{ij} = 0$
Horizontal step 2
$\delta r_{ij} = \Sigma \delta q_{ij} - \delta q_{ij}, \; s\delta r_{ij} = \Sigma_{N(i)} s_{ij} - s_{ij}$
Horizontal step 3
if $s\delta r_{ij}$ even
$r_{ij}^0 = \log(2) - f_{+}(\delta r_{ij}, 0)$
$r_{ij}^1 = \log(2) + f_{-}(\delta r_{ij}, 0)$
if $s\delta r_{ij}$ odd
$r_{ij}^0 = \log(2) + f_{-}(\delta r_{ij}, 0)$
$r_{ij}^1 = \log(2) - f_{+}(\delta r_{ij}, 0)$
Vertical step
$c_{ij}^x = -f_j^x - \Sigma_{M(j)} r_{ij}^x + r_{ij}^x$
$q_{ij}^0 = -c_{ij}^0 + \text{máx}(c_{ij}^0, c_{ij}^1) + f_{+}(c_{ij}^0, c_{ij}^1)$
$q_{ij}^1 = -c_{ij}^1 + \text{máx}(c_{ij}^0, c_{ij}^1) + f_{+}(c_{ij}^0, c_{ij}^1)$
Estimation of decoded symbol $\hat{r}_j$
$c_j^x = -f_j^x + \Sigma_{M(j)} r_{ij}^x$
$\hat{r}_j = 0$ if $c_j^0 \geq c_j^1$ else $\hat{r}_j = 1$

3 LogSP decoding algorithm

The low complexity logarithmic version of the SP algorithm, LogSP, has been described in [3, 4]. For convenience, a brief description of this algorithm is developed below.

We assume that the code words are transmitted in normalised polar format with amplitudes ± 1. If y_j is the received symbol from the channel at instant j, then the initial probability F_j^x that the jth symbol is x, $(x = \{0,1\})$ is given by MacKay and Neal [1]

$$F_j^1 = \mathrm{e}^{-f_j^1} = \frac{1}{1 + \mathrm{e}^{-2y_j/\sigma^2}} \tag{2}$$

$$F_j^0 = \mathrm{e}^{-f_j^0} = 1 - F_j^1 \tag{3}$$

where f_j^x is related to F_j^x by an exponential operation and σ^2 is the noise variance. Table 1 summarises the steps of this algorithm. For a given parity-check matrix $\boldsymbol{H}$, $N(i)$ is the set of indices of columns that have non-zero elements in row i, and $M(j)$ is the set of indices of rows with non-zero elements in column j. The logarithmic process can be significantly simplified by introducing correction factors called $f_{+}(a, \; b)$ and $f_{-}(a, \; b)$, which in a practical implementation, are determined by means of look-up tables with entry $|a - b|$ [3, 17], as described in Appendix 1.

The algorithm starts with the initialisation step, in which the estimate values q_{ij}^x $(x \in \{0, \; 1\})$ are set to the initial symbol estimate values obtained from the channel output, so that $q_{ij}^x = f_j^x$. The following step, called horizontal step 1, involves the calculation of the quantities δq_{ij}. Parameter s_{ij} determines the value of the addition performed in the following step, called horizontal step 2, where quantities δr_{ij} and $s\delta r_{ij}$ are determined. Even and odd values of $s\delta r_{ij}$ define in each case the calculation of quantities r_{ij}^x in horizontal step 3. Knowledge of the r_{ij}^x values allows us to determine the quantities c_{ij}^x that in turn lead to the updating of quantities q_{ij}^x. This is known as the vertical step. A similar procedure allows the evaluation of quantities c_j^x that then leads to an updated estimate of the decoded symbol for the current iteration. The updated (soft) symbol estimate is finally output from the decoder once the pre-determined number of iterations have been completed.

4 LogSP decoder architecture

The FPGA implementation of the LogSP decoder can be seen in Fig. 1. Memory ROM_*H* stores the positions of ones in the parity-check matrix $\boldsymbol{H}$. Its size depends on the number of ones per row, and the number of rows of the matrix. For instance, in our implementation we used randomly generated parity-check matrices $\boldsymbol{H}_1$ and matrix $\boldsymbol{H}_2$ of dimensions (60×30) and (1008×504) [18]; so the size of ROM_*H* is 256 words of 6 bits each, and the size of ROM_*H* is 4096 words of 10 bits each, respectively.

ROM_num_ones_rows stores the indices of the non-zero elements of each row. This memory is used together with ROM_*H* to determine the positions of ones in each parity check matrix. Memories ROM_*f*table+ and ROM_*f*table− contain the look-up tables $f_{+}(a, \; b)$ and $f_{-}(a, \; b)$, respectively. Both tables consist of 256 words of 16 bits each.

RAM_f_0 and RAM_f_1 store values of f_j^0 and f_j^1, respectively, each time a received word is input to the decoder. Its size is the code block length n. Memories

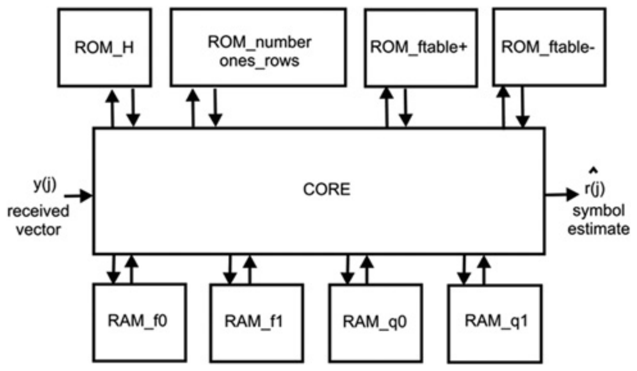

Fig. 1 *LogSP decoder architecture FPGA implementation using ALTERA [5]*

RAM_q_0 and RAM_q_1 contain the same number of words as ROM_H, but they are of 16 bits. Memories RAM_q_0 and RAM_q_1 are used for many purposes. They store values of q_{ij}^0 and q_{ij}^1 in the initialisation step. RAM_q_0 stores values of δq_{ij} and RAM_q_1 stores the values of s_{ij}, in horizontal step 1. They store values of δr_{ij} and $s\delta r_{ij}$ in horizontal step 2, and values of r_{ij}^0 and r_{ij}^1 in horizontal step 3. Finally, they store values of q_{ij}^0 and q_{ij}^1 in the vertical step. Thus, there is a high degree of reuse for these memories.

5 SSD decoding algorithm

In the SSD algorithm, with the same initial assumptions as in Section 3, probabilities are replaced by squared Euclidian distance values [5, 7]. Following a procedure detailed in [6], the use of logarithmic calculations leads to the SSD decoding algorithm [19]. Table 2 summarises the SSD algorithm. It can be seen that the calculations involved are sums, subtractions, comparisons and look-up tables.

We describe below the proposed algorithm and its simplified version. More detailed descriptions can be found in [6].

5.1 Soft distance (SD) iterative decoding algorithm: an introduction

We will briefly describe the SD algorithm by applying it to the factor (Tanner) graph [1, 2, 19] of a binary LDPC error-correcting code. As usual, the graph has n symbol nodes and m parity nodes, with edges, as defined by the parity-check matrix of the code, connecting the symbol and parity nodes. SD metric information will be passed from symbol nodes to parity nodes (the horizontal step in the algorithm), and then from parity nodes to symbol nodes (the vertical step), in an iterative manner.

Assuming the transmission of coded binary information $\boldsymbol{c} = (c_1 \quad c_2 \quad \ldots \quad c_j \quad \ldots \quad c_n)$ in normalised polar format with signals of amplitudes ± 1, and for a received vector $\boldsymbol{r} = (r_1 \quad r_2 \quad \ldots \quad r_j \quad \ldots \quad r_n)$, we calculate the vectors

$$d_0^2(j) = (r_j + 1)^2, \quad d_1^2(j) = (r_j - 1)^2, \quad j = 1, 2, \ldots, n \tag{4}$$

These first soft estimates of the code symbols are used to initialise the algorithm by setting the following coefficients q_{ij}^0 and q_{ij}^1 at each symbol node

$$q_{ij}^0 = d_0^2(j), \quad q_{ij}^1 = d_1^2(j), \quad j = 1, 2, \ldots, n, \quad i = 1, 2, \ldots, m \tag{5}$$

These are then passed to the parity-check nodes (first horizontal step) connected to each symbol node, where the following coefficients are computed

$$\begin{aligned} r_{ij}^0 &= -\log_2 \sum_{s:s_j=0} [2^{-(\Sigma_{k\in N(i)\backslash j} q_{ik}^x)}] \\ r_{ij}^1 &= -\log_2 \sum_{s:s_j=1} [2^{-(\Sigma_{k\in N(i)\backslash j} q_{ik}^x)}], \quad x \in \{0, 1\} \end{aligned} \tag{6}$$

Here $N(i)\backslash j$ is the set of indexes of the symbol nodes connected to the parity node h_i, excluding the symbol node s_j. The inner summations add together the q values corresponding to all the possible 0-state or 1-state configurations of the $N(i)\backslash j$ code symbols connected to parity check node h_i, and the negative antilogs of these coefficient sums are then added in the outer summation. Coefficients r_{ij}^0 and r_{ij}^1 are then passed back (first vertical step) to symbol node s_j. At each symbol node s_j the initial values of q_{ij}^0 and q_{ij}^1 are added to the r values passed from each node connected to it, to form a second symbol estimate at each node

$$q_{ij}^0 = d_0^2(j) + \sum_{k\in M(j)\backslash i} r_{kj}^0, \quad q_{ij}^1 = d_1^2(j) + \sum_{k\in M(j)\backslash i} r_{kj}^1 \tag{7}$$

Here $M(j)\backslash i$ is the set of indexes of all the parity nodes connected to the symbol node s_j, excluding parity node h_i. These horizontal and vertical steps are then iterated an appropriate number of times to give a final estimate for each received symbol, given by

$$\hat{d}_j^2 = \arg\min_x \left[d_x^2(j) + \sum_{k\in M(j)} r_{kj}^x \right] \tag{8}$$

Since the SD algorithm and the LogSP algorithm [1, 2, 19] have a similar factor graph processing structure, the new algorithm may seem to be a variant of the LogSP algorithm and its simplified versions, but this is not the case. The fundamental difference lies in the metric used: in the new algorithm the variables being computed are just soft (Euclidean) distances, not log-likelihoods.

5.2 Simplified version of the SD algorithm

The SD antilog-sum algorithm described in the previous section can be simplified, to create the SSD algorithm.

Determining the number of state configurations required to calculate the expressions in (3) is a complex task. This problem can be controlled by adapting a technique originally proposed by MacKay and Neal [1]. This involves

Table 2 Summary of the SSD algorithm

Initialisation
$q_{ij}^0 = d_0^2(j)$, $q_{ij}^1 = d_1^2(j)$
Horizontal step
$a_{ij} = f_+(q_{ij}^0, q_{ij}^1)$
$b_{ij} = -f_-(q_{ij}^0, q_{ij}^1)$
If $(-q_{ij}^0) \geq (-q_{ij}^1)$ $s_{ij} = 0$ else $s_{ij} = 1$
$c_{ij} = \Sigma_{k\in N(i)\backslash j} b_{ik} - \Sigma_{k\in N(i)\backslash j} a_{ik}$
if Σs_{ik} is even
$r_{n,ij}^0 = -f_+(0, c_{ij})$
$r_{n,ij}^1 = f_-(0, c_{ij})$
if Σs_{ik} is odd
$r_{n,ij}^0 = f_-(0, c_{ij})$, $r_{n,ij}^1 = -f_+(0, c_{ij})$
Vertical step
$q_{ij}^0 = d_0^2(j) + \Sigma_{k\in M(j)\backslash i} r_{n,kj}^0$, $q_{ij}^1 = d_1^2(j) + \Sigma_{k\in M(j)\backslash i} r_{n,kj}^1$
Decision
$\hat{d}_0^2(j) = r_{n,ij}^0 + q_{ij}^0$, $\hat{d}_1^2(j) = r_{n,ij}^1 + q_{ij}^1$
if $\hat{d}_1^2(j) < \hat{d}_0^2(j)$ then $\hat{c}(j) = 1$ else $\hat{c}(j) = 0\hat{c}(j) = 0$

working with sums and differences of distance antilogs, as the following expressions for the horizontal step indicate. We define

$$Q_{ij}^x = b^{-q_{ij}^x}, \quad R_{ij}^x = b^{-r_{ij}^x}, \quad x \in \{0, 1\} \tag{9}$$

and

$$A_{ij} = Q_{ij}^0 + Q_{ij}^1 = b^{a_{ij}}, \quad B_{ij} = Q_{ij}^0 - Q_{ij}^1 = (-1)^{s_{ij}} b^{b_{ij}} \tag{10}$$

$$R_{ij}^0 + R_{ij}^1 = \prod_{k\in N(i)\setminus j} A_{ik}, \quad R_{ij}^0 - R_{ij}^1 = \prod_{k\in N(i)\setminus j} B_{ik} \tag{11}$$

and so

$$\begin{aligned} R_{ij}^0 &= \frac{1}{2}\left(\prod_{k\in N(i)\setminus j} A_{ik} + \prod_{k\in N(i)\setminus j} B_{ik}\right) \\ R_{ij}^1 &= \frac{1}{2}\left(\prod_{k\in N(i)\setminus j} A_{ik} - \prod_{k\in N(i)\setminus j} B_{ik}\right) \end{aligned} \tag{12}$$

Equations (7) involve logarithmic operations over sums or subtracts, which can be solved as detailed in Appendix 1. Similar expressions apply to the vertical step.

Although working with the sums and differences of the distance antilogs helps to reduce overflow and underflow problems, we have found that overflow problems still remain when using a relatively large number of decoding iterations. In order to avoid this, R_{ij}^x values are normalised in the following way at the end of each iteration

$$R_{N,ij}^x = R_{ij}^x\left(\frac{2}{R_{ij}^0 + R_{ij}^1}\right) = R_{ij}^x\left(\frac{2}{D_{ij}}\right) \tag{13}$$

$$r_{n,ij}^x = -\log_b(R_{N,ij}^x) \tag{14}$$

By implementing all the above simplifying procedures, as detailed in [6], the proposed algorithm ends at the form of the so-called SSD algorithm, whose final form can be seen in Table 1.

Finally, and as also described in [6], some consideration has to be given to the selection of parameter b, the base of the logarithmic calculation for evaluating functions $f_+(\alpha, \beta)$ and $f_-(\alpha, \beta)$. This parameter depends on the rate R_c of the code, and it has to be properly selected for obtaining the best BER performance. Results shown in this paper are for codes of different rates in order to give examples of the different cases.

6 SSD decoder architecture

The FPGA implementation of the SSD decoder is seen in Fig. 2. As before, memory ROM_H stores the positions or indices of the ones of the parity check matrix $\boldsymbol{H}$. The number of words and of bits per word is the same as in Section 4, as the same matrices $\boldsymbol{H}_1$ (60 × 30) and $\boldsymbol{H}_2$ (1008 × 504) are used. ROM_num_ones_rows stores the indices of the non-zero elements of each row. This memory is used together with ROM_H to determine the positions of ones in each parity-check matrix. Memories ROM_$f$table+ and ROM_$f$table− contain the look-up tables $f_+(a, b)$ and $f_-(a, b)$, respectively, both having 256 words of 16 bits each.

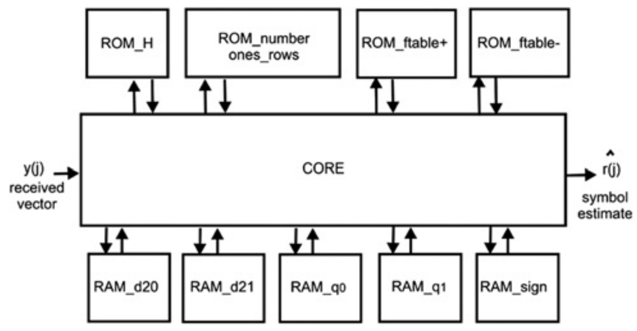

Fig. 2 *SSD decoder architecture FPGA implementation using ALTERA [5]*

RAM_d_{20} and RAM_d_{21} store values of d_0^2 and d_1^2, respectively, each time a received word is input to the decoder. Its size is the code block length n. Memories RAM_q_0 and RAM_q_1 contain the same number of words as ROM_H, but they are of 16 bits. As before, memories RAM_q_0 and RAM_q_1 are used for many purposes. They store values of d_0^2 and d_1^2 in the initialisation step. RAM_q_0 stores values of a_{ij} and RAM_q_1 stores values of b_{ij}, in the horizontal step, as well as the values of $\Sigma_{k\in N(i)\setminus j} b_{ik}$, $\Sigma_{k\in N(i)\setminus j} a_{ik}$, r_{ij}^0 and r_{ij}^1. Finally, values of q_{ij}^0 and q_{ij}^1 are stored in the vertical step. Thus, there is a high degree of reuse for these memories. Memory RAM_sign stores s_{ij} in the horizontal step and $s\delta r_{ij} = \Sigma_{N(i)} s_{ij} - s_{ij}$ in the horizontal step.

7 Complexity aspects

In order to analyse the complexity of the two decoding algorithms, we first define $t = M(j)_{\text{av}}$ as the average number of ones per column, and $v = N(i)_{\text{av}}$ as the average number of ones per row, where $m = n - k$. Usually it is true that [4]

$$v = nt/m \tag{15}$$

For an LDPC code of rate 1/2, $m = n/2$, then

$$v = 2t \tag{16}$$

Table 3 shows a comparison of the complexity of the two simplified decoding algorithms LogSP and SSD, together with the corresponding complexities of the original decoding algorithms SP and SD, for codes of rate 1/2 [the (60,30) LDPC and the (1008,504) LDPC codes], for which $t = 3$ and $v = 6$. As can be seen, the SSD algorithm involves slightly fewer calculations that the LogSP algorithm. Even though the SSD and LogSP algorithms require more sums and subtracts than the classic SP algorithm, they do not make use of products or quotients, and thus are algorithms of significantly less complexity.

Table 3 Complexity analysis of LDPC decoders ($t = 3$ and $v = 6$)

Algorithm	SD	SSD	SP (MacKay–Neal)	LogSP
products	–	–	36n	–
quotients	–	–	6n	–
sums	402n	72n	15n	78n
comparisons	90n	21n	–	24n
look-up tables	18n	12n	–	21n

8 Simulation results

FPGA implementations of the corresponding LogSP and SSD decoding algorithms were designed for the (60,30) LDPC code, and the (1008,504) LDPC code, as well as the (96,32) and (273,191) LDPC codes, using the VHDL programming language [20]. QUARTUS II from ALTERA [21] has been used as a synthesis tool. Decoding algorithms were implemented in the Altera DE2 Development Board [22]. The DE2 board contains a Cyclone II 2C35 FPGA. Cyclone II 2C35 FPGA includes 33 216 logic elements, 105 4K RAM blocks, 483 840 total RAM bits, 4 PLLs, 475 user I/O pins and a FineLine BGA 672-pin package [8]. Maximum clock frequency has been determined using the classic timing analyser tool of the program QUARTUS II [21].

Tables 4–7 show the characteristics of the implementation of each FPGA decoder. The SSD decoder FPGA implementations are faster than the LogSP decoder FPGA implementations for all the codes, and they use smaller numbers of logic components and registers.

With proper design of the look-up tables for the log functions $f_+(a, b)$ and $f_-(a, b)$ (typically tables of 256 entries), there is no significant BER performance degradation with respect to the use of the ideal function. Fig. 3 shows the BER performance of the (60,30) LDPC code and the (1008,504) LDPC code, for the SSD and LogSP algorithms. Fig. 4 shows the BER performance of the (96,32) LDPC code and the (273,191) LDPC code, for the SSD and LogSP algorithms. In addition, there are no significant differences between the performance of the original and simplified algorithms [6, 19].

Table 4 Characteristics of the logSP and SSD decoder implementations for the (60,30) LDPC code

(60,30) LDPC code		
hardware used	logSP	SSD
device	EP2C35F672C6	EP2C35F672C6
family	cyclone II	cyclone II
logic elements	1036	882
registers	394	387
memory bits	15 968	20 320
clock frequency	101.67 MHz	112.38 MHz

Table 5 Characteristics of the logSP and SSD decoder implementations for the (1008,504) LDPC code

(1008,504) LDPC code		
hardware used	logSP	SSD
device	EP2C35F672C6	EP2C35F672C6
family	cyclone II	cyclone II
logic elements	1109	951
registers	435	428
memory bits	210 944	219 136
clock frequency	94.43 MHz	118.11 MHz

Table 6 Characteristics of the logSP and SSD decoder implementations for the (96,32) LDPC code

(96,32) LDPC code		
hardware used	logSP	SSD
device	EP2C35F672C6	EP2C35F672C6
family	cyclone II	cyclone II
logic elements	1074	720
registers	410	388
memory bits	28 352	32 448
clock frequency	102 MHz	135.67 MHz

Table 7 Characteristics of the logSP and SSD decoder implementations for the (273,191) LDPC code

(273,191) LDPC code		
hardware used	logSP	SSD
device	EP2C35F672C6	EP2C35F672C6
family	cyclone II	cyclone II
logic elements	1074	915
registers	415	406
memory bits	41 984	45 568
clock frequency	95.36 MHz	106.17 MHz

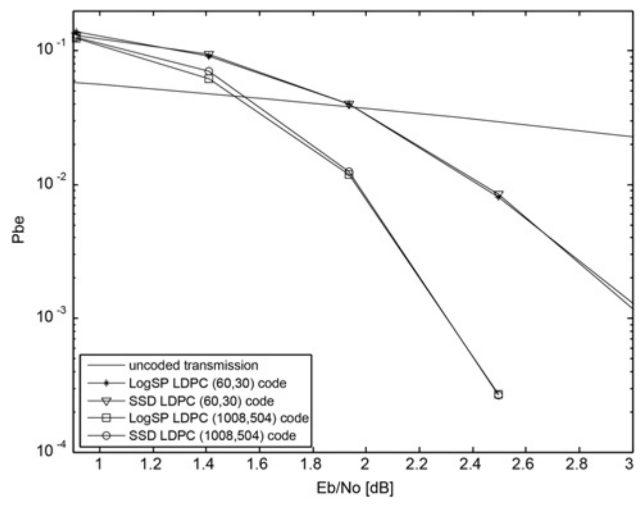

Fig. 3 *BER performances of the (60,30) LDPC decoder and of the (1008,504) LDPC decoder for the implemented decoding algorithms*

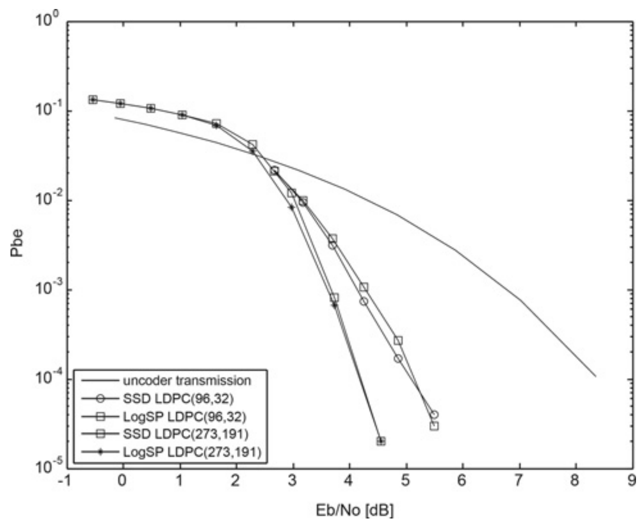

Fig. 4 *BER performances of the (96,32) LDPC decoder and of the (273,191) LDPC decoder for the implemented decoding algorithms*

9 Conclusions

In this paper, we have presented designs for FPGA implementations of SISO decoders for the (60,30) LDPC code, the (1008,504) LDPC code, the (96,32) LDPC code and the (273,191) LDPC code, using novel LogSP and SSD iterative algorithms. These are generic algorithms which can be used with codes of any rate and parity-check matrices of arbitrary structure. The algorithms are particularly suited to FPGA implementation, because they do not involve the use of multiplications or divisions, only requiring sums, subtractions, comparisons and look-up tables. The look-up tables replace ideal logarithm calculations, but there is no significant loss of performance if the table is carefully designed; we have found 256 entries to be perfectly adequate. The LogSP and SSD algorithms have virtually the same BER performance as the original SP and SD algorithms from which they are derived [19].

The SSD algorithm has two advantages over the LogSP algorithm: it has a slightly lower complexity, but more significantly the decoding process does not require knowledge of the SNR of the channel. This latter advantage is important because it removes the need to measure the SNR, or to provide a means of compensating for the lack of knowledge of the SNR. Both these alternatives add significantly to the complexity and practicality of effective decoding. However, we emphasise that both the SSD and LogSP algorithms are significantly less complex than the original SD and SP algorithms, indicating that it is possible and practical to design low complexity SISO decoders for powerful LDPC codes with no loss of performance.

10 References

1 MacKay, D.J.C., Neal, R.M.: 'Near Shannon limit performance of low density parity check codes', *Electron. Lett.*, 1997, **33**, pp. 457–458

2 Gallager, R.G.: 'Low density parity check codes', *IRE Trans. Inf. Theory*, 1962, **IT-8**, pp. 21–28

3 Arnone, L., Gayoso, C., González, C., Castiñeira, J.: 'Sum-subtract fixed point LDPC decoder', *Latin Am. Appl. Res.*, 2007, **37**, pp. 17–20

4 Castiñeira Moreira, J., Farrell, P.G.: 'Essential of error-control coding' (John Wiley and Sons, 2006), Ch. 8

5 Farrell, P.G.: 'Decoding error-control codes with soft distance as the metric'. Proc. Workshop on Mathematical Techniques in Coding Theory, Edinburgh, UK, 2008

6 Farrell, P.G., Arnone, L., Castiñeira Moreira, J.: 'Euclidean distance soft-input soft-output decoding algorithm for LDPC codes', *IET Commun. Inst. Eng. Technol*, 2011, **5**, (16), pp. 2364–2370

7 Farrell, P.G., Castiñeira Moreira, J.: 'Soft-input soft-output Euclidean distance metric iterative decoder for LDPC codes'. Proc. Argentine Symp. on Computing Technology (AST 2008), Santa Fe, Argentina, 2008

8 http://www.altera.com, 'Cyclone II Device Family Technical Information'

9 Zhang, T., Parhi, K.: '54 Mbps (3,6)-regular FPGA LDPC Decoder'. Signal Process. Syst., 2002. (SIPS'02). IEEE Workshop, 2002, vol. 1, pp. 127–132

10 Bhagawat, P., Uppal, M., Choi, G.: 'FPGA based implementation of decoder for array low-density paritycheck codes'. Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing, 2005. (ICASSP'05), 2005, vol. 5, pp. 29–32

11 Beuschel, C., Pfleiderer, H.: 'FPGA implementation of a flexible decoder for long ldpc codes'. Int. Conf. on Field Programmable Logic and Applications, 2008, FPL 2008. vol. 1, pp. 185–190

12 Cui, Z., Wang, Z.: 'A 170 Mbps (8176, 7156) quasi-cyclic LDPC decoder implementation with FPGA'. Proc. 2006 IEEE Int. Symp. on Circuits and Systems, 2006, ISCAS 2006. vol. 1, pp. 5095–5098

13 Chen, X., Kang, J., Lin, S., Fellow, L., Akella, V.: 'Memory system optimization for FPGAbased implementation of quasi-cyclic LDPC codes decoders', *IEEE Trans. Circuits Syst. I: Regular Papers*, 2011, **58**, (1), pp. 98–111

14 Zarubica, R., Wilson, S., Hall, E.: 'Multi-Gbps FPGA based low density parity check (LDPC) decoder design'. IEEE Global Telecommunications Conf., 2007. GLOBECOM'07, vol. 1, pp. 548–552

15 Sha, J., Gao, M., Zhang, Z., Li, L., Wang, Z.: 'A memory efficient FPGA implementation of quasi-cyclic LDPC decoder'. Proc. Fifth WSEAS Int. Conf. on Instrumentation, Measurement, Circuits and Systems, 2006, vol. 1, pp. 218–223

16 Ku, M.K., Li, H.S., Chien, Y.H.: 'Code design and decoder implementation of low-density parity-check code'. Emerging Information Technology Conf., 2005

17 Bhatt, T., Narayanan, K., Kehtarnavaz, N.: 'Fixed point DSP implementation of low-density parity check codes'. Proc. IEEE DSP2000, 2000

18 http://www.inference.phy.cam.ac.uk/mackay/CodesGallager.html

19 Farrell, P.G., Arnone, L., Castiñeira Moreira, J.: 'FPGA implementation of a Euclidean distance metric SISO decoder'. Proc. 10th Int. Symp. Communications Theory and Applications (ISCTA'09), Ambleside, UK, 2009

20 Terés, L., Torroja, Y., Olcoz, S., Villar, E.: 'VHDL. Lenguaje Estándar de Diseño electrónico' (McGraw-Hill/Interamericana de España, Madrid, 1997)

21 Quartus II Software. Available at: http://www.altera.com

22 DE2 Development and Education Board. Available at: http://www.altera.com

11 Appendix

If $C = e^c$, Ae^a and $B = e^b$, then $C = A + B$ can be determined as

$$c = \max(a, b) + \ln(1 + e^{-|a-b|}) \tag{17}$$

For $C = A + B$ or $C = (-1)^z \quad e^c = (-1)^z |C|$ with $|C| = |A - B|$ and $z = 0$ if $A > B$ else $z = 1$, then

$$\begin{aligned} c &= \max(a, b) + f_+(a, b) \\ c &= \max(a, b) - f_-(a, b) \end{aligned} \tag{18}$$

where

$$f_+(a, b) = \ln(1 - e^{-|a-b|}) \tag{19}$$

$$f_-(a, b) = |\ln(1 - e^{-|a-b|})| \tag{20}$$

Logarithmic calculations in (17) and (18) can be solved by using look-up tables $f_+(a, b)$ and $f_-(a, b)$ with entry $|a - b|$.