A synthesizable IP Core for DVB-S2 LDPC Code Decoding

Frank Kienle, Torben Brack, Norbert Wehn
Microelectronic System Design Research Group
University of Kaiserslautern
Erwin-Schrödinger-Straße
67663 Kaiserslautern, Germany
{kienle, brack, wehn}@eit.uni-kl.de

Abstract

The new standard for digital video broadcast DVB-S2 features Low-Density Parity-Check (LDPC) codes as their channel coding scheme. The codes are defined for various code rates with a block size of 64800 which allows a transmission close to the theoretical limits.

The decoding of LDPC is an iterative process. For DVB-S2 about 300000 *messages are processed and reordered in each of the 30 iterations. These huge data processing and storage requirements are a real challenge for the decoder hardware realization, which has to fulfill the specified throughput of* $255 MBit/s$ *for base station applications.*

In this paper we will show, to the best of our knowledge, the first published IP LDPC decoder core for the DVB-S2 standard. We present a synthesizable IP block based on ST Microelectronics $0.13 \mu m$ *CMOS technology.*

1 Introduction

The new DVB-S2 standard [2] features a powerful forward error correction (FEC) system which enables transmission close to the theoretical limit (Shannon limit). This is enabled by using Low-Density Parity-Check (LDPC) codes [3] one of the most powerful codes known today which can even outperform Turbo-Codes [4]. To provide flexibility 11 different code rates ranging from ($R = 1/4$ up to $9/10$) are specified with a codeword length up to 64800 bits. This huge maximum codeword length is the reason for the outstanding communications performance ($\sim$0.7dB to Shannon) of this DVB-S2 LDPC code proposal, so in this paper we only focus on the codeword length of 64800 bits. To yield this performance, the decoder has to iterate 30 times. At each iteration up to 300 000 data are scrambled and calculated. This huge data processing, storage and network/interconnect requirements is a real challenge for the decoder realization.

A LDPC code can be represented by a bipartite graph. For the DVB-S2 code 64800 so called variable nodes (VN) and $64800 * (1 - R)$ check nodes (CN) exist. The connectivity of these two type of nodes is specified in the standard [2]. For decoding the LDPC code messages are exchanged iteratively between this two type of nodes, while the node processing is of low complexity. Within one iteration first the variable nodes are procesed, then the check nodes.

For a fully parallel hardware realization each node is instantiated and the connections between them are hardwired. This was shown in [5] for a 1024 bit LDPC code. But even for this relatively short block length severe routing congestion problems exist. Therefore a partly parallel architecture becomes mandatory for larger block length, where only a subset of nodes are instantiated. A network has to provide the required connectivity between VN and CN nodes. But realizing any permutation pattern is very costly in terms of area, delay and power.

To avoid this problem a decoder first design approach was presented in [6]. First an architecture is specified and afterwards a code is designed which fits this architecture. This approach is only suitable for regular LDPC code where each VN has the same number of incident edges, the CN respectively. But for an improved communications performance so called irregular LDPC codes are mandatory [7], where the VN nodes are of varying degrees. This is the case for the DVB-S2 code. In [8] we have presented a design method for irregular LDPC codes which can be efficiently processed by the decoder hardware. We used so called irregular repeat accumulate (IRA) codes [9] which are within the class of LDPC codes with the advantage of a very simple (linear) encoding complexity. In general, LDPC code encoder are very difficult to implement due to the inherent complex encoding scheme.

The LDPC codes as defined in the DVB-S2 standard are IRA codes, thus the encoder realization is straight forward. Furthermore, the DVB-S2 code shows regularities which can be exploited for an efficient hardware realization.

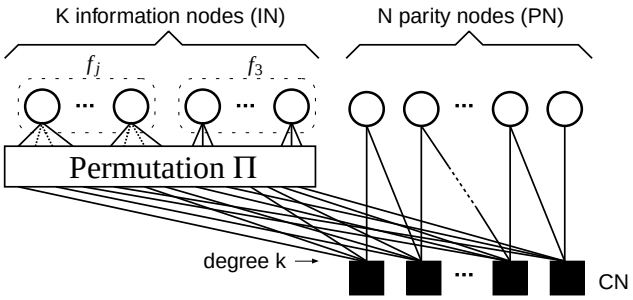

Figure 1. Tanner graph for the DVB-S2 LDPC code

These regularities are also the base for our methodology introduced in [8].

In this paper we show how to exploit these regularities and present an efficient mapping of VN and CN nodes to hardware instances. Memory area and access conflicts are most critical in this mapping process. Thus we used simulated annealing to minimize memory requirements and avoidance of RAM access conflicts.

We present to the best of our knowledge the first IP core capable to process all specified code rates in the DVB-S2 standard. We show synthesis results using a $0.13\mu m$ technology.

The paper is structured as follows: the DVB-S2 LDPC codes and the decoding algorithm are presented in Section 2. In Section 3 the mapping of nodes to hardware instances is explained. The overall decoder architecture is shown in Section 4. Section 5 gives synthesis results and Section 6 concludes the paper.

2 DVB-S2 LDPC Codes

LDPC codes are linear block codes defined by a sparse binary matrix (parity check matrix) H. The set of valid codewords $x \in C$ have to satisfy

$$Hx^T = 0, \qquad \forall x \in C. \tag{1}$$

A column in H is associated to a bit of the codeword and each row corresponds to a parity check. A nonzero element in a row means that the corresponding bit contributes to this parity check. The code can best be described by a Tanner graph [7], a graphical representation of the associations between code bits and parity checks. Code bits are shown as variable nodes (circles), and parity checks as check nodes (squares), with edges connecting them. The number of edges on each node is called the node degree. If the node degree is identical for all variable nodes, the corresponding parity check matrix is called regular, otherwise it's irregular.

By carefully inspection of the construction rules, the DVB-S2 parity check matrix consists of two distinctive parts: a random part dedicated to the systematic information, and a fixed part that belongs to the parity information. The Tanner graph for DVB-S2 is shown in Figure 1. There exist two types of variable nodes, the information (IN) and parity nodes (PN), corresponding to the systematic and parity bits respectively. The permutation Π represents the random matrix part of the connectivity between IN and CN nodes, while the PN nodes are all of degree two and are connected in a fixed zigzag pattern to the CN nodes. The N check nodes have a constant degree k. The K information nodes consist of two subsets f_j and f_3, with f the number of IN nodes of degree j and 3. Table 1 summarizes the code rate dependent parameters as defined in the standard [2].

Rate	j	f_j	f_3	k	N	K
1/4	12	5400	10800	4	49600	16200
1/3	12	7200	14400	5	43200	21600
2/5	12	8640	17280	6	38880	25920
1/2	8	12960	19440	7	32400	32400
3/5	12	12960	25920	11	25920	38880
2/3	13	4320	38880	10	21600	43200
3/4	12	5400	43200	14	16200	48600
4/5	11	6480	45360	18	12960	51840
5/6	13	5400	48600	22	10800	54000
8/9	4	7200	50400	27	7200	57600
9/10	4	6480	51840	30	6480	58320

Table 1. Parameters describing the DVB-S2 LDPC Tanner graph for different coderates

The connectivity of the IN and CN nodes is defined by the DVB-S2 encoding rule

$$p_j = p_j \oplus i_m, \qquad j = (x + q(m \bmod 360)) \bmod N. \tag{2}$$

p_j is the jth parity bit, i_m the mth information code bit, and x, q, and N are code rate dependent parameters specified by the DVB-S2 standard. Equation 2 determines the entries of the parity check matrix. The mth column has nonzero elements in each row j, thus the permutation Π generates one edge between every CN m and IN j.

The fixed zigzag connectivity of the PN and CN nodes is defined by the encoding scheme:

$$p_j = p_j \oplus p_{j-1}, \qquad j = 1, 2, ..., N-1. \tag{3}$$

This is a simple accumulator. The corresponding part of the parity check matrix has two nonzero elements in each column, forming a square banded matrix. This type of LDPC codes with this simple encoding procedure are also called irregular repeat accumulate (IRA) codes [9].

2.1 Decoding Algorithm

LDPC codes can be decoded using the message passing algorithm [3]. It exchanges soft-information iteratively

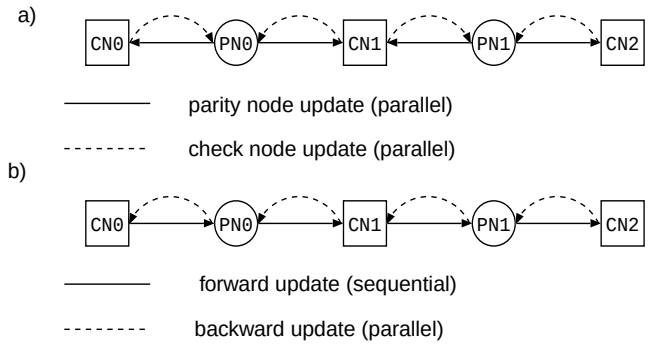

Figure 2. a) conventional message update scheme b) optimized message update scheme

between the variable and check nodes. The update of the nodes can be done with a canonical scheduling [3]: In the first step all variable nodes must be updated, in the second step all check nodes respectively. The processing of individual nodes within one step is independent, and can thus be parallelized.

The exchanged messages are assumed to be log-likelihood ratios (LLR). Each variable node of degree i calculates an update of message k according to:

$$\lambda_k = \lambda_{ch} + \sum_{l=0, l\neq k}^{i-1} \lambda_l, \tag{4}$$

with λ_{ch} the corresponding channel LLR of the VN and λ_i the LLRs of the incident edges. The check node LLR updates are calculated according to

$$\tanh(\lambda_k/2) = \prod_{l=0, l\neq k}^{i-1} \tanh(\lambda_l/2). \tag{5}$$

For fixed-point implementations it was shown in [10] that the total quantization loss is ≤ 0.1db when using a 6 bit message quantization compared to infinite precision. For a 5 bit message quantization the loss is 0.1-0.2 dB [7].

2.2 Optimized update of degree 2 Parity Nodes

The DVB standard supports LDPC codes ranging from code rate $R = 1/4$ to $R = 9/10$. Each code has one common property: the connectivity of the check nodes caused by the accumulator of the encoder. CN_0 is alway connected to CN_1 by a variable node of degree 2 and so on for all CN nodes. A variable node of degree 2 has the property that the input of the first incident edge is the output of the second incident edge (plus the received channel value) and vice versa. For a sequential processing of the check nodes (e.g. from left to right in Figure 1) an already updated message can directly passed to the next check node due to the simple zigzag connectivity. This immediate message update changes the conventional update scheme between CN an VN nodes (Equation 4).

Rate	q	E_{PN}	E_{IN}	Addr
1/4	135	97199	97200	270
1/3	120	86399	129600	360
2/5	108	77759	155520	432
1/2	90	64799	162000	450
3/5	72	51839	233280	648
2/3	60	43199	172800	480
3/4	45	32399	194400	540
4/5	36	25919	207360	576
5/6	30	21599	216000	600
8/9	20	14399	180000	500
9/10	18	12959	181440	504

Table 2. Code rate dependent parameters, with E the number of incident edges of IN and PN nodes and Addr the number of values required to store the code stucture

The difference in the update scheme is presented in Figure 2. Only the connectivity between check nodes and parity nodes is depicted, the incident edges from the information nodes are omitted. Figure 2a) shows the standard belief propagation with the two phase update. In the first phase all messages from the PN to CN nodes are updated, in the second phase the messages from CN to PN nodes respectively. The message update within one phase is commutative and can be fully parallized. Figure 2b) shows our new message update scheme in which the new CN message is directly passed to the next CN node. This data flow is denoted as a forward update and corresponds to a sequential message update. The backwards update from the PN to CN nodes is again a parallel update. Note that a sequential backwards update would result in a maximum a posteriori (MAP) algorithm.

This new update scheme improves the communications performance. For the same communications performance 10 iterations can be saved i.e. 30 iterations instead of 40 have to be used. Furthermore we need to store only one message instead of two messages for the next iteration, which is explained in more detail in Section 4.

3 Hardware mapping

As already mentioned only partly parallel architectures are feasible. Hence only a subset P of the nodes are instantiated. The variable and check nodes have to be mapped on these P functional units. All messages have to be stored during the iterative process, while taking care of RAM access conflicts. Furthermore we need a permutation networks which provides the connectivity of the Tanner graph.

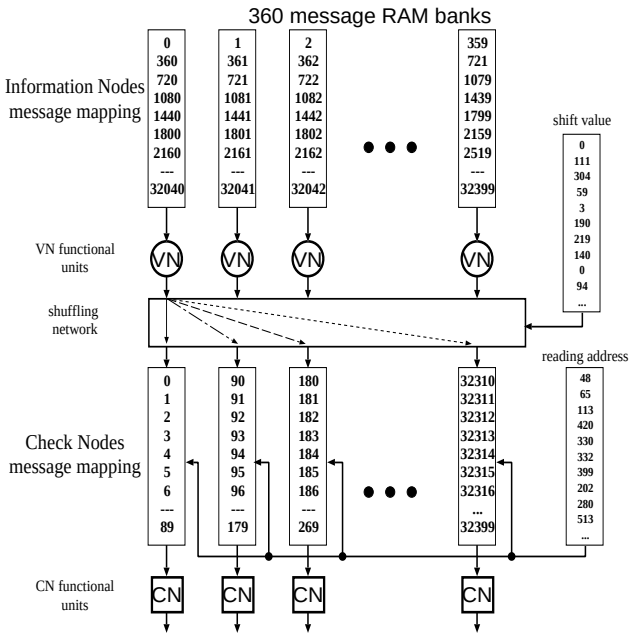

Figure 3. Message and functional unit mapping for $R = 1/2$

We can split the set of edges E connecting the check nodes in two subsets E_{IN} and E_{PN}, indicating the connections between CN/IN nodes and CN/PN nodes respectively. The subsets are shown in Table 2 for each code rate. Furthermore the q factor of Equation 2 is listed. The implementation of E_{IN} is the challenging part, since this connectivity (Π) changes for each code rate. The realization of E_{PN} is straightforward, thus we focus on the mapping of the IN and CN nodes.

Due to the varying node degrees the functional nodes process all incoming messages in a serial manner. Thus a functional node can except one message per clock cycle and produces at most one updated message per clock cycle.

A careful analysis of Equation 2 shows that the connectivity of 360 edges of distinct information nodes are determined by just one value x, while q is a code rate dependent constant, see Table 2.

These 360 edges can be processed simultaneously by $P = 360$ functional units. Within a half iteration a functional unit has to process $q * (k-2)$ edges. $(k-2)$ is the number of edges between one check node and information nodes. For each code rate q was chosen to satisfy the constraint

$$E_{IN}/360 = q * (k-2). \quad (6)$$

It guarantees that each functional unit has to process the same amount of nodes which simplifies the node mapping. Figure 3 shows the mapping of the IN and CN nodes for the LDPC code of rate $R = 1/2$. Always 360 consecutive VN nodes are mapped to 360 functional units. To each functional unit a RAM is associated to hold the corresponding messages (edges). Please note that for each IN of degree 8, 8 storage places are allocated to this VN, because each incident edge has to be stored.

The check nodes mapping depends on the rate dependent factor q. For $R = 1/2$ the first $q = 90$ CN nodes are mapped to the first functional unit. The next 90 CN nodes are mapped to the next producer and so on. Again the CN number corresponds to CN degree storage locations.

This node mapping is the key for an efficient hardware realization, since it enables to use a simple shuffling network to provide the connectivity of the Tanner graph. The shuffling network ensures that at each cycle 360 input messages are shuffled to 360 distinct target memories. Thus we have to store $E_{IN}/390 = 450$ shuffling and addressing information for the $R = 1/2$ code, see Table 2 for the other code rates. The shuffling offsets and addresses can be extracted from the x tables provided by [2].

4 Decoder Architecture

Based on the message mapping described in the previous chapter, the basic architecture of the DVB-S2 LDPC decoder is shown in Figure 4. It consists of functional units which can process the functionality of variable and check nodes. This is possible, since only one type of the node are processed during one half iteration. The IN message memories banks hold the messages which are exchanged between information and check nodes. Furthermore we have memories for storing the exchanged messages for the parity nodes (PN message memories), which are all of degree two. The address and shuffling RAM together with the shuffling network provides the connectivity of the Tanner graph.

As mentioned the decoder processes 360 nodes in parallel so 360 messages have to be provided per cycle. All 360 messages are read from the same address from the IN message memory bank. Though, for the information node processing we just increment the reading address. The functional unit can accept each clock cycle new data, while a control flag just labels the last message belonging to a node and starts the output processing. The newly produced 360 messages are then written back to the same address location but with a cyclic shift, caused by the shuffling network. To process the check nodes we have to read from dedicated addresses, provided by the address RAM. These addresses were extracted from node mapping as described in the previous chapter. Again 360 messages are read per clock cycle and written back to the same address after the processing via the shuffling network. This ensures that the messages are shuffled back to their original position.

The processing of the parity nodes can be done concurrently during the check node processing, by using the update scheme described in Section 2.2. Each functional

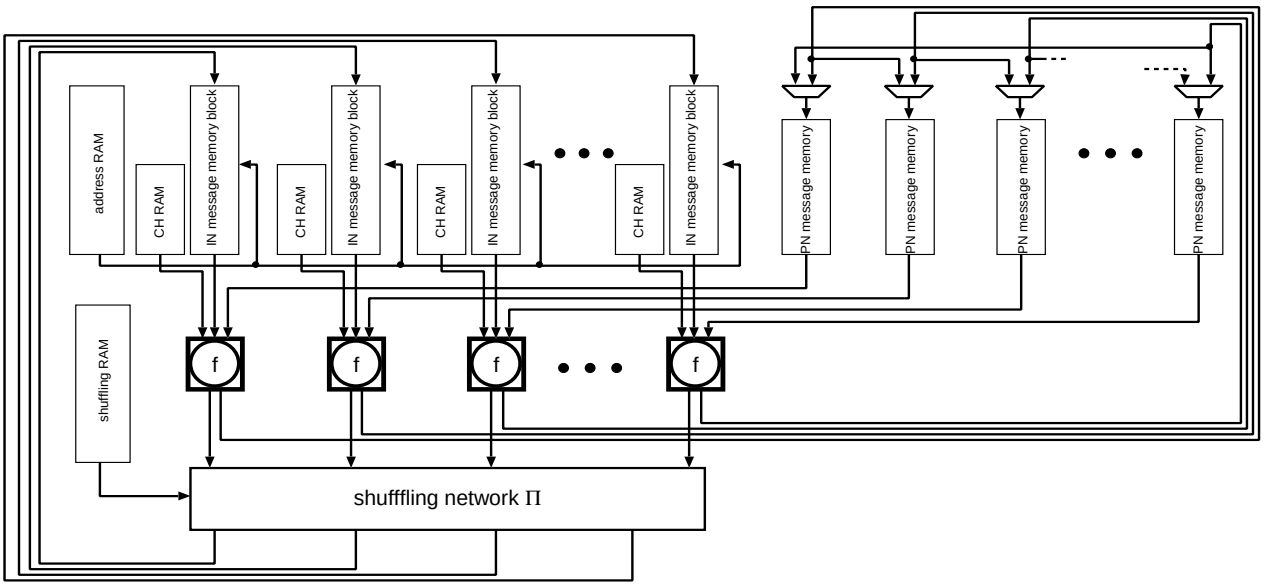

Figure 4. Basic architecture of our LDPC decoder

unit processes consecutive check nodes (Figure 3). The message which is passed during the forward update of the check nodes is kept in the functional unit. Only the messages of the backward update has to be stored which reduces the memory requirements for the zigzag connectivity to $E_{PN}/2$ messages. The PN message memories are only read and written during the check node phase, while the channel (CH) RAMs delivers the corresponding received channel value.

We use single port SRAMs due to area and power efficiency. Hence we have to take care of read/write conflicts during the iterative process. Read/write conflicts occur, since data are continously read from the 360 RAMs and provided to the functional units, while new processed messages have to be written back.

The check node processing is the most critical part. We have to read from dedicated addresses extracted during the mapping process. Therefore, the IN message memory block is partitioned in 4 RAMs which is shown in Figure 5. Even if the commutativity of the message processing within a check node is exploited all write conflicts can not be avoided. Therefore a buffer is required to hold a message if writing is not possible due to a conflict. We use simulated annealing to find the best addressing scheme to reduce RAM access conflicts and hence to minimize the buffer overhead. This optimization step ensures that only one buffer is required which holds for all code rates. Per clock cycle we read data from one RAM, and write at most 2 data back to two distinct RAMs, coming from the buffers or the shuffling network. The two least significant bits of the addresses determines the assignment to a partition. This allows a simple control flow, which just has to compare the reading and the writing addresses of the current clock cycle.

The resulting decoder throughput T is

$$T = \frac{I}{\#\text{cyc}} \cdot f_{\text{cyc}}, \tag{7}$$

with I the number of information bits to be decoded and #cyc the number of cycles to decode one block including the input/output (I/O) processing.

The number of cycles is calculated as $\frac{C}{P_{IO}} + It \cdot \left(2 \cdot \frac{E_{IN}}{P}\right)$. Thus Equation 7 yields:

$$T = \frac{I}{\frac{C}{P_{IO}} + It \cdot \left(2 \cdot (\frac{E_{IN}}{P} + T_{latency})\right)} \cdot f_{\text{cycle}}. \tag{8}$$

The part $\frac{C}{P_{IO}}$ is the number of cycles for input/output (I/O) processing. The decoder is capable to receive 10 channel values per clock cycle. Reading a new codeword of size C and writing the result of the prior processed block can be done in parallel with reading/writing P_{IO} data concurrently. The latency $T_{latency}$ for each iteration depends on the processing units and the shuffling network.

5 Results

The LDPC decoder is implemented as a synthesizable VHDL model. Results are obtained with the Synopsis Design Compiler based on a ST Microelectronics 0.13μm CMOS technology. The maximum clock frequency is 270 MHz under worst case conditions. The decoder is capable to process all specified code rates of the DVB standard with

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.