“Deep-Submicron Full-Custom VLSI-Design of Highly Optimized High-Throughput Low-Latency LDPC Decoders”

Von der Fakultät für Elektrotechnik und Informationstechnik der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften genehmigte Dissertation

vorgelegt von
Diplom-Ingenieur
Matthias Korb
aus Arnsberg

Berichter: Universitätsprofessor Dr.-Ing. Tobias G. Noll
Universitätsprofessor Dr.-Ing. Norbert Wehn
Tag der mündlichen Prüfung: 18. Januar 2012

This thesis results from my work at the Institute of Electrical Engineering and Computer Systems at the RWTH Aachen University which was partially supported by the German Research Foundation (DFG) in its priority programme SPP1202.

I would like to express my deepest gratitude to Prof. Dr.-Ing. Tobias G. Noll for giving me the opportunity to work in various interesting projects at the Institute of Electrical Engineering and Computer Systems. I owe him many thanks for innumerable, inspiring discussions.
Secondly, I would like to thank Prof. Dr.-Ing. Norbert Wehn for his commitment and dedication with regard to my coreferat.
I am very grateful to the colleagues at the Institute of Electrical Engineering and Computer Systems for all the constructive discussions in the recent years.
Many thanks to my mother Edeltraud Korb and Ulf Preuß for their continuous encouragement and support during my studies. Special thanks to Ulf Preuß who acquainted me with the field of Electrical Engineering in the first place.
Finally, I would like to thank my wife Silvia, for everything.

Aachen, April 2012 Matthias Korb

Table of Contents

1 Motivation

The rapidly growing demand in data communication results in an increasing throughput requirement of communication channels. As the bandwidth is typically limited the system has to assure a communication very close to the theoretical Shannon limit to meet these throughout specifications. In 1962 Gallager already presented a channel decoding algorithm [1] which nearly achieves this theoretical limit [2] and allows for the best decoding performance to this day. However, the complexity of this algorithm impeded an integration of such decoders for a long time. Therefore, in the past Viterbi [3] or Turbo decoders [4] have been adopted to various standards.

The first integrated LDPC decoder has been published 2002 by Blanksby and Howland [5]. From then on LDPC codes have been adopted to a wide range of communication standards beginning from wire-bound communication systems like the IEEE 802.3an standard [6] and the 1 GBit over powerline standard [7] to wireless communication systems like WiFi [8], WiMax [9], DVB-S [10], and DTTB [11]. Recently LDPC codes have also been integrated in read-write-channels of hard-disk drives [12].

The outstanding error correction performance of LDPC codes is due to an extensive exchange of messages between the two building blocks of the receiver sided decoder. The communication between these blocks highly affects the decoder features. The silicon area of the decoder in [5] e.g. is highly increased to realize the complex interconnect between the two block types leading to a utilization of the active silicon area of just 50 %. Since 2002 various optimization techniques to reduce the impact of the interconnect have been proposed. However, the benchmarking at the end of this work shows that for high-throughput applications this reduction always consists of a trade-off between silicon area and throughput and / or energy.

For a simultaneous reduction of all decoder features an optimization on all design levels ranging from system level down to physical implementation level as illustrated in Fig. 1-1 is mandatory. Thereby, on each design level the impact of different implementation options on the resulting decoder features have to be estimated as accurate as possible to minimize the probability of wrong decisions. For such a quantitative analysis accurate area-, timing-, and energy-cost models are required. For less complex logic structures like e.g. finite-impulse-response filters such generic cost models can be derived easily because they base on a simple gate count. However, for LDPC decoders the influence of the global interconnect complicates the derivation of general cost models. This might be the reason why no accurate cost models

are known from literature so far. Furthermore, the absence of such models impedes a fair comparison of different decoder implementations as highly simplified scaling rules are used.

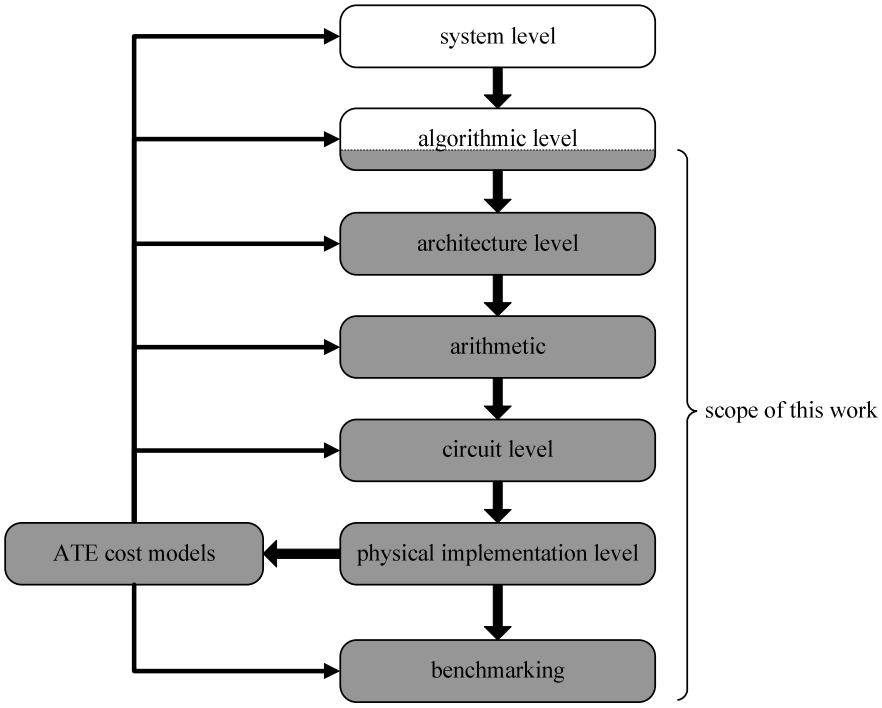


Fig. 1-1 Cost-model supported design process

In this work it will be shown how a careful and systematic optimization on all design levels allows for a significant reduction in hardware complexity in comparison to known decoder implementations. Thereby, the reduction in silicon area is not achieved at the expense of an increased energy or a decreased throughput as the decoder minimizes all three decoder metrics simultaneously. As a proof of concept a decoder compliant to the IEEE 802.3an standard is designed, as this is the application with the highest throughput requirement for LDPC decoders today.

The structure of the work is as follows:

Chapter 2 briefly introduces LDPC codes and state-of-the-art decoder architectures.

In chapter 3 general, quantitative cost models of high-throughput LDPC decoders for the three basic decoder metrics silicon area, iteration period, and energy per iteration are derived.

These models support the following design flow which starts with a quantitative analysis of fixed-point implementations of the decoding algorithm in chapter 4. Thereby, hardware-efficient realizations of the transcendent functions in the Sum-Product algorithm and hardware-efficient post-processing factors for the Min-Sum algorithm are analyzed.

Optimizations on architecture level are presented in chapter 5. In the first part of the chapter a new bit-parallel decoder architecture is discussed which shows a reduced

interconnect impact on the silicon area. Furthermore, a systematic analysis of bit- and check-node based architectures reveals a new partially bit-serial decoder architecture which overcomes the drawbacks of today's known architectures. This architecture is further optimized on arithmetic level before ATE-cost models are derived which underline the efficiency of this architecture.

In chapter 6 the optimizations performed on algorithm, architectural, and arithmetic level are combined. The resulting decoder is realized in a deep-submicron CMOS technology using a full-custom design flow for the LDPC code adopted in the IEEE 802.3an standard. Finally, the decoder features are compared to other decoder implementations in a benchmarking. Thereby, scaling rules have been derived based on the decoder cost-models of chapter 3 to allow for a fair comparison.

Chapter 7 summarizes and concludes this work.

2 Introduction

Every communication channel, whether wired or wireless, is imperfect. Due to the influences of other communication channels or other electromagnet noise sources the sent and received information differ in general. To reduce the differences to a tolerable level channel decoders are introduced into the transmission system. The underlying idea of channel decoders is to add additional information called redundancy in the transmitter-sided channel encoder. This redundancy can then be used in the receiver-sided channel decoder to cancel out possible transmission errors.

2.1 Channel decoders

One major class of channel codes contains the so called block codes. For these codes the information symbols generated by an information source are combined in blocks with a fixed block length *m* as assumed in the simplified channel model in Fig. 2-1.

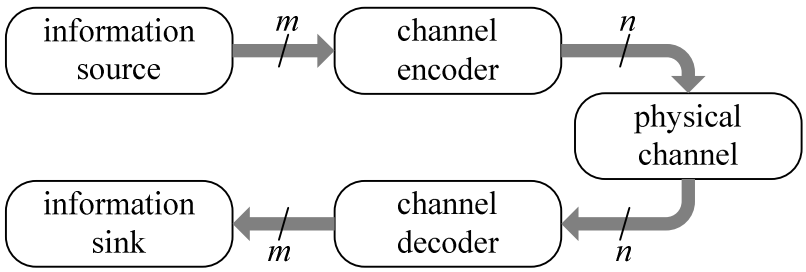


Fig. 2-1 Simplified model of communication channel with block code

In the channel encoder *n-m* redundant symbols are added to these *m* symbols leading to a code blocks length of *n*. The more redundancy is added the higher is the probability that communication errors can be corrected. However, if the bandwidth of the physical channel is limited, an increasing number of redundancy symbols would reduce the effective bandwidth which can be used for the information transfer. A metric for the error correction capability of the code is the code rate

$$\boldsymbol{R} = \frac{\boldsymbol{m}}{\boldsymbol{n}} . \tag{2-1}$$

A low code rate is an indicator of a large number of redundant symbols and, thus, a high error correction potential.

For block codes the code word *y* of block length *n* can be generated by multiplying the information word *x* of block length *m* with a generator matrix *G* as follows

$$\boldsymbol{y} = \boldsymbol{x} \cdot \boldsymbol{G} . \tag{2-2}$$

In the physical communication channel this code word gets distorted. Considering for example additive white Gaussian noise (awgn) the received code word would be

$$\boldsymbol{y}' = \boldsymbol{y} + \boldsymbol{wgn} \quad (2\text{-}3)$$

Based on the introduced redundancy communication errors can be detected in the receiver sided channel decoder. Therefore, the received code word is multiplied with a parity-check matrix *H*. As the product of the generator matrix and the parity-check matrix is

$$\boldsymbol{G} \cdot \boldsymbol{H}' = \boldsymbol{0} \quad (2\text{-}4)$$

the multiplication of an error-free code word with the parity-check matrix would also be equal to the zero vector:

$$\boldsymbol{y}' \cdot \boldsymbol{H}' = \boldsymbol{x} \cdot \boldsymbol{G} \cdot \boldsymbol{H}' = \boldsymbol{0} \quad (2\text{-}5)$$

Thereby, each of the *m* rows of *H* represents one parity check. The location of the one entries in that row defines which symbols of the received code word are taking part in that particular parity check. If the number of one entries per row and per column is constant, and, therefore, each parity check bases on the same number of symbols and each symbol takes part in the same number of parity checks, the matrix is called regular. Otherwise, it is called irregular.

If the received code word is not error-free, the information which is generated during the processing of the *m* parity checks can be used to correct the occurred transmission errors. This basic principle is e.g. used in Turbo Codes introduced by Berrou, Glavieux and Thitimajshima in 1993 [4]. The basic principle of a Turbo decoder is shown on the left side of Fig. 2-2. For each of the *n* received noisy symbols a channel information $L(c_i)$ is calculated which indicates the probability of a sent ‘1’ or ‘0’. Thereby, the sign is an estimation of the sent symbol and the magnitude is an indicator for the reliability of the sign statement with a high magnitude representing a high reliability.

In a Turbo decoder the information of all received symbols are fed into a soft-input-soft-output (SISO) decoder which performs the parity checks defined by the matrix *H* and calculates additional A-posteriori information $L(r_i)$ for each symbol *i*. These new information are summed up with the channel information leading to a new estimation of the sent symbol $L(Q_i)$.

Although the LDPC decoding algorithm was already introduced by Gallager in 1962 [1], it can be seen as a modification or improvement of this Turbo principle. In contrast to the basic Turbo decoder of Fig. 2-2, there are *m* SISO decoders in an LDPC decoder. Each of these *m* decoders perform one of the *m* parity checks defined in the parity-check matrix and calculates new A-posteriori information $L(r_{i,j})$ for each participating symbol *i*. The A-posteriori

information for one symbol is combined with the channel information to derive a new estimate of the received symbol. In the next iteration this sum is fed back to the SISO decoders.

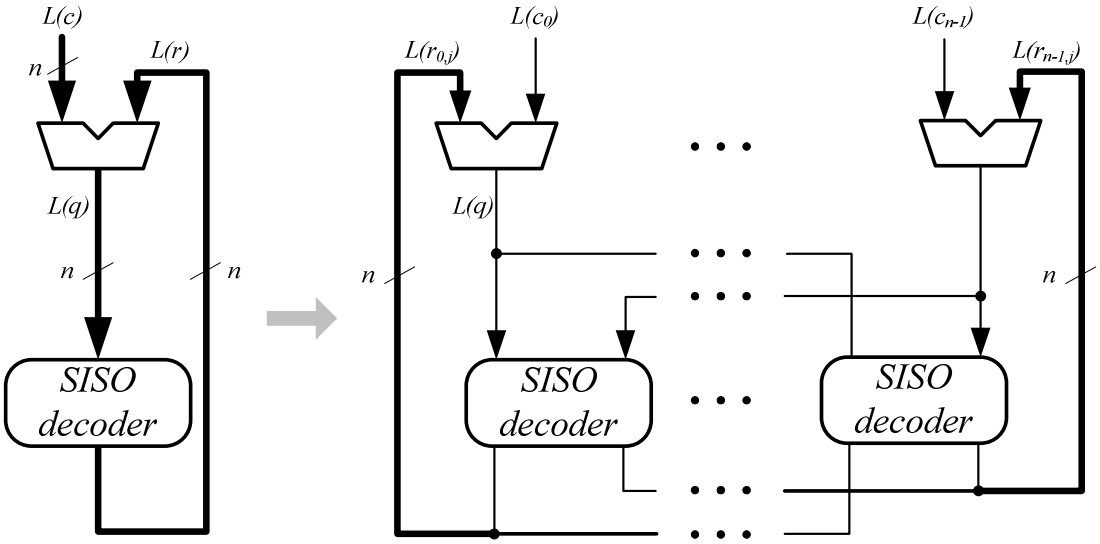


Fig. 2-2 Turbo- and LDPC- decoding principle

In LDPC decoders the single SISO decoders are called check nodes. The subsequent combination of the A-priori and A-posteriori information is done in the so called bit nodes leading to the so called Tanner Graph. The Tanner Graph for a simplified parity-check matrix H is depicted in Fig. 2-3. The number of connections of each check node (bit node) corresponds to the number of one entries in each row (column) of the parity-check matrix H and is called the degree of the check node d_C (bit node d_V).

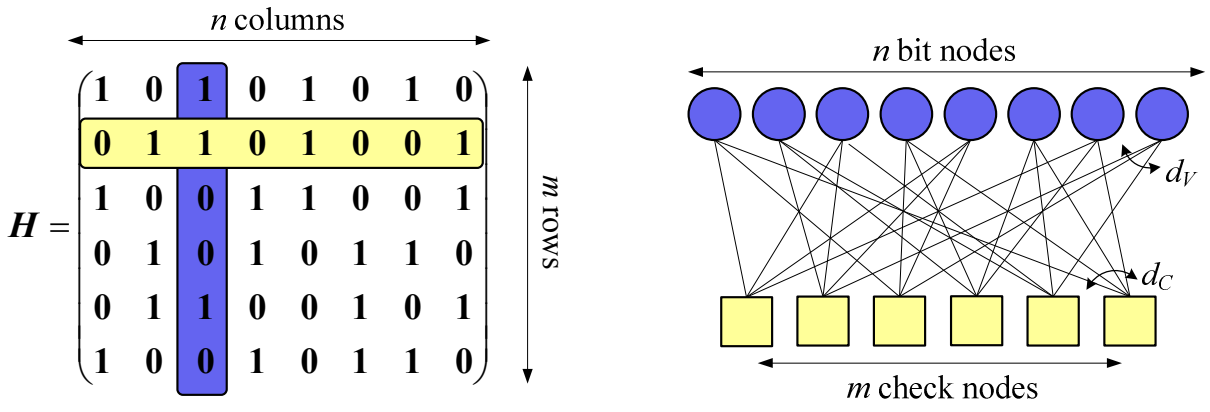


Fig. 2-3 Tanner Graph

2.2 LDPC decoding algorithm

In Fig. 2-4 a decoder loop consisting of one bit and one check node is depicted. For the sake of clarity all the connections to other nodes are not drawn. At the beginning of the

decoding process the channel values are initialized. Considering e.g. a BPSK modulation scheme the initialization would be done as follows

$$L(c_i) = \frac{2}{\sigma^2} \cdot y'_i \,. \tag{2-6}$$

Again, the sign of $L(c_i)$ indicates whether the current estimate of the received symbol i is a '-1' or a '+1' or rather a '0' or a '1', respectively.

The iterative decoding starts subsequent to the initialization. As there is no A-posteriori information at this time the bit-node message $L(q_{i,j})$ sent from bit node i to check node j just contains the channel information. In each check node a parity check is performed which bases on the bit-node messages received from the d_C connected bit nodes. Therefore, the signs of these messages are combined in a multi-operand XOR-gate. If the resulting signal is '0', the considered d_C messages have an even parity. In such a case the reliability of the estimation should be strengthen for all participating symbols. Thus, the A-posteriori information for a bit-node message with a positive (negative) sign should be positive (negative). In case of an uneven parity the reliability should be weakened. This time the result of the multi-operand XOR-gate is '1' and the A-posteriori information for a bit-node message with a positive (negative) sign should be negative (positive). This mapping can be realized by a controlled inverter which is an XOR-gate in hardware. As long as the parity is even the XOR-gate transfers the bit-node message sign to the output which strengthens the reliability. In case of an uneven parity the bit-node message sign is inverted.

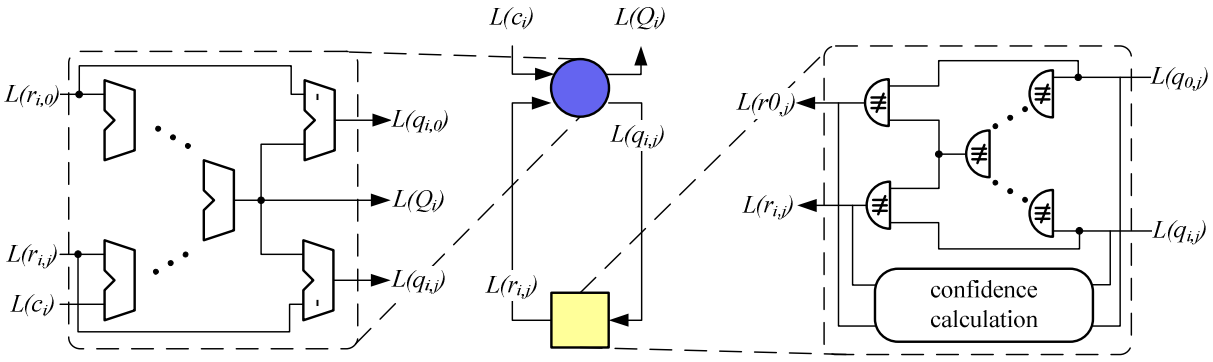


Fig. 2-4 LDPC-decoder loop

A separate logic calculates the reliability in these new check-node message signs. The underlying idea is that if all bit-node messages have a high reliability, the reliability in the check-node message signs is high, as well. However, if the reliability of only one of the input signs is low, it degrades the reliability in the check-node message signs significantly.

The d_C check-node messages $L(r_{i,j})$ are sent back to the connected bit nodes. In the bit nodes the channel information and the check-node messages received from the d_V connected parity checks are combined using a multi-operand adder. The sign of the resulting $L(Q_i)$ value

$$L(Q_i) = L(c_i) + \sum_{j',H(j',i)=1} L(r_{i,j'}) \tag{2-7}$$

is the new estimation for the received symbol. The direct influence of the A-posteriori information of one check node on the same check node in the next iteration would result in a degradation of the decoding performance. Before the A-priori values are sent to the check node the influence is eliminated by subtracting $L(r_{i,j})$ leading to the new bit-node message

$$L(q_{i,j}) = L(Q_i) - L(r_{i,j}). \tag{2-8}$$

As the decoding is performed in two steps, namely the parallel update of all check nodes followed by a parallel calculation of all new A-priori information in the bit nodes, this algorithm is also named two-phase message passing (TPMP).

There are multiple ways to calculate the reliability of the check-node messages based on the reliability of the bit-node messages. In the original Sum-Product decoding algorithm the reliability in the A-posteriori information signs is calculated as

$$\left| L(r_{i,j}) \right| = 2 \cdot atanh\left(\prod_{i',H(j,i')=1,i'\neq i} tanh\left(\frac{\left| L(q_{i',j}) \right|}{2} \right) \right). \tag{2-9}$$

As a realization of the multiplication would result in a high hardware complexity the calculation can be performed in the logarithmic domain leading to

$$\left| L(r_{i,j}) \right| = \Phi^{-1}\left(\sum_{i',H(j,i')=1,i'\neq i} \Phi\left(\left| L(q_{i',j}) \right| \right) \right) \tag{2-10}$$

with

$$\Phi(x) = log\left(tanh\left(\left[\frac{x}{2} \right] \right) \right). \tag{2-11}$$

For each of the d_C check-node outputs (2-10) has to be calculated. Instead of calculating d_C different sums with d_C-*1* operands it is possible to calculate the sum of all inputs initially. Afterwards for each output only one operand has to be subtracted from this sum as follows

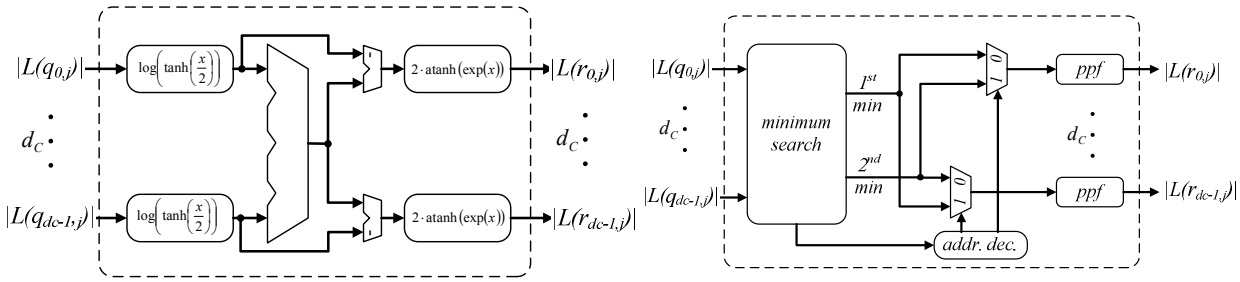
$$\left| L(r_{i,j}) \right| = \Phi^{-1}\left(\sum_{i',H(j,i')=1} \Phi\left(\left| L(q_{i',j}) \right| \right) - \Phi\left(\left| L(q_{i,j}) \right| \right) \right) = \Phi^{-1}\left(L(R_j) - \Phi\left(\left| L(q_{i,j}) \right| \right) \right). \tag{2-12}$$

A block diagram of the resulting structure is depicted in Fig. 2-5 a). At the input of the reliability calculation each magnitude of the bit-node messages is transformed using (2-11).

Afterwards all the results are added with the subsequent subtraction of one value. Finally, the reliability is calculated by applying Φ^{-1}.

Due to the complex implementation of the transcendent Φ and Φ^{-1} functions in the Sum-Product algorithm, an approximate calculation is proposed by Fossorier et al. in [13]. As the reliability of the A-posteriori information in the original algorithm is mainly determined by the smallest A-priori reliability, the underlying idea is to use the minimum A-priori reliability as follows

$$\left| L(r_{i,j}) \right| = \min_{i',H(j,i')=1,i'\neq i} \left[\left| L(q_{i',j}) \right| \right]. \tag{2-13}$$



a) Sum-Product algorithm **b) Min-Sum algorithm**

Fig. 2-5 Reliability calculation

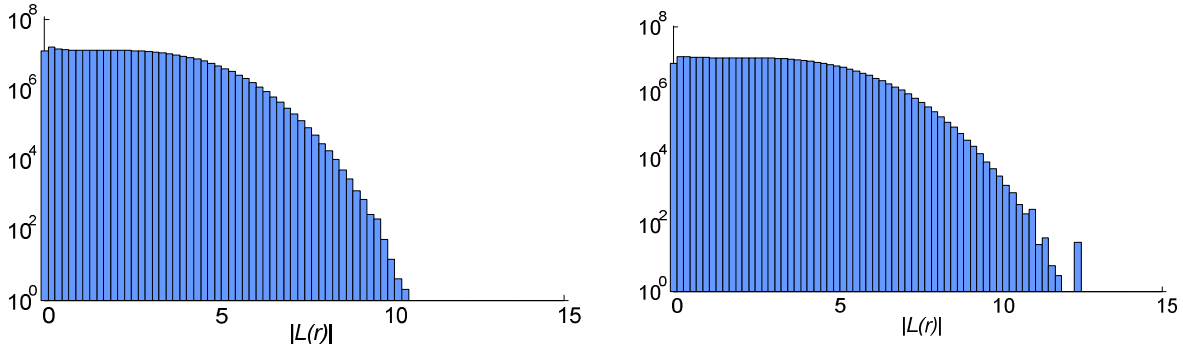
Again, the A-posteriori information which is sent back to bit node i only includes the information of the other d_C-1 A-priori information. Thus, only two different reliability levels are possible at the output of a Min-Sum based check node: the smallest and second smallest absolute $|L(q_{i,j})|$ value. To reduce the hardware complexity it is possible to search for these values and subsequently select one of them for each check-node output. The resulting equation is

$$\left| L(r_{,ij}) \right| = \begin{cases} \min\limits_{i',H(j,i')=1}^{2^{nd}} \left| L(q_{i',j}) \right| & \text{if } \left| L(q_{i,j}) \right| \text{ is minimum} \\ \min\limits_{i',H(j,i')=1} \left| L(q_{i',j}) \right| & \text{else} \end{cases}. \tag{2-14}$$

Considering $|L(q_{i',j})|$ is the minimum among all the check-node inputs the second minimum is assigned to the output $|L(r_{i',j})|$. $|L(q_{i',j})|$ is assigned to all other outputs. A block diagram of the reliability calculation basing on this equation is depicted in Fig. 2-5 b).

In general the approximation leads to larger absolute A-posteriori values as shown in Fig. 2-6. Both histograms of the A-posteriori information show a similar characteristic, but in contrast to the Sum-Product decoder for the Min-Sum based decoder there are magnitudes up to 12.5. As the increased values negatively affect the decoding performance, a reduction of the Min-Sum A-posteriori information is proposed in [14]. Therein, before sending the A-

posteriori information to the connected bit nodes the magnitude is reduced by a post-processing function as it is also shown in Fig. 2-5 b).



a) Sum-Product algorithm **b) Min-Sum algorithm**

Fig. 2-6 Histogram of A-posteriori information

2.3 Decoder architectures

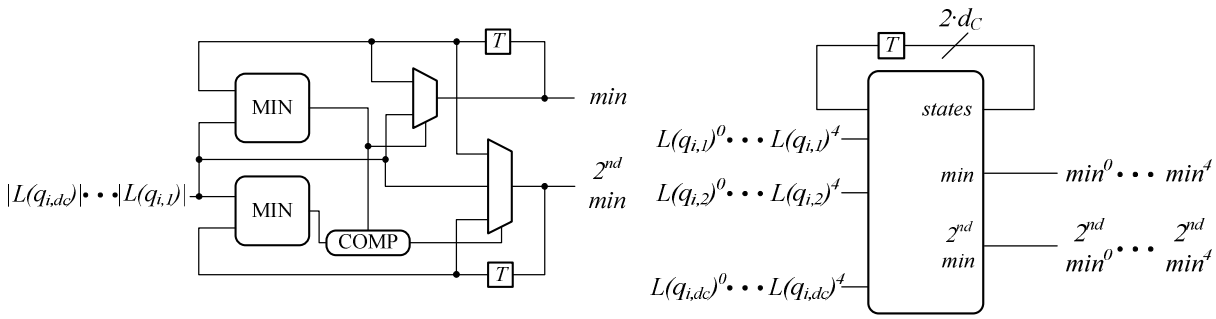
The first ever published integrated CMOS decoder implementation origins from Blanksby and Howland in 2002 [5] and bases on the Sum-Product algorithm. In this decoder all the nodes of the Tanner Graph are instantiated in the VLSI macro in parallel. The global interconnect is hard wired and realized bit-parallel. Additionally, the arithmetic in the nodes is realized bit-parallel using node architectures as shown in Fig. 2-4 and Fig. 2-5 a). As the complex bit-parallel interconnect can not be realized atop of the logic area, the decoder has to be extended artificially leading to a large silicon area. Only about 50 % of the decoder macro is utilized by node logic.

As the hardware effort for such a decoder is very high even in today's CMOS technologies, various trade-offs have been proposed in the last years. In the majority of cases hardware complexity is traded-off with energy and especially with decoder throughput. However, there are some proposals in which a trade-off between hardware complexity and decoding performance is proposed, e.g. [15], [16]. These approaches are not applicable when targeting a system for a certain specification such as the IEEE 802.3an standard [6] as the resulting decoder would not guarantee the required decoding performance. Therefore, such approaches are not investigated in this work.

When considering to trade-off hardware complexity with throughput, there are three levels to introduce multiplexing in time into LDPC decoders: message-, word- or bit-level. In contrast to the fully parallel implementation of [5] it is possible to instantiate only a part of the bit and / or a part of the check nodes like it is e.g. done in [17]. Therefore, the bit and / or check-node messages are calculated serial.

Another possibility is the serialization on word-level as e.g. proposed in [18]. Fig. 2-7 a) exemplarily shows the architecture of a word-serial reliability calculation for the Min-Sum algorithm. The actual input message is compared to the stored minimum and second minimum. If the input value is smaller than the minimum value, the minimum is replaced. In this case the old minimum becomes the second minimum. Otherwise, the input still can become the second minimum which depends on the comparison of the input value with the old second minimum. The reduction in hardware complexity directly becomes obvious when comparing the word-serial with the word-parallel check node of Fig. 2-5 b). While the word-parallel minimum search would require ($2 \cdot d_C - 3$) comparators, only two comparators are required in a word-serial realization.

Finally, an introduction of multiplexing in time on bit-level is also possible. In Fig. 2-7 b) exemplarily a bit-serial search for the minimum and the second minimum value as it is e.g. used in [19] is sketched. In such a minimum search for each of the d_C MSB-first input data streams a state vector of two bits decodes whether the input stream can possibly be the minimum, the second minimum or neither of them. Therefore, in total $2 \cdot d_C$ bits decode the state of the minimum search and needs to be fed back. Based on the state and the actual input bits, the bits of the minimum and second minimum is calculated and the state is updated.



a) Word-serial check node **b) Bit-serial minimum search**

Fig. 2-7 Serialization levels in LDPC decoders

As multiplexing in time can be introduced to more than one level in parallel in total, there are eight different architecture classes leading to the tree depicted in Fig. 2-8. Tab. 2-1 lists various decoder implementations known from literature and assigns each implementation to one of the eight classes of Fig. 2-8.

For some of the decoder classes no implementations are known as some combinations are not suggestive. E.g. a parallel calculation of all check and all bit nodes would conflict with a word-serial input of the nodes. This is the reason for missing implementations in architecture classes 3 and 4.

Tab. 2-1 ASIC LDPC decoder implementations

Author	Year	Architecture	Algorithm	Application	Ref.
Blanksby	2002	1	Sum-Product	n.s.	[5]
Cocco	2004	5	Min-Sum	n.s.	[20]
Lin	2005	5	Min-Sum	n.s.	[17]
Darabiha	2005	n.a.	Hard-Decision	802.3an	[21]
Yeo	2005	7	Min-Sum	n.s.	[22]
Urard	2005	n.s.	Sum-Product	DVB-S2	[23]
Ishikawa	2006	5	Mod. Min-Sum	n.s.	[24]
Mansour	2006	5	Turbo-Decoding	n.s.	[25]
Karkooti	2006	7	Min-Sum	802.11n	[18]
Brack	2006	7	Min-Sum	802.16e	[26]
Kang	2006	5	Sum-Product	n.s.	[27]
Gunnam	2007	5	Min-Sum	802.11n	[28]
Swamy	2007	5	Min-Sum	LDPC-CC	[29]
Sun	2007	5	Min-Sum	n.s.	[30]
Bimberg	2007	7	Min-Sum	LDPC-CC	[31]
Darabiha	2007	2	Approx. Min-Sum	802.3an	[15]
Brandon	2008	2	Min-Sum	n.s.	[19]
Mohsenin	2009	1	Split-Row	802.3an	[16]
Chen	2009	7	Min-Sum	n.s.	[32]
Sha	2009	7	Min-Sum	n.s.	[33]
Zhang	2009	5	Min-Sum	802.3an	[34]
Xiang	2009	7	Min-Sum	DTTB	[35]
Huang	2009	5	Min-Sum	802.16e	[36]
Jiang	2009	5	Min-Sum	DTTB	[37]
Shih	2009	5	Min-Sum	WiMax	[38]

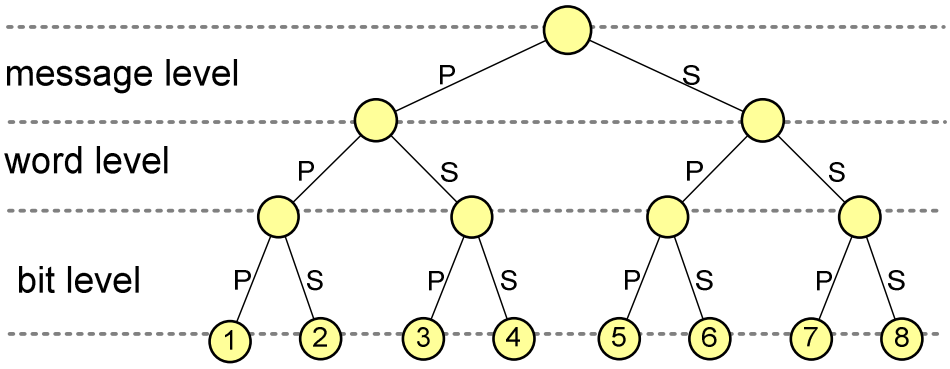


Fig. 2-8 LDPC-decoder architecture classification

As no multiplexing in time is introduced in architecture 1, it allows for the highest decoder throughput. Each additional serialization decreases the decoder throughput. Nevertheless, with the introduction of multiplexing in time on one level high decoder throughputs can be achieved. In [15] e.g. a decoder of architecture class 2 is proposed which achieves a throughput of 5 million blocks per second in a 130-nm CMOS technology. This high throughput is achieved by interleaved decoding of two blocks. In [39] it is shown that with multiplexing in time on message-levels a throughput of even 23 million blocks per second in a 65-nm CMOS technology is possible. Here, the high throughput is achieved by an extensive pipelining scheme.

2.4 Metrics of LDPC decoders

A quantitative comparison of different decoder implementations requires the definition of cost metrics. For digital circuits a common approach is to use a metric for the manufacturing costs (typically the silicon area *A)*, one for the cost during usage (typically the energy per operation *E)* and a timing metric *T*. Here, often the inverse of the throughput is used but for some applications also the latency of the operation is critical and should be considered. Based on these metrics an ATE-complexity can be defined which is the area-timing-energy-product ($A \cdot T \cdot E$). Then, a decoder implementation is superior to another, if its ATE-complexity is lower than that of the other implementation.

When designing LDPC decoders for certain applications there typically are decoding performance specifications in terms of bit- or frame-error rates (BER and FER, respectively). Additionally, typically a certain block throughput and possibly a certain block latency have to be supported.

Considering e.g. the block interleaved decoder in Fig. 2-9 the received block is loaded digit-serial with a word length of w_{IN} into the input shift register. When the load of the block

is completed, the decoding is started. Meanwhile, the second block can be loaded simultaneously.

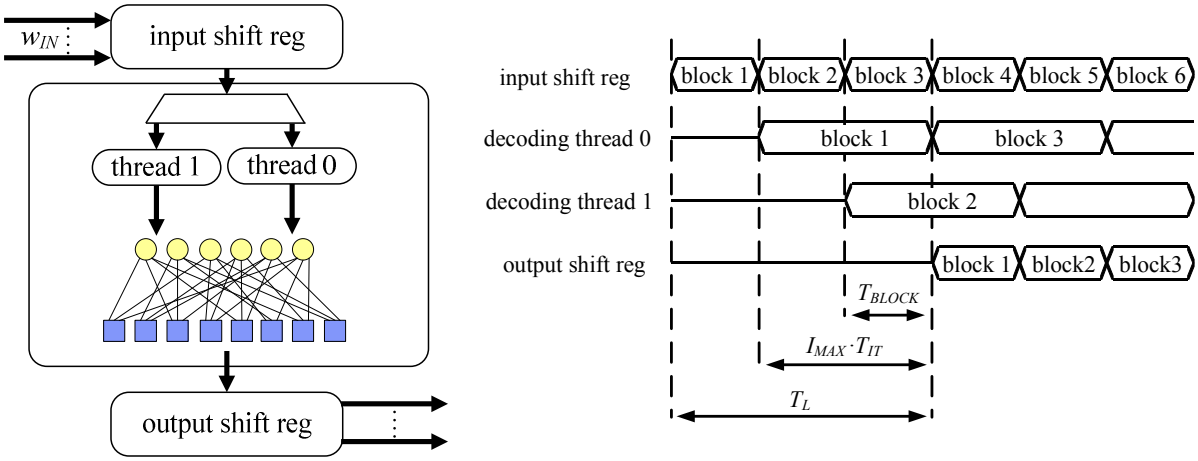


Fig. 2-9 Timing diagram of an interleaved decoder

The timing features of the decoder can be determined when looking at the timing diagram in Fig. 2-9. Assuming a constant iteration period T_{IT} the interleaving of B blocks leads to a reduced block period T_{BLOCK}, which is the inverse of the decoder block throughput. It depends on the iteration period, on the maximal number of iterations I_{MAX} and on the number of interleaved blocks as follows

$$T_{BLOCK} = \frac{1}{f_{BLOCK}} = \frac{I_{MAX} \cdot T_{IT}}{B} . \tag{2-15}$$

The block latency T_L on the other hand consists of the time T_{LOAD} to load the n channel-information of one received code word into the input shift registers and the duration of the decoding process $I_{MAX} \cdot T_{IT}$. Thus, the block latency can be written as

$$T_L = \frac{n \cdot w_{A-PRIORI}}{f_{CLK} \cdot w_{IN}} + I_{MAX} \cdot T_{IT} = T_{LOAD} + B \cdot T_{BLOCK} . \tag{2-16}$$

Therein, $w_{A\text{-}PRIORI}$ defines the word length of the input values. Obviously, both timing metrics linearly depend on the iteration period. The minimization of the iteration period simultaneously increases the decoder throughput and reduces the block latency. Therefore, instead of using two timing metrics, the iteration period can be used as the timing metric T.

Exceeding any timing specifications does not feature any advantages. Therefore, instead of optimizing the ATE-complexity a sole minimization of the manufacturing and usage costs should be performed. However, in the following the ATE-complexity is used as it allows a comparison of decoder implementation targeting different timing specifications. Besides the iteration period, the silicon area of the decoder macro A_{DEC} and the energy per iteration E_{IT} are used and the ATE-complexity of an LDPC decoder is defined as

$$\boldsymbol{ATE-complexity} = \boldsymbol{A_{DEC}} \cdot \boldsymbol{T_{IT}} \cdot \boldsymbol{E_{IT}} . \tag{2-17}$$

In the following, this metric will be used to quantitatively compare different design choices throughout the design process and to compare different decoder implementations in a benchmarking.

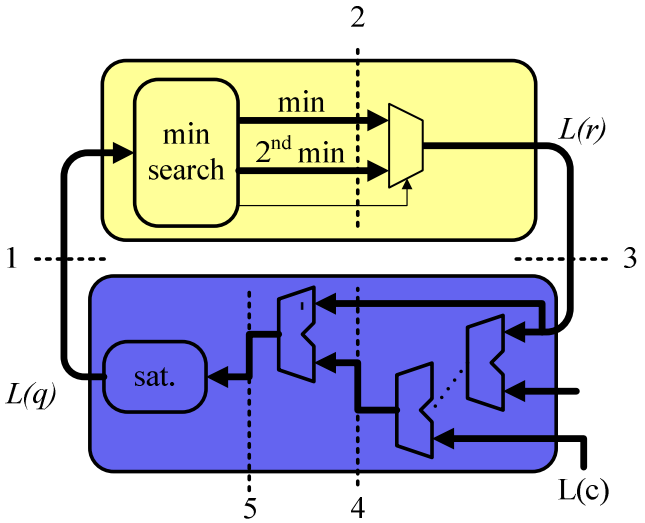
3 Generic ATE-cost models of LDPC decoders

The design of ATE-efficient LDPC decoders requires an optimization and, therefore, a quantitative analysis on all design levels. Thus, accurate ATE-cost models are mandatory, especially in early design phases as these allow for the highest optimization potential. For less complex logic structures like e.g. finite-impulse-response filters the derivation of generic cost models is more simple, as e.g. the circuit features are dominated by the logic leading to a simple gate count. In contrast LDPC decoders are dominated by the exchange of the messages between the two nodes and, therefore, for decoders of architecture classes 1 and 2 (Fig. 2-8) by the complex interconnect.

As the impact of the interconnect varies significantly depending on whether it is realized bit-serial or bit-parallel, two separate cost models for the two architecture classes 1 and 2 are derived [40]. These models are primarily targeting a Min-Sum-based LDPC decoder. However, it will be shown that they can be extended to support decoders which base on the Sum-Product algorithm, as well.

3.1 Bit-parallel LDPC decoder

In the decoder loop at least one system delay is required which should be located at the position leading to the smallest number of registers. A decoder loop for a Min-Sum-based decoder and the equations to calculate the number of registers for the five depicted cross sections is stated in Fig. 3-1.



1) $n \cdot d_V \cdot w = m \cdot d_C \cdot w$

2) $m \cdot \left[d_C + 2 \cdot (w-1) + \lceil ld(d_V) \rceil \right]$

3) $n \cdot d_V \cdot w = m \cdot d_C \cdot w$

4) $n \cdot d_V \cdot w + n \cdot \left(w + \lceil ld(d_V + 1) \rceil \right)$

5) $n \cdot d_V \cdot \left(w + \lceil ld(d_V + 1) \rceil \right)$

Fig. 3-1 Bandwidth of message communication

Thereby, *w* defines the word length of the messages which are communicated between the bit and the check nodes. As discussed in [41] the optimal position for the system delay can be

observed at cross section two. At this point all the information is compressed to the two minima, to the position of the minimum and to the sign signals. Therefore, in the following the registers are assumed to be at this cross section and, thus, the area and energy impact can be neglected.

3.1.1 Logic area

A lower bound of the decoder's silicon area is the accumulated area of all required logic gates. For a rough estimation the control logic of a bit-parallel decoder can be neglected. Thus, the decoder logic mainly consists of the bit-parallel realization of the arithmetic in the n bit- and m check-node instances.

One bit node mainly contains a multi-operand adder and the subsequent subtractors as shown in Fig. 3-2. The multi-operand adder summarizes d_V A-posteriori information and the channel information leading to d_V two-operand adders. Additionally, d_V adders are required to subtract the A-posteriori information from the resulting $L(Q_i)$ value.

Accounting for the required sign extension of the operands to the resulting word length of

$$w_{EXT\_BN} = w + \lceil ld(d_V) \rceil \tag{3-1}$$

bit the bit node includes

$$N_{BN\_P\_FA} = 2 \cdot d_V \cdot \left(w + \lceil ld(d_V) \rceil \right) \tag{3-2}$$

full adders (FA).

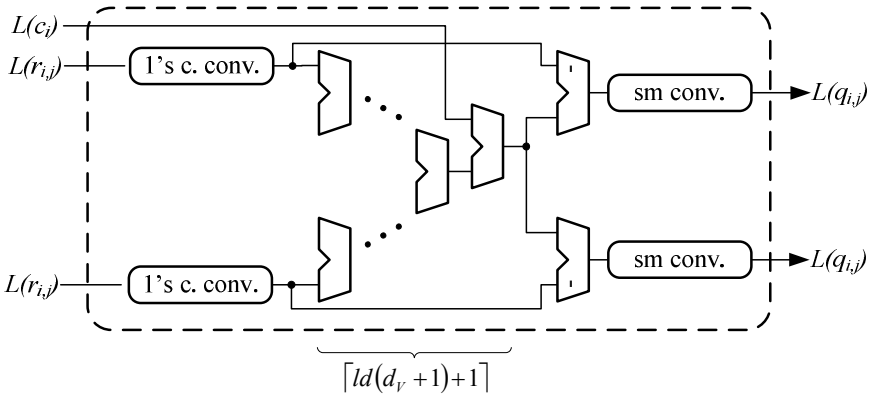


Fig. 3-2 Data-path bit node

This adder scheme requires a two's-complement representation of the A-posteriori information. However, due to the separate sign and magnitude calculation in the check node two conversions between the two number representations are required. The hardware effort of the converters at the input of the bit node includes $d_V \cdot (w\text{-}1)$ XOR-gates to calculate the one's complement. The conversion to two's complement is done using free carry inputs of the

multi-operand adders. Each two's-complement to sign-magnitude converter at the output of the bit node consists of *(w-1)* XOR-gates and *(w-1)* half adders (HA) . With

$$A_{FA} \approx A_{XOR} + A_{HA} \approx 2 \cdot A_{XOR} \tag{3-3}$$

the logic area of the bit-parallel bit node can be roughly estimated to

$$A_{BN\_P} \approx d_v \cdot \left[3.5 \cdot w - 1.5 + 2 \cdot \left\lceil ld(d_v) \right\rceil\right] \cdot A_{FA} . \tag{3-4}$$

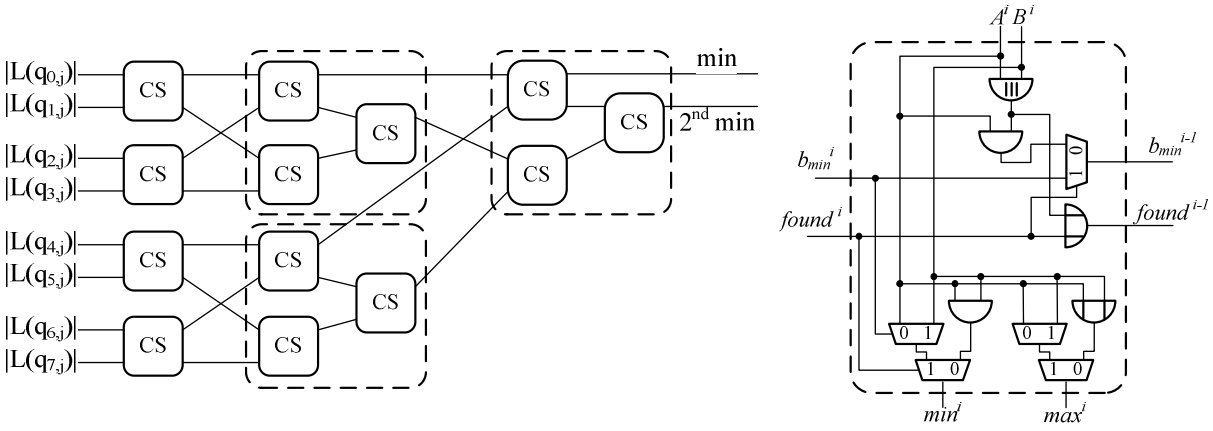
As discussed in chapter 2.2, the Min-Sum check node is built up of a sign calculation, a minimum search, logic to select either the absolute or the second minimum, and an optional post-processing function. Thereby, the major contribution is due to the minimum search. The search for the smallest and second smallest of the d_C magnitudes is a sub problem of the complete sorting of d_C values. A lot of hardware-efficient sorters have been proposed in the past, especially in the field of signal and image processing applications. One sorter architecture, known for its minimal number of compare-and-swap (cs-) elements bases on the odd-even merge algorithm [42]. Therein, subgroups of operands are sorted and afterwards merged together.

For the given problem the merging of the subgroups can be simplified since only the two smallest operands of each group are required. Exemplarily, an eight-operand sorter is illustrated in Fig. 3-3 a) in which each subgroup only consists of two operands. After the initial sorting of these operands, each merge operation requires three additional cs-elements leading to a total number of

$$N_{CN\_P\_CS} = 2 \cdot d_C - 3 \tag{3-5}$$

cs-elements.

One possible implementation of a cs-element would be the subtraction of the two operands with a subsequent swap. Thereby, the sign of the difference indicates whether the two operands have to be swapped. This can be realized by two multiplexer. The critical path of such an element would begin at the LSB input of the subtractor, run through the whole adder ripple path, and end at the outputs of the multiplexers. The output is not known until the sign of the difference is calculated. Therefore, the concatenation of *k* cs-elements would result in a critical path containing *k* adder ripple paths.



a) Odd-even merge sorter **b) Bit compare-and-swap cell**

Fig. 3-3 Odd-even merge sorter

In contrast MSB-first comparators as used e.g. in [43] show a significant timing advantage and, thus, are considered in the following. The structure of such a cs-element is similar to a carry-ripple-adder except that the ripple-path starts at the MSB and ends at the LSB. Instead of using full-adder cells, the cs-element consists of bit compare-and-swap cells (BCS-cells). A possible architecture of such a cell is depicted in Fig. 3-3 b).

In contrast to a FA in which only one carry signal exists in the BCS-cell, two signals propagate to the next bit weight. The $found^i$ signal indicates, if the two operands differ in the bit positions *w-1* downto *i+1*. If they differ in these positions, the second status signal b_{MIN} indicates whether the operand *B* is smaller than operand *A*. As long as the signal $found^i$ is '0', and, thus, no decision is made the minimum of the two inputs A^i and B^i is directed to the min^i output and the maximum to the max^i output. Thus, the min^i (max^i) output is set to '1' ('0'), only if both inputs A^i and B^i are '1' ('0') which can be realized by an AND-gate (OR-gate). In the case of equal inputs the $found^{i-1}$ signal remains '0'. If in contrast the two inputs differ, the signal is set to '1'. In this case the $b_{min}{}^{i-1}$ signal indicates whether the input A^i is the minimum. If the two operands differ in a bit slice $j > i$, the $found^i$ signal is already set. In this case either input A^i or B^i is directed to the min^i output depending on the $b_{min}{}^i$ signal.

In a bit-parallel realization of the check node each cs-element consists of *(w-1)* BCS-cells leading to a total area of the check node of

$$A_{CN\_P} \approx (2 \cdot d_C - 3) \cdot (w - 1) \cdot A_{BCS} . \tag{3-6}$$

Therefore, the total logic area of the basic decoder can be estimated to

$$\begin{aligned} A_{L\_P} \approx\ & m \cdot (2 \cdot d_C - 3) \cdot (w - 1) \cdot A_{BCS} \\ & + n \cdot d_v \cdot \left[3.5 \cdot w - 1.5 + 2 \cdot \lceil ld(d_v) \rceil\right] \cdot A_{FA} . \end{aligned} \tag{3-7}$$

3.1.2 Routing area

Usually, it is not possible to realize the global interconnect on this silicon area and the decoder needs to be expanded artificially as it is done e.g. in [5] achieving a very low utilization of the active silicon area. In such a case the decoder area is significantly larger than the logic area and a separate model for the required routing area is required. Typically, the check nodes are realized in the centre of the macro surrounded by the bit nodes as is shown in Fig. 3-4. In total $2 \cdot n \cdot d_V \cdot w$ interconnect lines have to be realized between the bit and the check nodes. The complexity of the global interconnect can be defined by the Manhattan length of all required connections. Considering an average Manhattan length l_{AVG} of one interconnect line the total required Manhattan length l_{REQ_P} is

$$l_{REQ\_P} = 2 \cdot n \cdot d_v \cdot w \cdot l_{AVG}. \tag{3-8}$$

Thereby, the total Manhattan length and, thus, the routing complexity highly depend on the placement of each bit- and check node in the decoder macro. Obviously, that placement with the smallest Manhattan length should be chosen to reduce the interconnect complexity. Considering a decoder macro with n bit- and m check-node positions the optimization problem is to find the assignment of the node instances to these positions which minimizes this interconnect complexity. A complete search for the optimal placement is not possible since it requires the calculation of the Manhattan length for $m! \cdot n!$ possible placements.

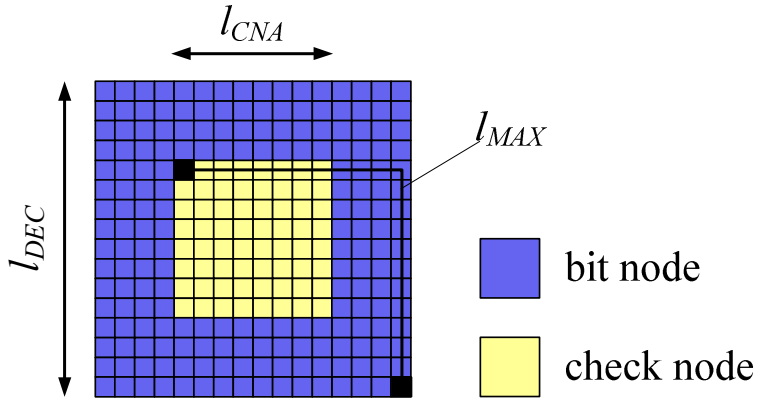


Fig. 3-4 Check- and bit-node placement

Such a placement process is common in VLSI designs and a wide range of optimization strategies have been suggested. One of these strategies is the use of a simulated annealing algorithm. Thereby, the algorithm iteratively reduces a predefined cost function. If the placement should for example be optimized with respect to a reduced interconnect complexity, the summed-up Manhattan length of all required interconnect lines can be used.

The flow chart of the underlying algorithm is shown in Fig. 3-5. At the beginning an initial mapping of the node instances to the positions in the macro is required. This mapping can either be a systematic mapping based on the parity-check matrix or random. Subsequently, the cost of the initial placement is calculated.

In the next step a new mapping is generated by a slight modification of the old mapping. This can for example be realized by an exchange of two check nodes or of two bit nodes. Obviously, the new mapping should be retained, if the new cost is smaller than the old cost. Nevertheless, sometimes a new mapping which increases the costs should also be accepted as otherwise, the algorithm would tend to result in a local cost minimum.

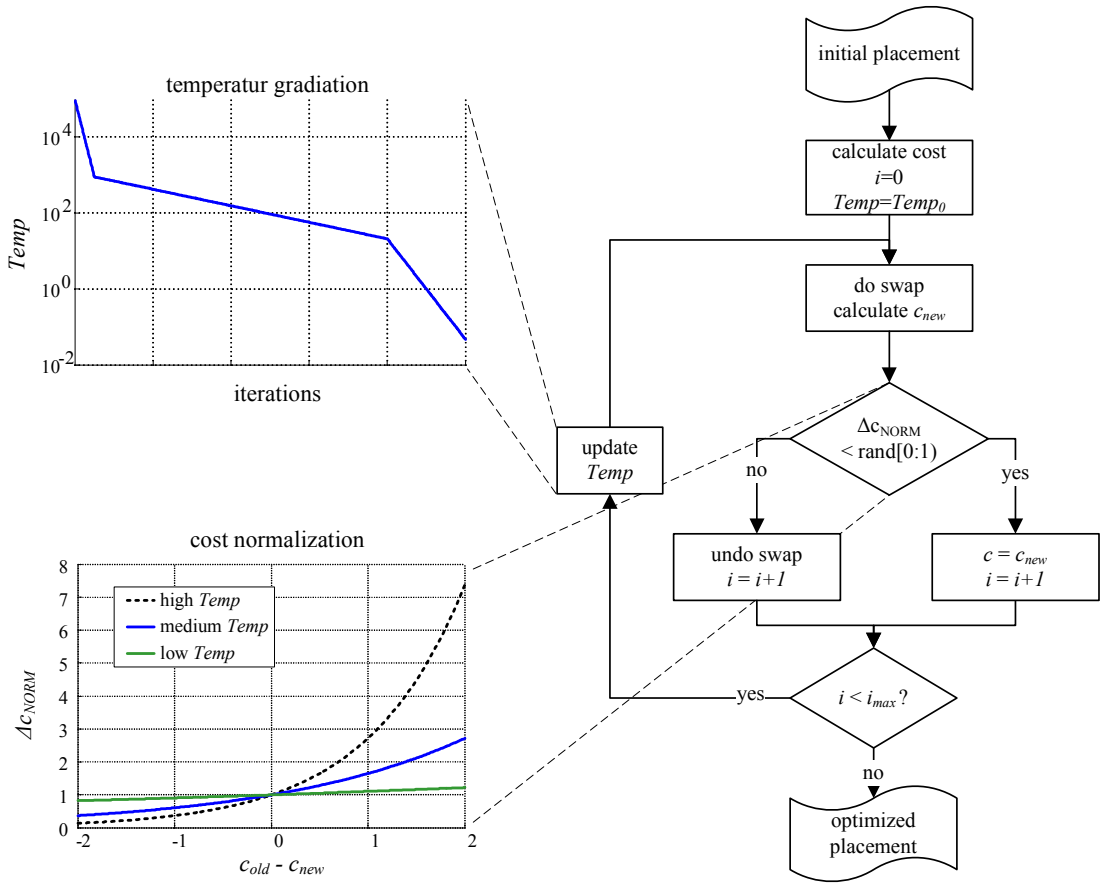


Fig. 3-5 Simulated-annealing algorithm

Therefore, instead of just comparing the two costs itself, a normalized cost difference Δc_{NORM} is calculated using

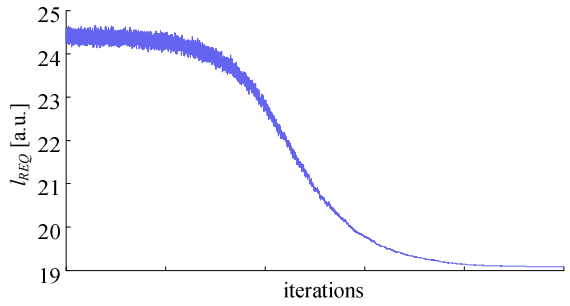
$$\Delta c_{NORM} = e^{\frac{c_{old} - c_{new}}{Temp}} \tag{3-9}$$

With a positive temperature *Temp* this function returns a normalized cost value larger than one, if the costs decrease and a positive result smaller than one, if they increase. The characteristic of this function is also shown in Fig. 3-5 for three different temperatures. In the

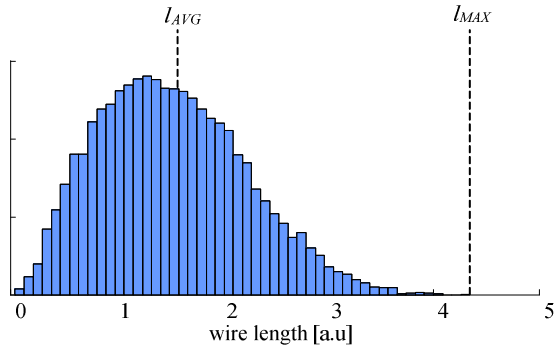
simulated annealing algorithm the result of this normalization is compared to a random number between ‘0’ and ‘1’. If the result is larger than the random number, the modified placement is kept and the costs are updated. Otherwise, the modification is undone. This process is repeated until the defined maximum number of iterations is performed.

Typically, the process starts with a high temperature leading to relatively large normalized cost values for increasing costs. Therefore, the probability that a modification which increases the cost is kept is high at the beginning of the simulated annealing. This results in a jagged progress of the Manhattan length during these iterations as can be seen in Fig. 3-6 a).

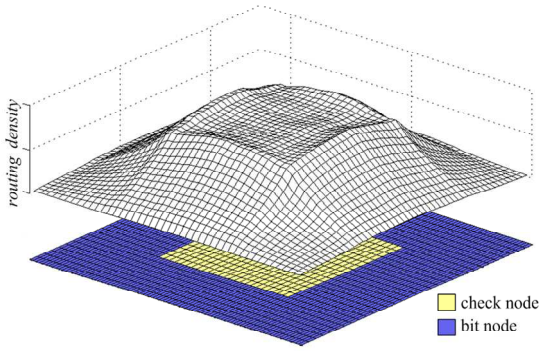
Then the temperature is stepwise reduced as it is shown in Fig. 3-5. Therefore, the curve in Fig. 3-6 a) gets smoother with a higher number of iterations. Finally, at the end of the simulated annealing algorithm the temperature is too low to allow modifications which would increase the costs. In this phase the simulated annealing algorithm degenerates to a local search.



a) Summed Manhattan length during optimization (n = 2048)



b) Wire-length histogram (n = 4,000)



c) Interconnect density

Fig. 3-6 Simulated-annealing based placement optimization

Fig. 3-6 b) shows a typical wire-length histogram after placement optimization. Therein, the average length of one interconnect line l_{AVG} approximately is 0.34 times the maximum possible length l_{MAX} between one bit and one check node. This maximum length connects a check node in the corner of the check-node array with the bit node in the opposing corner of the decoder macro as illustrated in Fig. 3-4. Ideally, the check-node array, as well as the complete decoder macro nearly form a square with a lateral length of l_{CNA} and l_{DEC}, respectively. Then, the maximal length can be expressed in terms of the decoder dimensions as follows

$$l_{MAX} \approx 2 \cdot \left(l_{CNA} + \frac{l_{DEC} - l_{CNA}}{2} \right) = l_{CNA} + l_{DEC} \,. \tag{3-10}$$

Considering that the size of a BCS-cell approximately is four times the size of a full-adder, the analysis of (3-7) reveals that the check nodes occupies about 60 % of the decoder area for typical code parameters. Therefore, the length of the check-node array is about 0.6 times the decoder length leading to a maximum wire length of

$$l_{MAX} \approx 1.77 \cdot l_{DEC} \,. \tag{3-11}$$

In Tab. 3-1 the results for LDPC codes with block sizes between 96 and 8,000 are listed. While code no. 10 is the LDPC code specified in [6], the other codes are randomly chosen from [44]. Although the code complexities $n \cdot d_V$ of the analyzed codes vary between 288 and 24,000, the ratio between the average and the maximum Manhattan length varies only between 0.30 and 0.37.

Tab. 3-1 Interconnect properties of various LDPC codes

Code nr.	n	m	d_V	d_C	$n \cdot d_V$	l_{AVG} / l_{MAX}	ρ_{AVR} / ρ_{MAX} vertical	ρ_{AVR} / ρ_{MAX} horizontal
1	96	48	3	6	288	0.33	0.58	0.59
2	408	204	3	6	1224	0.31	0.56	0.55
3	408	204	3	6	1224	0.30	0.55	0.56
4	408	204	3	6	1224	0.31	0.54	0.50
5	816	408	3	6	2448	0.31	0.52	0.57
6	816	408	5	10	4080	0.34	0.52	0.58
7	816	408	5	10	4080	0.34	0.52	0.57
8	816	408	5	10	4080	0.34	0.50	0.55
9	999	111	3	27	2997	0.37	0.36	0.33
10	1008	504	3	6	3024	0.32	0.54	0.46
11	2048	384	6	32	12288	0.37	0.42	0.40
12	4000	2000	3	6	12000	0.34	0.57	0.54
13	4000	2000	4	8	16000	0.35	0.55	0.55
14	8000	4000	3	6	24000	0.35	0.55	0.45

This observation allows for an estimation of the average Manhattan length in dependence of the decoder side length for any LDPC code. The resulting average Manhattan length is

$$\bar{l} \approx 0.35 \cdot l_{MAX} \approx 0.6 \cdot l_{DEC} \,. \tag{3-12}$$

Using (3-8) and (3-12) the total Manhattan length of the global interconnect of a bit-parallel decoder can be estimated to

$$l_{REQ\_P} \approx 1.2 \cdot n \cdot d_v \cdot w \cdot l_{DEC} \,. \tag{3-13}$$

The basic idea for an estimation of the required routing area is to compare this length with the available routing length l_{AVAIL} in the decoder macro. An estimation of the available Manhattan length requires an approximation of the interconnect length which can be realized in one metal layer. Considering a routing pitch p, $M_{ROUTING}$ metal layers and a metal layer usage of u, the available routing length can be estimated as

$$l_{AVAIL} \approx u \cdot \frac{l_{DEC}^2}{p} \cdot M_{ROUTING} \,. \tag{3-14}$$

Typically, the number of metal layers in a CMOS process is not sufficient to realize the interconnect solely atop of the node logic and, thus, l_{AVAIL} is smaller than l_{REQ}. In such a case a realization of the global interconnect requires an artificial expansion of the decoder macro. This can be done e.g. by a uniform stretch as illustrated in Fig. 3-7. An important feature of this uniform stretch is that the dependencies derived so far still hold.

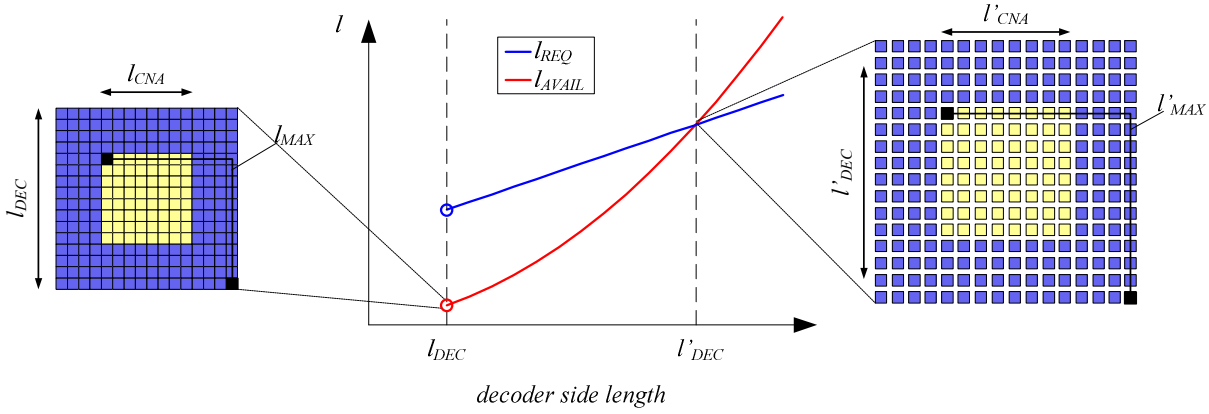


Fig. 3-7 Uniform stretch of decoder

As illustrated by increasing the decoder area two effects can be observed. Starting from a decoder area which is equal to the logic area the available routing length is smaller than the required interconnect length. An increase of the decoder side length linearly increases the required routing length (3-13). The second effect is the quadratic increase of the available routing length (3-14). While increasing the decoder side length, the required and the available routing length converge. The side length l_{DEC} which brings both into match is the minimum side length of the decoder.

Before calculating the decoder side length and the routing area a further improvement of the estimation for the available Manhattan length is necessary. If the decoder area is increased artificially, the utilization of the active silicon area is reduced. This is also true for the lower metal layers which are used to realize the local interconnect. To minimize the required routing area the lower metal layers should also be utilized for the global interconnect in these regions between the nodes. The resulting available Manhattan length can therefore be approximated using

$$l_{AVAIL} \approx \frac{u}{p} \cdot \left[l_{L\_P}^2 \cdot M_{ROUTING} + min\left(l_{DEC}^2 - l_{L\_P}^2, 0 \right) \cdot M \right]. \tag{3-15}$$

The first term in (3-15) is the available routing length on the $M_{ROUTING}$ layers atop of the nodes. If the decoder area is larger than the logic area, the second term approximates the available interconnect length in the regions between the node logic using M metal layers.

The decoder side length can now be estimated based on (3-13) and (3-15) leading to

$$1.2 \cdot n \cdot d_v \cdot w \cdot l_{DEC\_P} = \frac{0.5}{p} \cdot \left[l_L^2 \cdot M_{ROUTING} + \left(l_{DEC\_P}^2 - l_L^2 \right) \cdot M \right]. \tag{3-16}$$

Therein an utilization factor of $u = 0.5$ is assumed. Even if an ideal routing algorithm would exist, it would not be possible to use 100% of each metal layer. The reason is the structure of the routing problem. The interconnect density for an optimized placement for the (2048,1723) code [6] is depicted in Fig. 3-6 c). While a high interconnect density at the boundary of the check-node array is observed, the density at the outer part of the macro is low. A utilization factor can be derived by comparing the average ρ_{AVG} to the maximum density ρ_{MAX}. The quotient of these two values for the different codes is given in Tab. 3-1, as well. Although the quotient varies between 0.4 and 0.6 a utilization factor of $u = 0.5$ is assumed in the following as a first-order model.

Based on (3-16) the decoder side length and, thus, the routing area can be derived which is given by

$$A_{R\_P} = l_{DEC\_P}^2 \approx \left(1.2 \cdot \frac{n \cdot d_V \cdot w}{M} \cdot p + \sqrt{\left(1.2 \cdot \frac{n \cdot d_V \cdot w}{M} \cdot p \right)^2 - l_L{}^2 \cdot \frac{M_{ROUTING} - M}{M}} \right)^2. \tag{3-17}$$

3.1.3 Iteration period

For bit-parallel decoder implementations the iteration period can be calculated by summing-up the critical path along the decoder loop (Fig. 3-1). Basically, this loop consists of four parts: the arithmetic in the bit node, the interconnect between the bit- and the check node, the arithmetic in the check node, and the interconnect back to the bit node. To estimate the

critical path the bit- and check-node pair with the longest interconnect length l_{MAX} has to be considered. For a rough approximation of the interconnect delay, the RC-delay of that interconnect line is used which can be estimated as

$$T_{INT\_P} \approx 0.69 \cdot R' \cdot C' \cdot l_{MAX}^{\,2} \approx 1.76 \cdot R' \cdot C' \cdot l_{DEC\_P}^{\,2} \tag{3-18}$$

with *R'* being the resistive and *C'* the capacitive load per unit length. The critical path inside the bit node (see Fig. 3-2) runs through

$$\lceil ld(d_V + 1) \rceil + 1 \tag{3-19}$$

adder and subtractor stages. Considering a carry ripple adder implementation of the multi-operand adder inside the bit node and the extension of the word length the path can be approximated by

$$T_{BN\_P} \approx \left(2 \cdot \lceil ld(d_V + 1) \rceil + w\right) \cdot T_{FA} . \tag{3-20}$$

The depth of the odd-even merge tree in the check node is

$$2 \cdot \lceil ld\,(d_C) \rceil - 1 \tag{3-21}$$

leading to a critical path of

$$T_{CN\_P} \approx \left(2 \cdot \lceil ld(d_C) \rceil + w - 2\right) \cdot T_{BCS} . \tag{3-22}$$

With (3-18), (3-20), and (3-22) the iteration period can be written as

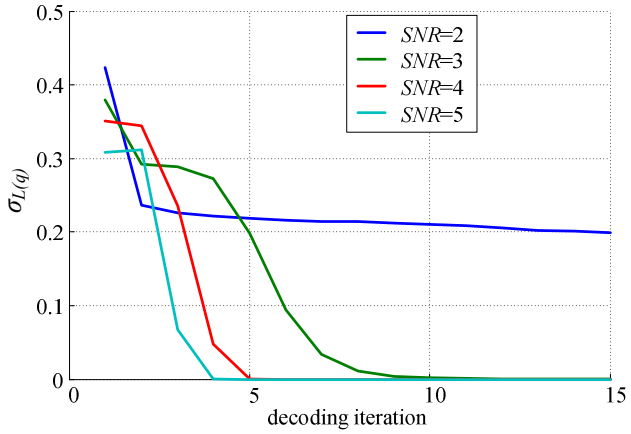
$$T_{IT\_P} \approx \left(2 \cdot \lceil ld(d_V + 1) \rceil + w\right) \cdot T_{FA} + \left(2 \cdot \lceil ld(d_C) \rceil + w - 2\right) \cdot T_{BCS} + 3.52 \cdot R' \cdot C' \cdot l_{DEC}^{\,2} . \tag{3-23}$$

3.1.4 Energy per iteration

The dynamic power consumption highly depends on the switching activities of each capacitive node inside the decoder. The AC power of the interconnect lines depends e.g. on the mean switching activities of the quantized A-priori values $L(q_{i,j})$ and A-posteriori values $L(r_{i,j})$. These in turn highly depend on the considered signal-to-noise ratio (SNR). Fig. 3-8 exemplarily shows the switching activities for each decoding iteration for the (2048,1723) code [6] with $w = 6$. For a very low SNR of two dB the switching activity saturates to a relatively high activity as the decoder does not converge. For higher SNR a high switching activity is observed in the first decoding iterations and a lower activity when the decoder has converged. Additionally, the average switching activities depend on the number of maximal decoding iterations as more iterations lower the average switching activity.

However, typically a certain BER is specified which has to be guaranteed for a certain application. Considering for example eight decoding iterations, it is possible to determine the SNR for a given code to reach a BER of e.g. 10^{-5}. In Fig. 3-8 b) the average switching activities for the different LDPC codes of Tab. 3-1 for the individual SNR to reach a BER of

10^{-5} are listed. These switching activities are nearly independent on the used LDPC code, although there seem to be exceptions (see code nr. 9). Nevertheless, this observation allows for an approximation of the switching activities in dependency of the BER without prior knowledge of the LDPC code.



a) $L(q_{i,j})$ (IEEE 802.3an LDPC code)

Code nr.	$\sigma_{L(q)}$	$\sigma_{L(r)}$
1	0.14	0.30
2	0.17	0.34
3	0.17	0.33
4	0.14	0.30
5	0.18	0.34
6	0.16	0.33
7	0.16	0.33
8	0.16	0.33
9	0.41	0.29
10	0.19	0.34
11	0.20	0.33
12	0.19	0.34
13	0.20	0.35
14	0.19	0.34

b) Average switching activity of various codes

Fig. 3-8 Switching activity (8 iterations, BER 10^{-5})

Based on these results and the total wire length of the global interconnect l_{REQ} the energy required to load and unload the interconnect calculates as follows

$$E_{INT\_P} \approx \frac{1}{2}\left(\sigma_{L(q)}(BER)+\sigma_{L(r)}(BER)\right)\cdot \xi \cdot C' \cdot \frac{l_{REQ\_P}}{2}\cdot V_{DD}^{\,2}. \tag{3-24}$$

As the routing metal layers are only utilized by about 50 % (see chapter 3.1.2) the average distance between two interconnect lines is not the minimal allowed distance in the considered CMOS technology. This is accounted for by the factor ξ. An estimation of this factor can be based on the routing scenarios in Fig. 3-9. In routing scenario a) all four possible tracks are occupied by interconnect lines and the capacitance per wire is

$$C'_{a)} = C'. \tag{3-25}$$

Due to the utilization of just 50 % of the metal layers only two of the four routing tracks would be used in average. In routing scenario b) a uniform distribution of the wires is considered. If considering that the pitch of p equally splits into a wire width and a minimal wire spacing of $p/2$, ξ would be

$$\xi = \frac{C'_{b)}}{C'_{a)}} = \frac{C'_{a)}/2.3}{C'_{a)}} = 0.43. \tag{3-26}$$

In contrast the possible routing scenario c) would result in

$$\xi = \frac{C'_{c)}}{C'_{a)}} = \frac{\lfloor C'_{a)} + 0.5 \cdot C'_{a)} \rfloor}{2 \cdot C'_{a)}} = \frac{1.5}{2} = 0.75 \,. \tag{3-27}$$

The calculation of the average ξ would require the analysis of all possible routing scenarios and an analysis of the probabilities of these scenarios in an actual routed decoder. However, for the targeted accuracy of the cost models the mean ξ of the two considered routing scenarios b) and c) is sufficient leading to a ξ of 0.6.

For a rough estimation of the energy per bit- and check-node operation the switching activity of the A-priori information $L(q_{i,j})$ (A-posteriori information $L(r_{i,j})$) is used as the switching activity of all the capacitive nodes inside the check (bit) node. Thus, the energy for one iteration of the node logic can be estimated as

$$\begin{aligned} E_{L\_P} \approx\; & n \cdot \sigma_{L(r)}(BER) \cdot d_V \cdot (3.5 \cdot w - 1.5 + 2 \cdot \lceil ld(d_V) \rceil) \cdot E_{FA} \\ & + m \cdot \sigma_{L(q)}(BER) \cdot (2 \cdot d_C - 3) \cdot (w - 1) \cdot E_{BCS} \end{aligned} \tag{3-28}$$

with E_{FA} and E_{BCS} being the average energy per operation for a full adder and a BCS-cell.

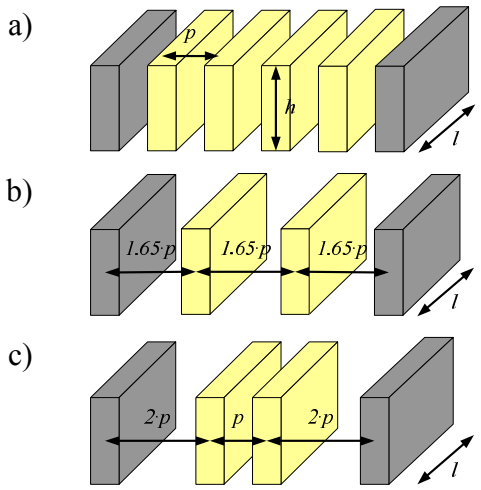


Fig. 3-9 Capacitance normalization factor α

The clock power can be neglected for bit-parallel decoders, as the number of registers is minimized and, thus, have almost no effect on the power consumption. Based on (3-24) and (3-28) the energy per iteration can be estimated to

$$E_{IT\_P} = E_{L\_P} + E_{INT\_P}. \tag{3-29}$$

3.2 Bit-serial LDPC decoder

As the decoder features are highly influenced by the global interconnect, bit-serial decoders are promising especially for larger block lengths. Due to the bit-serialization of the node arithmetic the complexity of the nodes can be reduced, as well. E.g. the complexity of the minimum search can be reduced as only one instead of *(w-1)* BSC-cells is required for

each compare-and-swap element. However, as the cs-operation is processed in multiple clock cycles the status signals *found* and b_{min} have to be stored in a register as shown in Fig. 3-10.

The two additional storage elements also need to be reset at the beginning of a new comparison. Therefore, the area of the minimum search can be estimated as

$$A_{CN\_S\_MIN} \approx (2 \cdot d_c - 3) \cdot (A_{BCS} + 2 \cdot A_{REG} + 2 \cdot A_{MUX}). \quad (3\text{-}30)$$

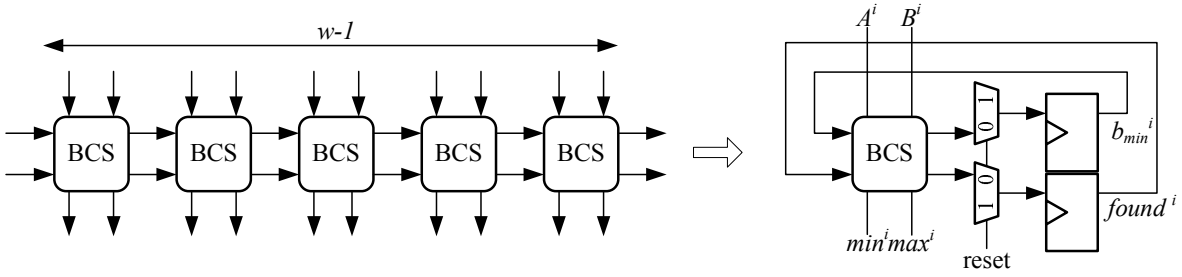


Fig. 3-10 Bit-parallel and bit-serial CS element

In contrast to the area model of the bit-parallel check node the area contribution of the registers to store the two minima, the position of the minimum and the signs of the check-node messages can not be neglected for a bit-serial check node. The area of these registers e.g. approximately is

$$A_{CN\_S\_M\_REG} \approx (d_c + 2 \cdot (w - 1) + \lceil ld(d_C) \rceil) \cdot A_{REG} \quad (3\text{-}31)$$

Furthermore, additional control logic has to be included into the area model to prevent an underestimation of the decoder's logic area. Therefore, a more detailed look at the implementation of a bit-serial check node is required. A block diagram of a possible architecture is depicted in Fig. 3-11.

The bit-serial input stream of the bit-node messages is stored in registers at the input of the node. In the first clock cycle the signs of the bit-node messages are processed in the sign calculation unit which is not depicted. The resulting bits are stored in the select-and-convert units at the output of the check node. In the following clock cycles the minimum search is performed. The underlying structure of the minimum search is equal to the one in Fig. 3-3 but using the bit-serial cs-elements of Fig. 3-10. The bit-serial output of the minimum search is stored in two stacks. After the LSBs of the bit-node messages have been processed, the location of the minimal input can be determined based on the b_{min} signals of all the cs-elements. To reduce the number of registers the position is encoded to a minimal number of $\lceil ld(d_C) \rceil$ bits.

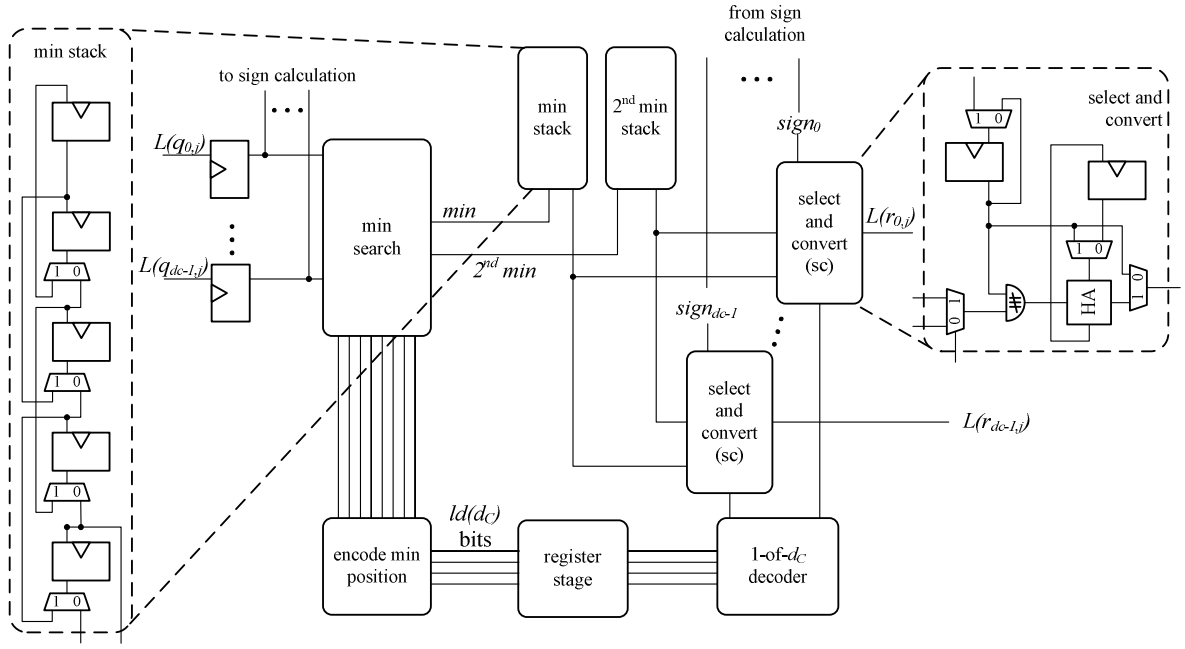


Fig. 3-11 Block diagram of a bit-serial check node

In the next clock cycle for each check-node message either the minimum or the second minimum is selected. Therefore, based on the stored position of the minimum a 1-of-d_C decoder generates d_C select signals. As the check-node messages are sent in a LSB-first fashion to the bit nodes, the required conversion from sign-magnitude to two's-complement representation has to be performed in the check node, as well. Therefore, the bits of the check-node message magnitudes are combined with the corresponding sign using a XOR-gate. As the sign also has to be added to the LSB, a subsequent bit-serial adder is also implemented in the select-and-convert unit.

Neglecting the encoding and decoding of the minimum position the area of one check node can be approximated as

$$\begin{aligned} A_{CN\_S} \approx\ & d_C \cdot A_{REG} + A_{CN\_S\_MIN} + A_{CN\_S\_M\_REG} \\ & + 2 \cdot A_{CN\_S\_STACK} + d_C \cdot A_{CN\_S\_SC} \end{aligned} \tag{3-32}$$

with

$$A_{CN\_S\_STACK} \approx (w-1) \cdot A_{REG} + (w-2) \cdot A_{MUX} \tag{3-33}$$

and

$$A_{CN\_S\_SC} \approx 2 \cdot A_{REG} + 4 \cdot A_{MUX} + A_{FA}\,. \tag{3-34}$$

A block diagram of a possible bit-serial bit-node architecture is depicted in Fig. 3-12. The bit-serial input data stream is stored in input registers. In each clock cycle the d_V input bits and one bit of the channel information is summarized in a bit-serial multi-operand adder.

Therefore, each FA is extended with a register to store the carry signal and a multiplexer which allows a reset of the bit-serial adder.

Subsequently, the $L(r_{i,j})^k$ signal is subtracted from the resulting bit of $L(Q_i)^k$ by adding the two's complement of $L(r_{i,j})^k$. Both required operations, the two's-complement calculation and the summing-up are done in the subtraction-and-register stage. The resulting bit is stored in a register chain. Due to the extended word length of the resulting bit-node message, the adder would have to calculate for $\left(w+\lceil ld(d_V)\rceil\right)$ clock cycles, if no additional logic would be realized. However, it is possible to read out the carry overflows in the w^{th} clock cycle and to perform the summing-up of the upper bits in parallel. Therefore, a carry overflow adder sums-up all the carry bits to one operand. The highest bit of the two's-complement converted check-node message is then added to this operand in parallel in the subtraction-and-register stage, so the complete bit-node message is calculated after w clock cycles. Additionally, the bits of the $L(Q_i)$ value are stored in a separate register stage.

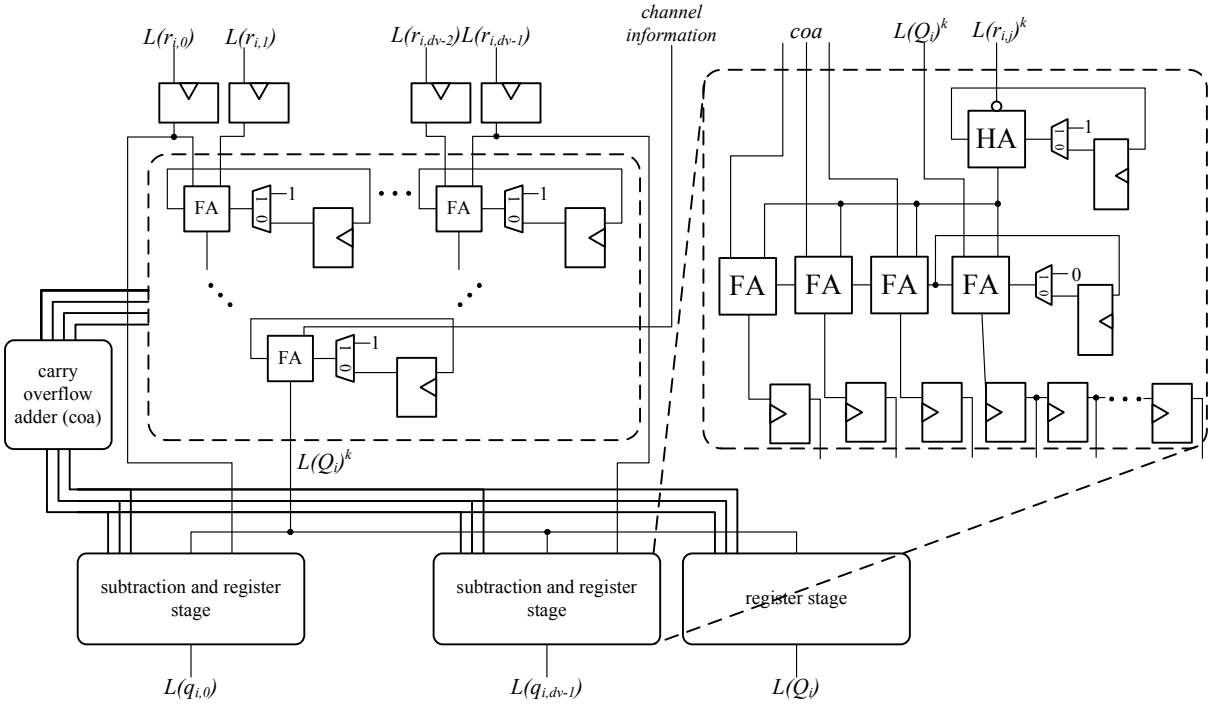


Fig. 3-12 Bit-serial bit node

The required silicon area of the bit-serial structure depicted in Fig. 3-12 approximately is

$$\begin{aligned} A_{BN\_S\_MOA} \approx\; & d_V \cdot \left(A_{HA\_SERIAL} + 2 \cdot A_{FA\_SERIAL} + \lceil ld(d_V)\rceil \cdot A_{FA}\right) \\ & + \left(d_V + 1\right) \cdot \left(w + 1 + \lceil ld(d_V)\rceil\right) \cdot A_{REG}. \end{aligned} \tag{3-35}$$

The resulting bit-node message has a word length of $\left(w+\lceil ld(d_V)\rceil\right)$ bit in two's-complement representation. In the next iteration the check node requires a w-bit wide bit-node message in sign-magnitude representation. Thus, a saturation and two's-complement

conversion logic, as it is shown in Fig. 3-13, is required. As the sign of the bit-node message is not affected by both operations, the calculation can be performed while the sign is sent to the check node. Therefore, these operations do not affect the critical path.

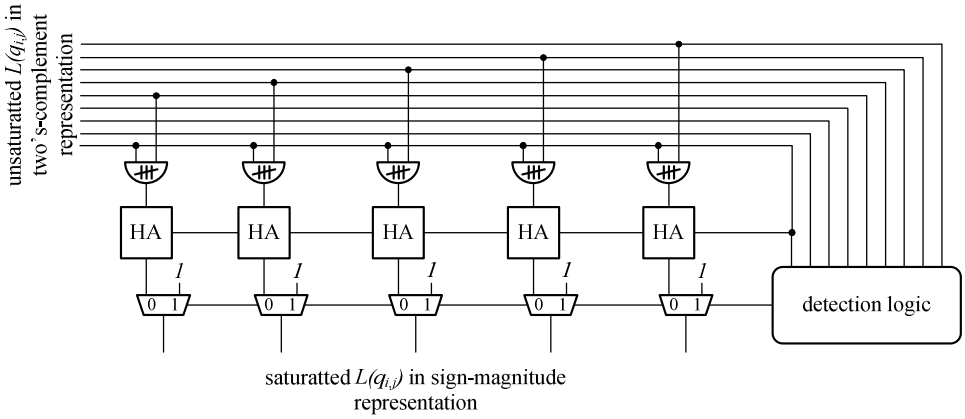


Fig. 3-13 Saturation and two's-complement to sign-magnitude conversion

If the sign of the unsaturated bit-node message which is stored in the MSB register of the subtraction-and-register stage is '1', the lower five bits of the operand are inverted and a '1' is additionally fed into the LSB HA. Otherwise, the sign-magnitude conversion passes the original bits to the saturation logic. Furthermore, based on the unsaturated operand a detection logic determines, if the magnitude can be represented using five bits. Otherwise, a control signal is set which selects the maximal possible magnitude ('11111') as the magnitude of the saturated bit-node message. As this saturation logic has to be instantiated d_V times in one bit node, the area impact approximately is

$$A_{BN\_S\_SAT} \approx d_V \cdot (w-1) \cdot (A_{XOR} + A_{HA} + A_{MUX}) . \tag{3-36}$$

By combining (3-35) and (3-36) the resulting area of a bit-serial bit node can be estimated to

$$A_{BN\_S} \approx A_{BN\_S\_MOA} + A_{BN\_S\_SAT} . \tag{3-37}$$

The logic area of the complete bit-serial decoder approximately is

$$A_{L\_S} \approx n \cdot A_{BN\_S} + m \cdot A_{CN\_S} . \tag{3-38}$$

Instead of $2 \cdot w$ interconnect lines between two connected nodes, only one interconnect line in each direction is required. Therefore, the interconnect complexity is reduced by a factor w and the routing area can be approximated as

$$A_{R\_S} = l^2_{DEC\_S} \approx \left(1.2 \cdot \frac{n \cdot d_V}{M} \cdot p + \sqrt{\left(1.2 \cdot \frac{n \cdot d_V}{M} \cdot p \right)^2 - l^2_{L\_S} \cdot \frac{M_{ROUTING} - M}{M}} \right)^2 . \tag{3-39}$$

The two data flow swaps in the decoder loop lead to a minimum of $2 \cdot (w+1)$ clock cycles per iteration. The clock period is determined by the longest critical path of the arithmetic in

the nodes and the wire delay. In contrast to the nodes in the bit-parallel decoder, the critical path is reduced due to the missing ripple path and is mainly determined by the depth of the arithmetic. The critical path of the minimum search can be approximated as

$$T_{CN\_S} \approx \left(2 \cdot \lceil ld(d_c) \rceil - 1\right) \cdot T_{BCS} \,. \tag{3-40}$$

The critical path of the bit-serial bit node runs through the multi-operand adder, the FA which realizes the subtraction of $L(r_{i,j})$, and the FAs required for the upper bits in the last clock cycle of the bit-node calculation. Thus, the critical path approximately is

$$T_{BN\_S} \approx \left[\lceil ld(d_v + 1) \rceil + 1 + \lceil ld(d_v) \rceil\right] \cdot T_{FA} \,. \tag{3-41}$$

The iteration period of a bit-serial decoder can be roughly estimated as

$$T_{IT\_S} \approx 2 \cdot (w+1) \cdot max\left(T_{CN\_S}, T_{BN\_S}, T_{INT\_S}\right) \tag{3-42}$$

with T_{INT_S} calculated using (3-18) with the maximum interconnect length l_{MAX} for the bit-serial decoder.

The arithmetic operations which have to be calculated in a bit-serial decoder do not differ to a bit-parallel realization. The only difference in the calculation is that the operations are carried out sequentially. Therefore, the energy of the logic in a bit-serial decoder can roughly be estimated based on (3-28), as well. The average energy to load and unload the interconnect per iteration can be estimated using (3-24). However, the switching activities of the interconnect does not depend on the relation of one bit in consecutive iterations but on the relation of subsequent bits in one $L(q_{i,j})$ or $L(r_{i,j})$ value. Nevertheless, the same switching activity as for the bit-parallel decoder is assumed in the following.

One major difference in the energy of the bit-serial decoder is the higher clock power as a significantly higher number of registers is required in the bit-serial decoder. The total number of registers consists of the swap registers, the registers to hold the carry signal in the multi-operand adder and the registers to hold the *found*- and b_{min}- signals in the minimum search. The total number of registers approximately is

$$N_{DEC\_S\_REG} \approx \underbrace{3 \cdot m \cdot d_C + 2 \cdot m \cdot (w-1)}_{check-node} + \underbrace{2 \cdot n \cdot d_V + n \cdot (d_V + 1) \cdot w_{EXT\_BN}}_{bit-node}. \tag{3-43}$$

The required energy of each clock cycle to load the input capacitance of all the registers in the decoder approximately is

$$E_{CLK\_CYC\_S} \approx N_{DEC\_S\_REG} \cdot C_{REG\_IN} \cdot V_{DD}^{\,2} \tag{3-44}$$

with C_{REG_IN} being the accumulated input capacitance of all clock inputs of the registers. Considering that one decoding iteration requires $(2 \cdot w+2)$ clock cycles the corresponding clock energy approximately is

$$E_{CLK\_S} \approx (2 \cdot w + 2) \cdot N_{DEC\_S\_REG} \cdot C_{REG\_IN} \cdot V_{DD}^2 . \quad (3\text{-}45)$$

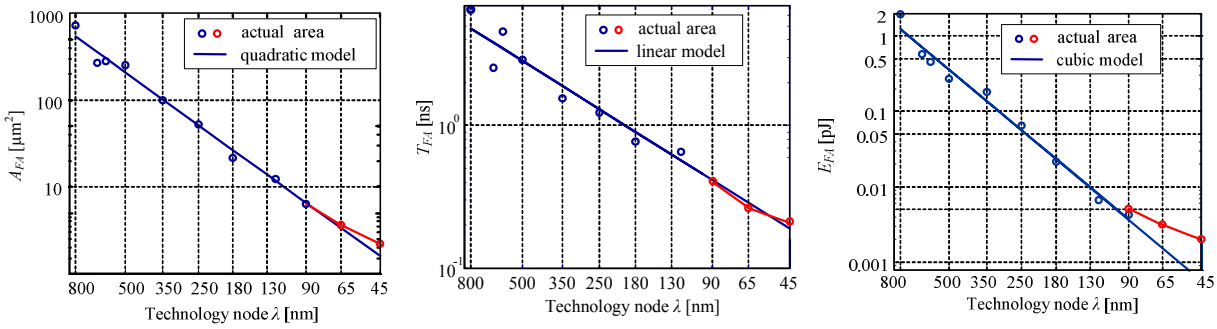
Thus, the total energy per iteration is

$$E_{IT\_S} = E_{L\_S} + E_{INT\_S} + E_{CLK\_S} . \quad (3\text{-}46)$$

3.3 Quantitative analysis of decoder architectures

The derived models (see also [45]) allow for an estimation of decoder features based on the features of basic digital components like e.g. FAs and on technology parameters like the routing pitch. For an estimation of the actual costs the features of logic gates, interconnect lines, and registers have to be introduced into these models. In the following a quantitative analysis of the cost models is done considering a physically optimized full-custom design style. Therefore, the absolute cost metrics have to be regarded as lower bounds.

Exemplarily, the technology trend of the silicon area, delay, and energy per operation for a 24 transistor FA cell is illustrated in Fig. 3-14. Area scaling models predict a quadratic scaling of the area with the technology feature size which is approved by FA implementations in CMOS technologies down to 90-nm. However, preceding to smaller feature sizes this trend is not maintained and the actual area differs from the estimated one by up to a factor of two. The reasons are more restrictive design rules and the introduction of design-for-manufacturing (DFM) rules to achieve a sufficient yield in chip production.



a) Area trend **b) Delay trend** **c) Energy trend**

Fig. 3-14 Technology trend of FA

Nevertheless, in the following a quadratic scaling of the area is assumed. Based on the area trend the area of a FA can be approximated to

$$A_{FA} \approx 1000 \cdot \lambda^2 . \quad (3\text{-}47)$$

In contrast the delay trend of a FA (Fig. 3-14 b)) shows that down to 45-nm the delay nearly follows the expected linear model and the delay of a full-adder can be estimated to

$$T_{FA} \approx 5 \cdot 10^{-3} \frac{s}{m} \cdot \lambda . \quad (3\text{-}48)$$

The reason for the continuity of the linear decreasing delay even for deep-submicron technologies is the introduction of constant-voltage scaling for technologies smaller than 90-nm. In the technology generations down to 90-nm the supply voltage has been decreased with the technology feature size leading to scaling of the energy with λ^{-3} as it can be seen in Fig. 3-14 c). However, the constant voltage scaling results in an energy scaling with only λ^{-1}. To cope with this trend the energy per full-adder is estimated in dependence of the supply voltage and the technology feature size to

$$E_{FA} \approx 35 \cdot V_{DD}{}^2 \cdot \lambda \frac{nJ}{mV^2}. \tag{3-49}$$

The features of the other basic components which are required for the estimation of the ATE-costs like the BCS-cells have been derived in a similar way and are listed in Tab. 3-2.

Tab. 3-2 ATE features of building blocks

	FA	BCS	SCAN REG	MUX	HA	XOR
Silicon area [m^2]	$1{,}000 \cdot \lambda^2$	$4{,}000 \cdot \lambda^2$	$2{,}000 \cdot \lambda^2$	$500\ \lambda^2$	$800 \cdot \lambda^2$	$500 \cdot \lambda^2$
Delay [s]	$5 \cdot 10^{-3} \cdot \lambda$	$6 \cdot 10^{-3} \cdot \lambda$				$3 \cdot 10^{-3} \cdot \lambda$
Energy per operation [nJ / mV^2]	$35 \cdot V_{DD}{}^2 \cdot \lambda$	$190 \cdot V_{DD}{}^2 \cdot \lambda$				

Additionally, the features of interconnect lines have to be modeled, as well. E.g. the capacitive load per unit length, which in a first-order model is independent of the technology feature size [46], can be determined to about $C' = 0.16\ ^{fF}/_{\mu m}$. In contrast to the capacitive load the resistive load highly depends on the technology feature size [46]. For a rough estimation the resistive load can be approximated to scale with λ^{-2}. Considering the miller-effect on the capacitive load for worst-case timing the delay of an interconnect line with the length l can be approximated to

$$T_{INT} \approx \frac{0.2 \cdot 10^{-18}}{\lambda^2} \cdot l^2 \cdot s. \tag{3-50}$$

Furthermore, the relation between the number of metal layers used for the internal node connections and those for the global interconnect and the routing pitch are required for an estimation of the interconnect features. Considering that three metal layers are utilized to realize the nodes the relation between M and $M_{ROUTING}$ is as follows

$$M_{ROUTING} = M - 3. \tag{3-51}$$

The estimation of the routing pitch has to consider its dependency of the technology feature size. In the following a linear dependency is assumed and the routing pitch is approximated as

$$p = 2 \cdot \lambda. \tag{3-52}$$

Finally, the capacitance of the clock input of the considered registers has to be approximated for estimating the clock power of the bit-serial decoder. The input capacitance of a static master-slave flip-flop including local clock wires can be determined by circuit simulations and can be approximated to

$$C_{REG\_IN} \approx 45 \frac{nF}{m} \cdot \lambda \,. \tag{3-53}$$

3.3.1 Bit-parallel decoder

The introduction of the implementation-style-dependent features into the decoder cost-models allows for a quantitative analysis of the decoder features. E.g. based on (3-7) and on the silicon area of the basic components taken from Tab. 3-2 the total logic area of a bit-parallel decoder implementation can be estimated. Furthermore, introducing the considered routing pitch of $2 \cdot \lambda$ into (3-17) yields in an approximation of the routing area. In Fig. 3-15 a) the routing and logic area are outlined for an exemplary LDPC code and technology parameter set. While the logic area scales linearly with the code block length, the routing area scales quadratically. The resulting silicon area can then be estimated as the maximum of the routing and the logic area. For small block lengths the interconnect between the nodes can be realized atop of the logic area leading to a logic dominated decoder. In contrast for higher block lengths the decoder area is determined by the area which is required to route the complex interconnect. Based on (3-7) and (3-17) the boundary between logic and routing dominated decoder can be derived. Approximately no artificial area extension is required, if

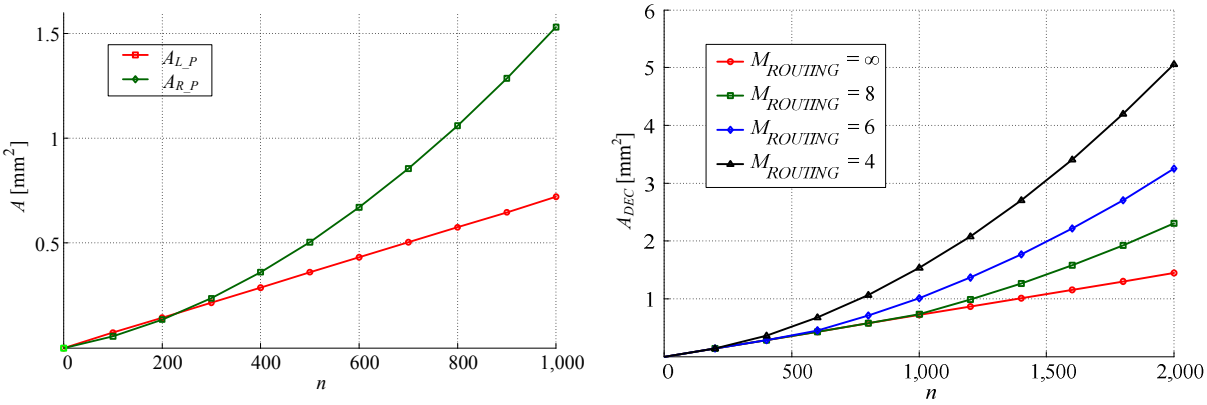
$$n \cdot d_v < 500 \cdot \frac{M_{ROUTING}^2}{w} \tag{3-54}$$

leading to a logic dominated decoder. Therefore, the code complexity $n \cdot d_V$ determines whether a decoder is logic or routing dominated.

Obviously, the code complexity range for which no area extension is required gets smaller the smaller the number of routing layers is. Fig. 3-15 b) quantitatively illustrates this behavior. For $M_{ROUTING} = \infty$ the decoder area would be the logic area. By restricting the number of routing layers to a large number of eight, only code complexities $n \cdot d_V$ of less than about 6,000 would result in a logic dominated decoder. A further limitation to six and four metal layers would reduce the range to 3,500 and 1,500, respectively.

The iteration period of the bit-parallel decoder is the sum of the critical paths of the bit and check node and twice the propagation delay of the longest interconnect line (see (3-23)).

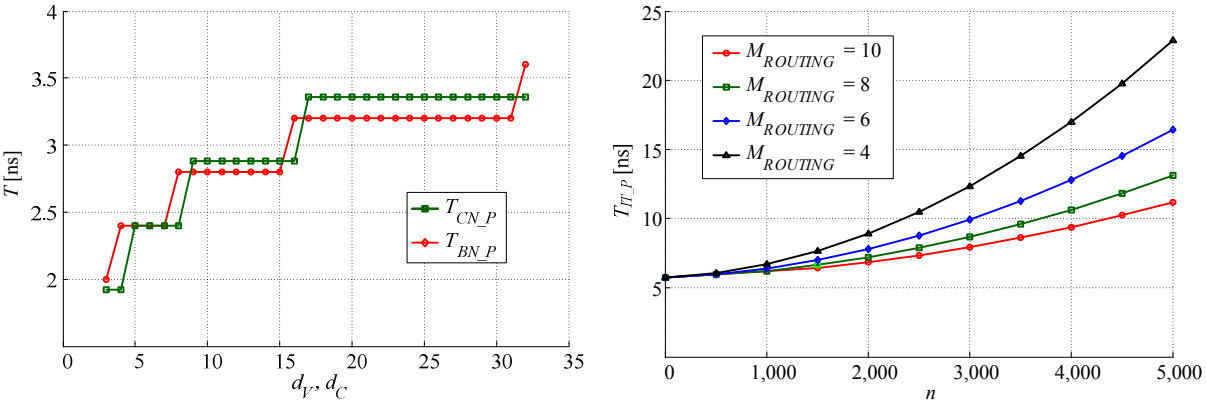
Fig. 3-16 a) illustrates the critical path of both nodes for a word length $w = 6$ for different node degrees d_V and d_C.



a) Routing and logic dominated decoder **b) Influence of routing metal layer**

Fig. 3-15 Silicon area of bit-parallel decoder ($d_V = 6$, $d_C = 6$, $w = 6$, $\lambda = 40nm$)

Due to the tree structure in both node types the relation follows a logarithm dependency with base two. As only integer numbers are possible as the number of adder stages the delay increases stepwise with an increasing degree. Check-node degrees of a power of two are attractive with respect to decoder throughput. In contrast the degree of the bit node should be a power of two decremented by one, as in the multi-operand adder the channel information has to be summed-up with the d_V A-posteriori information.



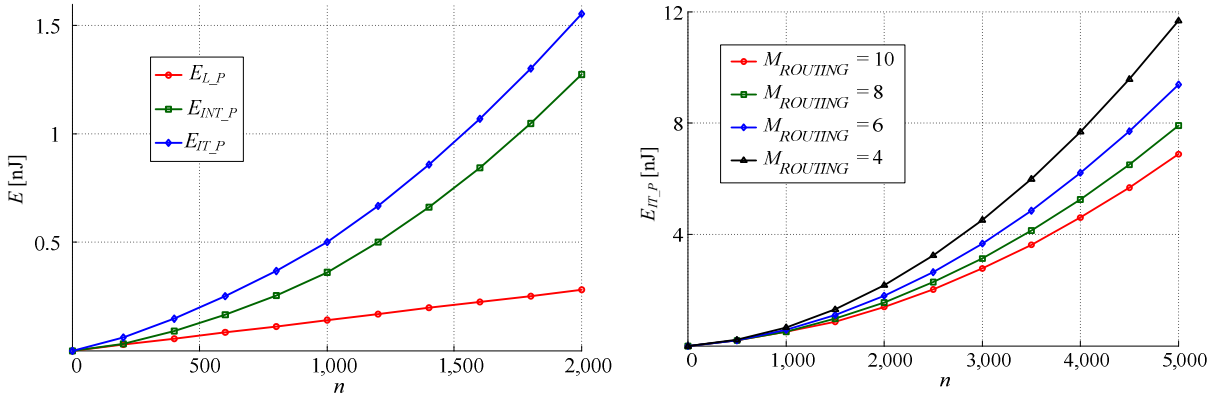
a) Critical path of node logic **b) Iteration period ($d_V = 6$, $d_C = 32$, $w = 6$, $\lambda = 40nm$)**

Fig. 3-16 Timing in bit-parallel decoder

The critical path of both nodes is almost equal for a certain node degree. However, the bit-node degree of an LDPC code is typically smaller than the one of the check-node degree. Therefore, the critical path in the check node is typically longer.

In Fig. 3-16 b) the iteration period for various code lengths and different number of routing metal layers is shown. Therein, the critical path of the logic is constant because of the constant degree of the nodes. For small block lengths the delay of the nodes dominates the

iteration period of the bit-parallel decoder. However, for larger block length the impact of the interconnect delay on the iteration period is significant. E.g. for a block length of about 2,800 and a usage of four routing metal layers the delay due to the interconnect equals the propagation delay of the nodes for this kind of decoders.



a) Energy break-down ($M_{ROUTING} = 8$) **b) Influence of number of routing layers**

Fig. 3-17 Energy per iteration of bit-parallel decoder ($d_V = 6$, $d_C = 32$, $w = 6$, $\lambda = 40$nm)

The energy per iteration for a BER of 10^{-5} and a maximum number of decoding iterations of eight is depicted in Fig. 3-17. The energy per iteration is separated into the logic and the interconnect part in Fig. 3-17 a). For larger block lengths the energy to load and unload the interconnect highly dominates the energy per iteration of the decoder. Thereby, up to 80 % of the energy is due to the interconnect for a block length of 2,000.

Furthermore, the energy per iteration scales quadratically with the block length. As illustrated in Fig. 3-17 b): the lower the number of metal layers is, the steeper is this increase.

3.3.2 Bit-serial decoder

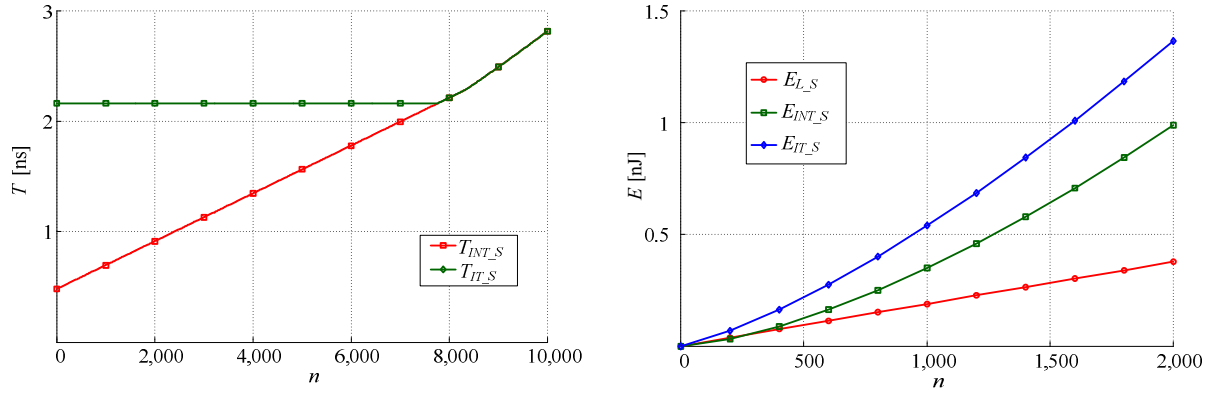
The routing area of the bit-serial decoder is significantly smaller than the one of the bit-parallel decoder. Again, the comparison of the logic area (3-38) and the routing area (3-39) reveals an upper limit of the code complexity which still leads to a logic dominated decoder. This code complexity approximately is

$$n \cdot d_V = (1700 + 150 \cdot w) \cdot M_{ROUTING}^2. \tag{3-55}$$

As the absolute value of the right hand side of the equation is significantly higher than in (3-54), only for higher code complexities the bit-serial decoder results in a routing dominated decoder. Considering e.g. a message word length of six bit and four metal routing layers, only LDPC codes with a code complexity larger than 40,000 lead to a routing dominated decoder. For bit-parallel decoders the boundary lies at about 1,300.

The estimation of the iteration period of bit-serial decoders requires an analysis of the minimal possible clock period. It is limited by the longest critical path in the decoder loop ((3-42)). As discussed for the bit-parallel decoder, the degree and, thus, critical path of the check node is larger than the one of the bit node for typical code parameters. As the interconnect delay is small for small block lengths, the check-node operation limits the clock frequency.

In Fig. 3-18 a) the minimal clock period of the check node and of the interconnect is illustrated for different block lengths. For small block lengths the interconnect delay is smaller than the critical path of the check node. As the critical path is constant for a certain check-node degree, the clock period and, thus, the iteration period do not depend on the code complexity. However, for an increasing block length the interconnect delay increases. This increase is linear as long as the decoder is logic dominated and quadratically, if the decoder needs to be expanded artificially. For block lengths larger than 8,000 the interconnect delay is larger than the critical path in the check node. Then the clock period and the iteration period are not constant but depend on the block length.



a) Maximal clock period of bit-serial decoder **b) Energy per iteration**

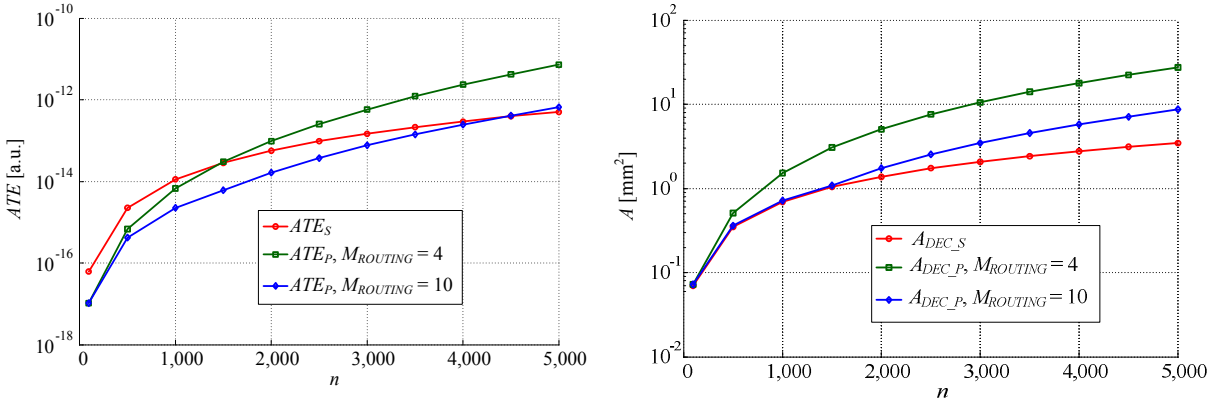
Fig. 3-18 Features of bit-serial decoder architecture ($d_V = 6$, $d_C = 32$, $M_{ROUTING} = 4$, $w = 6$, $\lambda = 40$nm)

As the interconnect complexity is reduced for bit-serial decoders, the impact on the energy per iteration should be reduced. Fig. 3-18 b) approves this assumption quantitatively. While more than 80 % of the energy per iteration is due to the global interconnect in a bit-parallel decoder ($n = 2{,}000$, $d_V = 6$, $M_{ROUTING} = 4$, $w = 6$, $\lambda = 40$nm), in a bit-serial decoder this fraction is reduced to about 70 %.

3.3.3 Comparison of decoder architectures

The derived models allow for an ATE-cost estimation in an early design phase and, therefore, enable a decoding performance and ATE-cost trade-off analysis. Exemplarily, the

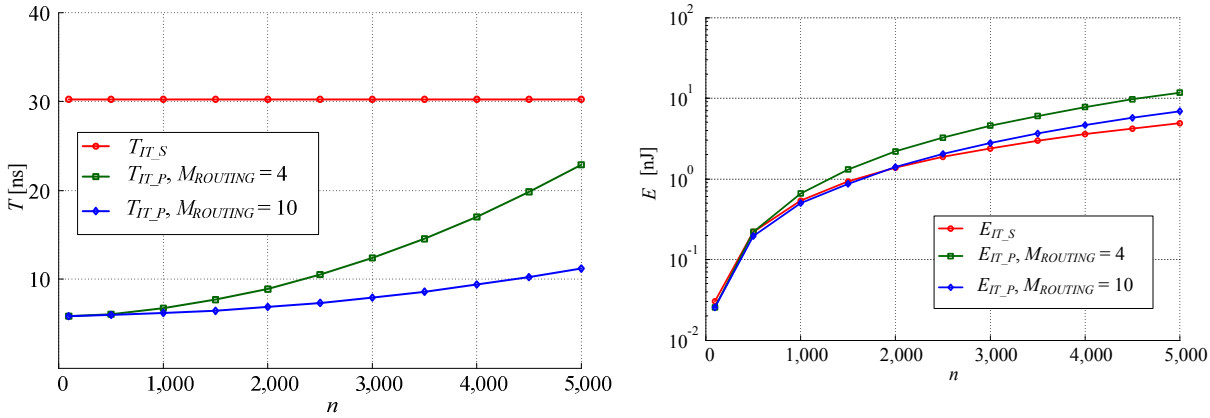
ATE-complexity for various block lengths ($d_V = 6$, $d_C = 32$) is illustrated in Fig. 3-19 a) for a bit-serial (ATE_S) and a bit-parallel decoder (ATE_P) . As the bit-serial decoder ($M_{ROUTING} \geq 4$) is logic dominated for block lengths shorter than 7,000, the number of metal layer does not affect the decoder features in the considered block length range. For a bit-parallel decoder implementation the ATE-costs are shown for four and a fictive number of ten routing metal layers.



a) *ATE*-complexity ($d_V = 6$, $d_C = 32$) **b) Silicon area ($d_V = 6$, $d_C = 32$)**

Fig. 3-19 Quantitative analysis of decoder features I

For small block lengths ($n < 1{,}500$) both bit-parallel implementations show a smaller ATE-complexity than the bit-serial decoder. This is due to the very low iteration period which is up to five times lower than for the bit-serial decoder, while the area overhead is moderate (see Fig. 3-20 a) and Fig. 3-19 b), respectively). Considering a very high number of routing layers ($M_{ROUTING} = 10$), the complexity is smaller than for the bit-serial decoder for the whole block length range, because there is almost no increase in iteration period and the area overhead is small in comparison to the bit-serial decoder.



a) Iteration period ($d_V = 6$, $d_C = 32$) **b) Energy per iteration ($d_V = 6$, $d_C = 32$)**

Fig. 3-20 Quantitative analysis of decoder features II

However, for a smaller number of routing layers the iteration period of the bit-parallel decoder increases significantly with an increasing block length. Additionally, the silicon area overhead is immense in comparison to the bit-serial architecture leading to a high ATE-complexity.

The energy per iteration for all three decoders show a similar behavior (Fig. 3-20 b)). However, while the bit-parallel decoder with ten routing layers and the bit-serial decoder almost lead to the same energy, the energy of the parallel decoder with only four routing layers is more than twice as high.

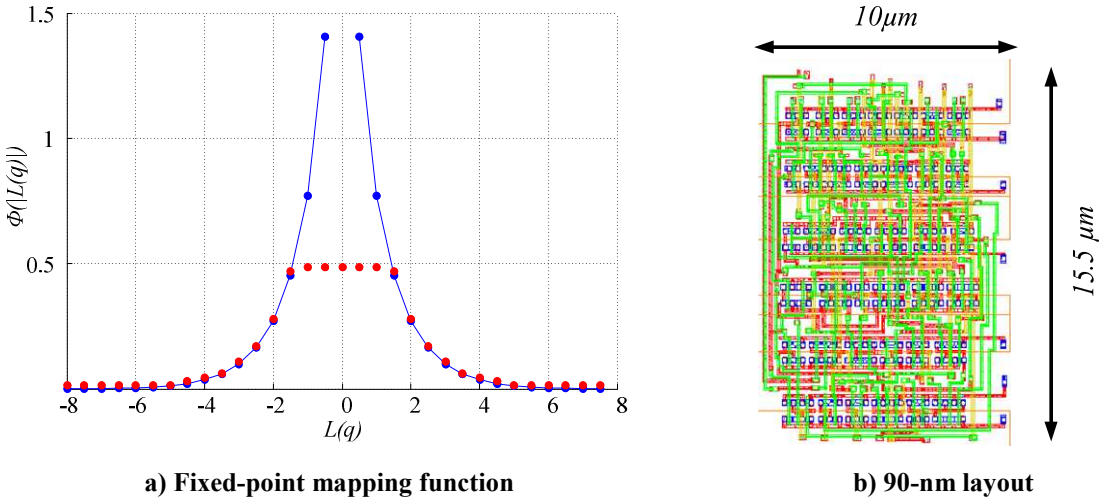
In summary bit-parallel decoder implementations are very attractive for LDPC codes with a short block length. For larger block lengths the ATE-complexity increases significantly. The slope of this increase is steeper the fewer metal layers are used to route the global interconnect. As the increase in ATE-complexity is smaller for bit-serial LDPC decoder, these become more ATE-efficient for larger block lengths. Nevertheless, high-throughput applications may require a bit-parallel decoder. Even if considering interleaved decoding of two blocks, which increases the decoding performance by a factor of two, the bit-parallel decoder ($M_{ROUTING} = 4$) would still allow for a higher throughput and especially a lower block latency even for a block length of 2,000.

3.4 Sum-Product decoder

Although the cost-models derived so far assume a Min-Sum-based LDPC decoder, the model can easily be modified to support Sum-Product-based LDPC decoders, as well. The only difference between both decoders is the calculation of the reliability in the check node as shown in Fig. 2-5.

The estimation of the features of the multi-operand adder can be taken from the estimation of the bit-node multi-operand adder. Therefore, only a new model for the Φ and the Φ^{-1} function has to be derived. As the word length of the input, as well as of the output of these functions is limited a Boolean function for each output bit in dependence on the input bits can be derived. Additionally, Boolean optimization tools such as e.g. espresso [47] can be used to optimize the Boolean function. Then the silicon area, critical path, and energy per operation can be approximated based on these results by a simple gate count.

Considering e.g. the fixed-point realization of the Φ function given in Fig. 3-21 a) the model would estimate a silicon area of 151 μm^2 and a critical path of 816 ns in a 90-nm CMOS technology. Thereby, the required conversion from two's complement to sign magnitude is included in the Boolean function.



a) Fixed-point mapping function **b) 90-nm layout**

Fig. 3-21 Realization of fixed-point *Φ* function

The Boolean function has been realized in a 90-nm CMOS technology. The layout is shown in Fig. 3-21 b) and occupies a silicon area of 155 um^2. The critical path is 750 ps (V_{DD}=0.9, slow, 110°). The comparison of the estimated and the measured features reveals a difference smaller than 10 %. Thus, the integration of this modeling scheme into the ATE-cost models allows for an accurate estimation of Sum-Product algorithm based LDPC decoders.

4 Analysis of fixed-point decoding algorithm

The design of an LDPC decoder for a certain application typically starts with the mapping of the algorithm to fixed-point arithmetic. The chosen fixed-point realization has a high impact on the hardware-complexity of LDPC decoders. Therefore, a detailed analysis of fixed-point implementations with respect to decoding performance is essential. Such an analysis has to contain e.g. an optimization of the word lengths used in the decoder. Furthermore, there are algorithm related optimizations such as the fixed-point approximation of the transcendent Φ function in Sum-Product decoders or the hardware-efficient realization of the post-processing function in Min-Sum-based decoders.

In this analysis the very low error floor of LDPC codes has to be respected. The error floor is e.g. as low as a BER of less than 10^{-10} for the LDPC code used in the 802.3an standard [48]. At this BER only every billion's block can not be decoded. This behavior impedes the analysis of the decoding performance in two ways. First of all, the analysis of the decoding performance for a certain algorithm and word length requires the simulation of more than ten billion blocks. Thus, a simulator with a high throughput is required. A developed and throughput optimized C-Model e.g. simulates the decoding of approximately 1,300 blocks per second (2.8 GHz CPU with 32 GByte RAM) leading to a simulation time of about 90 days to simulate the error floor of the IEEE 802.3an-compliant code. Second of all, the quality of the noise generator needs to be very high as it has to generate those rare error events which results in trapping sets [49]. Especially the quality of software build in noise generators is not sufficient.

To overcome these challenges a custom generic HDL simulator has been developed which will be presented in the following. As the performance of HDL simulators is lower than for C-code based simulators a hardware accelerator is used to execute the simulations.

4.1 Hardware-accelerated HDL simulator

The HDL simulator consists of four basic components: the AWGN-channel, the decoder, an analyzer, and a controller. In the AWGN-channel (chapter 4.1.1) the n noisy channel symbols for a certain SNR, a specified modulation scheme, and a certain word length w are generated and sent to the actual decoder model. As it will be described in chapter 4.1.2 the decoder model is highly parameterizable and supports the Sum-Product, as well as the Min-Sum algorithm. The n error corrected symbols are transmitted to the analyzer in which the bit and frame errors are accumulated and finally the BER and the FER is determined. The

simulator is controlled by a state-machine which synchronizes the AWGN-channel, the decoder, and the analyzer.

4.1.1 AWGN Channel

The AWGN channel generates blocks of n noisy symbols. These are generated by adding white Gaussian noise of a certain SNR to a code word. As the quality of the white Gaussian noise can highly impact the analysis results, an accurate replication of the normal distribution is crucial. Good hardware noise generators allow for a replication of the Gauss function in a range of up to $8 \cdot \sigma$ [50]. Typically, the generation of a normal-distributed variable bases on a transformation of a unified random variable into a normal-distributed variable. A possible transformations is the Box-Muller method [51]. Due to its good properties with respect to hardware realization the Box-Muller method is e.g. used in an FPGA-based hardware generator in [50] which has been the basis of the implemented generator.

The algorithm converts two uniformly distributed random numbers in the range [0,1) u_1 and u_2 into two independent normally distributed variables n_1, $n_2 \sim \mathrm{N}(0,1)$ with a mean value $\mu = 0$ and a variance $\sigma^2 = 1$.

The basic ideas of the transformation become obvious when looking at the characteristics of the two dimensional histogram of two normal-distributed random numbers n_1 and n_2. As the distribution is rotational-symmetric, the probability for an angle φ on any circle of a radius r is constant. Therefore, the normalized angle

$$u_2 = \frac{1}{2 \cdot \pi} \cdot \varphi = \frac{1}{2 \cdot \pi} \cdot \arctan\left(\frac{n_2}{n_1}\right) \tag{4-1}$$

is uniformly distributed on [0,1).

Additionally, the square of the radius of this two-dimensional distribution follows a Chi-squared distribution as the negative logarithm of a uniform-distributed variable does [51] leading to

$$-2 \cdot \log(u_1) = n_1^2 + n_2^2 = r^2. \tag{4-2}$$

Eq. (4-1) and (4-2) contain a transformation instruction to convert two uniformly distributed variable u_1 and u_2 into two standard normal-distributed variables n_1 and n_2. As n_1 and n_2 can be written as

$$n_1 = r \cdot \cos(\varphi) \text{ and } n_2 = r \cdot \sin(\varphi), \tag{4-3}$$

it directly follows

$$n_1 = \sqrt{-2 \cdot \log(u_1)} \cdot \cos(2 \cdot \pi \cdot \varphi) \tag{4-4}$$

and

$$n_2 = \sqrt{-2 \cdot log(u_1)} \cdot sin(2 \cdot \pi \cdot \varphi) \quad (4\text{-}5)$$

A block diagram of the modeled random noise generator is shown in Fig. 4-1. At the beginning two uniform-distributed numbers u_1 and u_2 are generated. To assure highly accurate normal-distributed noise samples the uniform-distributed noise samples have to be of high quality, as well. Tausworthe random number generators which base on linear feedback shift registers allow for a hardware-efficient realization of such a random number generator. However, a too simple recursion would result in a low noise quality. In contrast combined generators yield in a better stochastic characteristic and a higher sequence length.

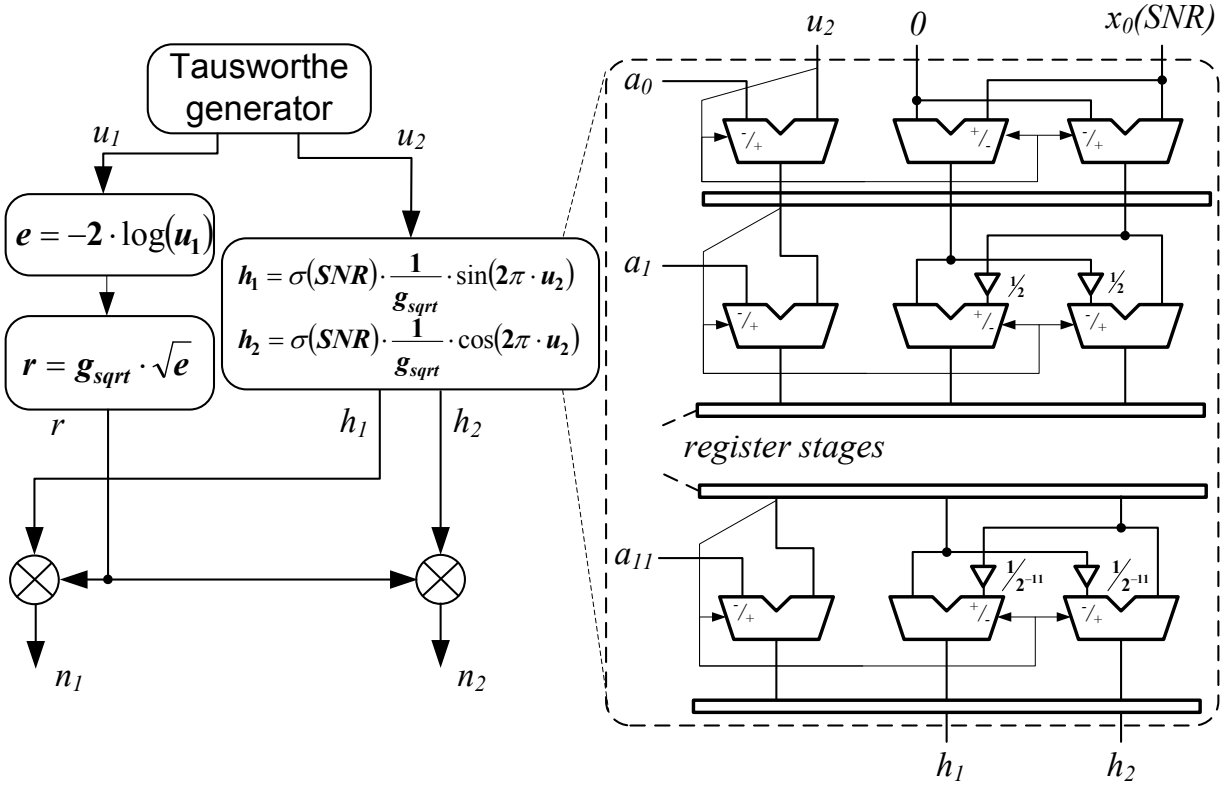


Fig. 4-1 Box-Muller-based converter

The implemented Tausworthe generator is taken from [52]. It consists of a combination of three linear feedback shift registers which are based on the following primitive polynomials

$$\begin{aligned} P_1(z) &= z^{31} - z^{13} - 1 \\ P_2(z) &= z^{29} - z^{2} - 1 \\ P_3(z) &= z^{28} - z^{3} - 1. \end{aligned} \quad (4\text{-}6)$$

The combination results in a linear feedback shift register with a maximum length of $2^{88} \approx 10^{25}$.

Subsequently, the Box-Muller algorithm is performed to transform the uniform-distributed variables u_1 and u_2 into normal-distributed variables n_1 and n_2 based on (4-4) and (4-5). The

calculation can be separated into a radius and an angle calculation. The angle calculation is a transformation of polar into Cartesian coordinates on an unit circle. The radius r is calculated in two steps. First of all, a variable e is calculated which is the negative logarithm of u_1 and which has a Chi-squared distribution. In a second step the radius r is derived by calculating the square root of e. All operations can be efficiently realized using the Cordic-algorithm. Then the originally proposed Box-Muller algorithm [51] should be modified. The Cordic-based calculation of the sine / cosine and of the square root requires a subsequent multiplication with an inverse gain factor. Instead of realizing the multipliers at the output of the corresponding blocks it is possible to use an initial value

$$\boldsymbol{x}(\boldsymbol{SNR}) = \boldsymbol{\sigma}(\boldsymbol{SNR}) \cdot \frac{\mathbf{1}}{\boldsymbol{g}_{\boldsymbol{sqrt}}} \cdot \frac{\mathbf{1}}{\boldsymbol{g}_{\sin/\cos}} \tag{4.7}$$

in the Cordic-based calculation of the sine and cosine as shown in Fig. 4-1. This factor also includes the conversion from a N(0,1) normal-distributed random variable into a N(0,σ^2) distributed variable which is done by multiplying with the standard deviation σ

$$\boldsymbol{n_i}' = \boldsymbol{n_i} \cdot \boldsymbol{\sigma}(\boldsymbol{SNR}) \tag{4.8}$$

To support a wide range of applications the noise generator is highly parameterizable. Therefore, the word length of the uniform-distributed variables w_{u1} and w_{u2}, of internal signals and of the normal-distributed variables w_{n1} and w_{n2} can be adapted.

Exemplarily, a required range of the normal distribution of ±5·σ and a binary phase shift keying modulation is considered. The maximum possible noise sample is determined by the minimal possible value of u_1. Based on (4-4) or (4-5) the minimal value can be determined to $3.7 \cdot 10^{-6}$ which results in 19 fractional bits of u_1. To avoid quantization effects a word length $w_{u1} = 20$ is chosen. The output word length of the considered Tausworthe generators is 32 bits, leading to a word length of $w_{u2} = 12$. To account for the BPSK modulation scheme the standard deviation for a certain SNR is given as [53]

$$\sigma(SNR) = \sqrt{\frac{1}{2} 10^{-\frac{SNR}{10}}} \tag{4.9}$$

For a SNR range of three to six dB the corresponding standard deviation would vary between 0.5 und 0.35. Therefore, the scaling yields in a compression of the bell curve.

To evaluate the quality of the generated noise the distribution of the generated noisy symbols are compared to the ideal normal distribution given as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{x}{\sigma} \right)^2 \right). \tag{4.10}$$

In Fig. 4-2 the ideal bell curve and the histogram of 80 million generated noise values is depicted for a standard deviation of $\sigma = 0.45$. The generated symbols consist of five integer and five fractional bits.

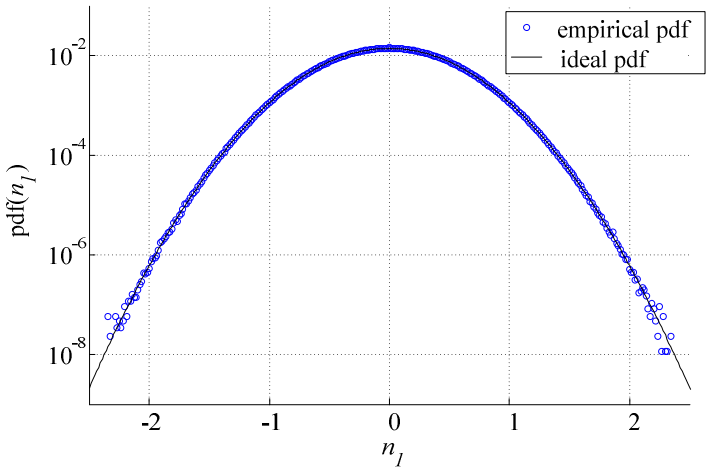


Fig. 4-2 Ideal and empirical bell curve of noise values ($\sigma = 0.45$)

An analysis of the generated to the ideal bell curve shows a deviation of just 0.1% in the range of $\pm$ *5.1* σ.

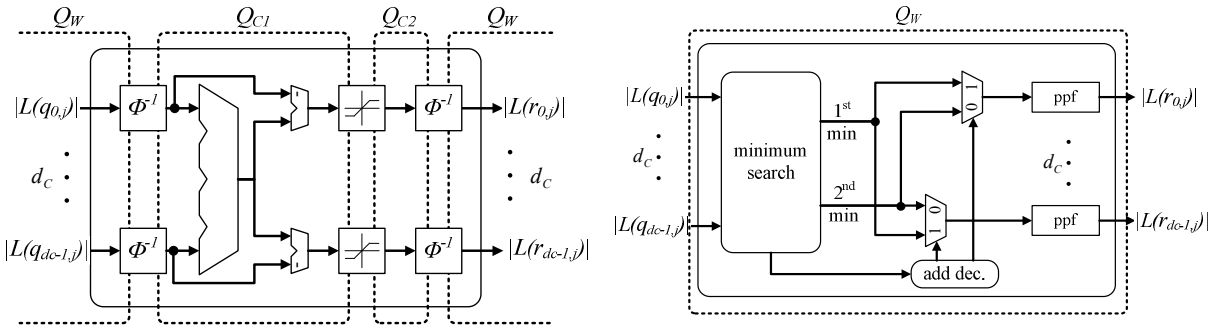
4.1.2 Decoder model

The parameterizable decoder model supports the simulation of any regular LDPC code for various decoding algorithms. Based on a definition of the parity-check matrix H in an ASCI text file the corresponding verilog HDL model is automatically derived for a specified message word length w. The modeled decoder has a bit-parallel architecture and, therefore, the model consists of the parallel instantiation of n bit- and m check-node modules connected by a bit-parallel network. In dependence on the considered algorithm either parameterizable Sum-Product or Min-Sum check nodes are instantiated.

A block diagram of the Sum-Product check node is depicted in Fig. 4-3 a). As the check node is performed in the logarithmic domain, the word length w_{CI} in the domain Q_{C1} is an additional parameter besides the message word length w. As the Φ function is transcendent, it has to be mapped to a fixed-point realization. To allow for a high flexibility the function is realized as a look-up table (LUT). A generator has been realized to generate the HDL model of the function for various w and w_{CI} combinations. The subsequent multi-operand adder and subtractor stage results in an increased word length of

$$w_{EXT\_CN} = w_{C1} + \lceil ld(d_C) \rceil . \tag{4.11}$$

The realization of the Φ^{-1} function for this extended word length would result in a complex circuit. However, due to the characteristic of the function a saturation of the values is possible without affecting the decoding performance. Therefore, the values are saturated to w_{C2} bits before the Φ^{-1} function is applied. Here, again a LUT approach is used to maintain flexibility.



a) Sum-Product check node **b) Min-Sum check node**

Fig. 4-3 Architecture of check-node models

The block diagram of the Min-Sum check node is shown in Fig. 4-3 b). As this node does not operate in the logarithmic domain, the word length is kept constant in the whole node. The post-processing in this decoder is realized as a LUT like the realization of the Φ function in the Sum-Product decoder. Therefore, it is possible to simulate any possible input-output combination as an approximation of the post-processing function.

4.1.3 Hardware-accelerated HDL simulation

Due to the complexity of the simulator, a software simulation of the described model does not allow for a sufficient simulation throughput. A simulation using Modelsim e.g. results in a throughput which is some orders of magnitudes lower than simulations using a throughput optimized C-model.

To increase the simulation throughput of the simulator the HDL accelerator ZEBU-AX [54] is used. The provided compiler maps the model on very-long-instruction-words application-specific-processors. The used configuration of the simulator supports simulations of HDL designs up to a size of 64 million gate-equivalents. The processors in this configuration allow for a processing of up to 4096 verilog/vhdl operations in one clock cycle. The throughput of the simulator for the IEEE 802.3an-compliant LDPC code for high SNRs is approximately 5 MBit/s. In comparison to software simulations of the HDL model using Modelsim on a 3 GHz CPU (32 GB memory) the throughput is increased by a factor of 1,000. The throughput improvement is still a factor of 30 compared to the throughput optimized C-

model. Additionally, in contrast to software build in noise generators the quality of the noise generator is high enabling the analysis of even very low error floor.

4.2 Decoding performance analysis of fixed-point decoders

The described hardware simulator has been used to analyze the decoding performance of the IEEE 802.3an-compliant LDPC code for the Sum-Product and the Min-Sum decoding algorithm for a wide range of decoder-parameter sets.

4.2.1 Fixed-point effects on decoding performance

The used word length of the channel information $L(c_i)$ and of the messages $L(q_{i,j})$ and $L(r_{i,j})$ communicated between the two node types highly affects the decoding performance. The achievable performance varies in particular in comparison to floating-point implementations of the decoding algorithm. The derivation of a required word length can be separated into the determination of the integer and fractional bits. Thereby, a sufficient number of integer bits are required to allow for a representation of the channel information in the whole probability range. In Fig. 4-4 the probability distribution of the channel information $L(c_i)$ for various SNR is depicted which are initialized using (2-6). Additionally, the saturation boundaries for up to six integer bits are drawn. Only six integer bits allow for a correct representation of nearly all possible channel symbols.

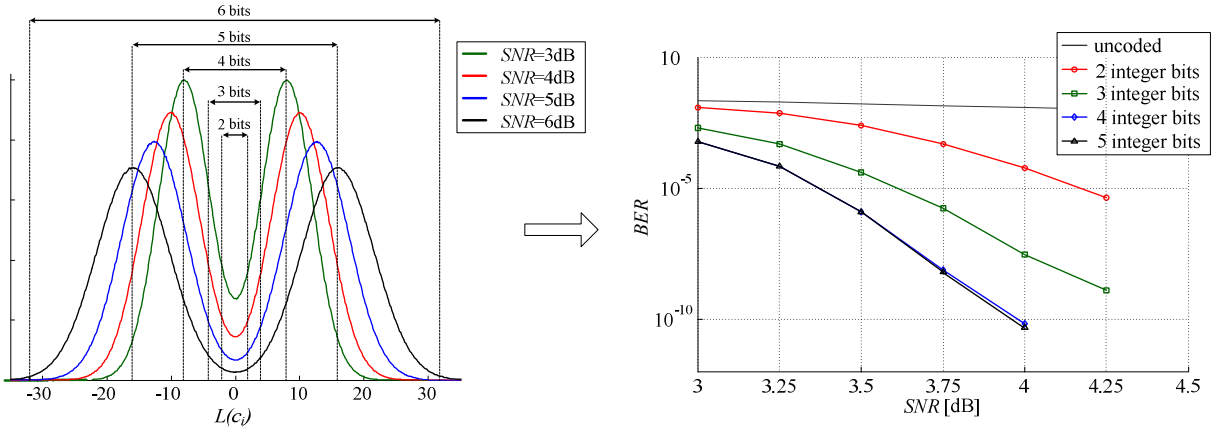


Fig. 4-4 Quantization of $L(c_i)$ and impact on Sum-Product decoding performance (10 iterations)

Considering e.g. three integer and three fractional bits, a range between [-4,3.875] can be represented using a two's-complement representation. Thus, about 80 % of the channel symbols would be saturated for an SNR of three dB. The decoding performance of such a decoder is low as it can be seen in the BER chart of Fig. 4-4. The reason is the saturation of channel symbols with a high reliability level leading to a reduction of the reliability level for

correct transmitted symbols. A decoder with only two integer bits would require an even higher SNR for the same decoding performance. While the increase to four integer bits results in a better decoding performance, an increase to five bits does not yield in a significant further improvement. To minimize the hardware complexity four integer bits should be chosen for this code.

A similar analysis can be done for the number of fractional bits with maintaining a constant number of integer bits of four. The BERs for no, one, two, and three fractional bits are shown in Fig. 4-5. As the number of fractional bits increase, the decoding performance is improved. However, the higher the number of bits, the less is the improvement per additional bit. E.g. the difference of the decoding performance for two and three fractional bits is only about 0.07 dB for a BER of 10^{-9}.

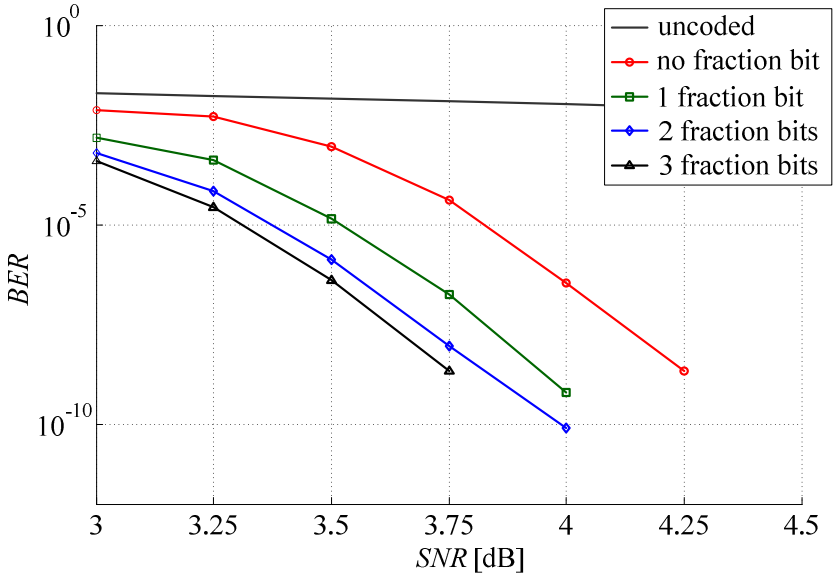


Fig. 4-5 Analysis of fractional bit impact on Min-Sum algorithm decoding performance (4 integer bits, 10 iterations)

4.2.2 Fixed-point Sum-Product decoder

A rough estimation of the required quantization in the logarithmic domain of a Sum-Product decoder can be done by analyzing the characteristics of the Φ function. By performing a density evolution of a floating-point Sum-Product decoder the range of operation for the Φ and Φ^{-1} function can be determined. The density distribution of the input of the Φ function $|L(q_{i,j})|$ is shown in Fig. 4-6.

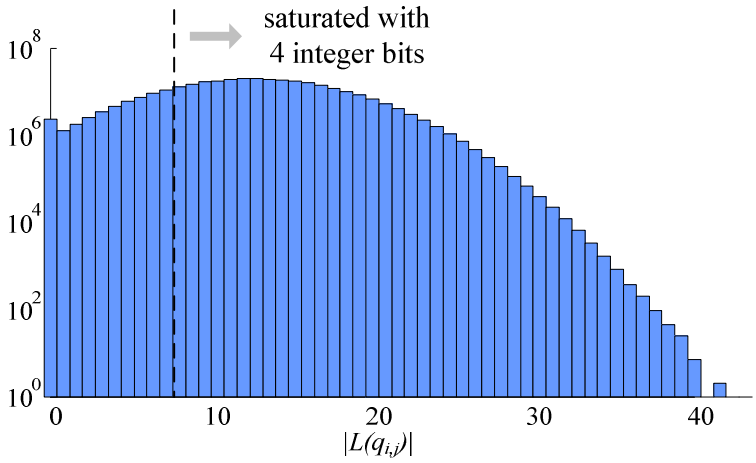
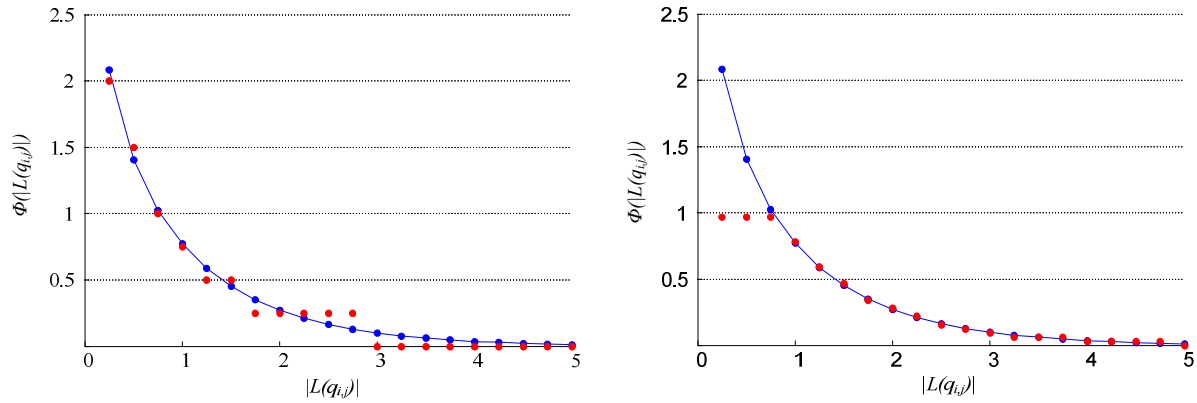


Fig. 4-6 Density of $|L(q_{i,j})|$ in first decoding iteration

Considering the same quantization scheme for the bit-node messages as for the channel symbols (four integer and two fractional bits), 84 % of the values are saturated. Additionally, in the unsaturated region larger magnitudes are more probable as smaller ones. Therefore, an accurate fixed-point implementation of the Φ function is necessary especially for large input values.

A first approach is the mapping of $|L(q_{i,j})|$ values to the output value corresponding to the Φ function. Fig. 4-7 shows the accurate output of the Φ function for the possible input values between 0 and 7.75 in blue. As the output is also quantized using w_{CI} bits, the accurate output values need to be mapped to possible values in the quantization domain. One possible mapping of the accurate to the fixed-point values is rounding.



a) 3 integer and 2 fractional bits of $\Phi(|L(q_{i,j})|)$ **b) 0 integer and 5 fractional bits of $\Phi(|L(q_{i,j})|)$**

Fig. 4-7 Fixed-point implementation Φ function

Exemplarily, the derived quantized output using three integer and two fractional bits and zero integer and five fractional bits is shown in Fig. 4-7 a) and b), respectively. While the decrease of integer bits on the one hand reduces the accuracy for small input values, an

increase of fractional bits allows for a more detailed modeling of the function for larger input magnitudes.

The resulting decoding performance for the two different fixed-point implementation of Fig. 4-7 and two additional realizations is shown in Fig. 4-8. As expected, the accurate reproduction of the function for large input magnitudes with four or five fractional bits results in the best decoding performance.

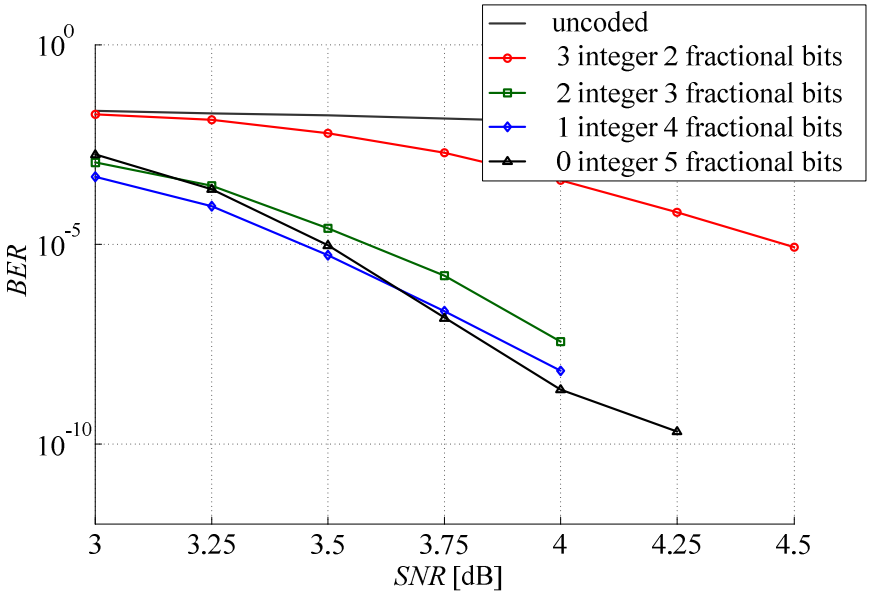
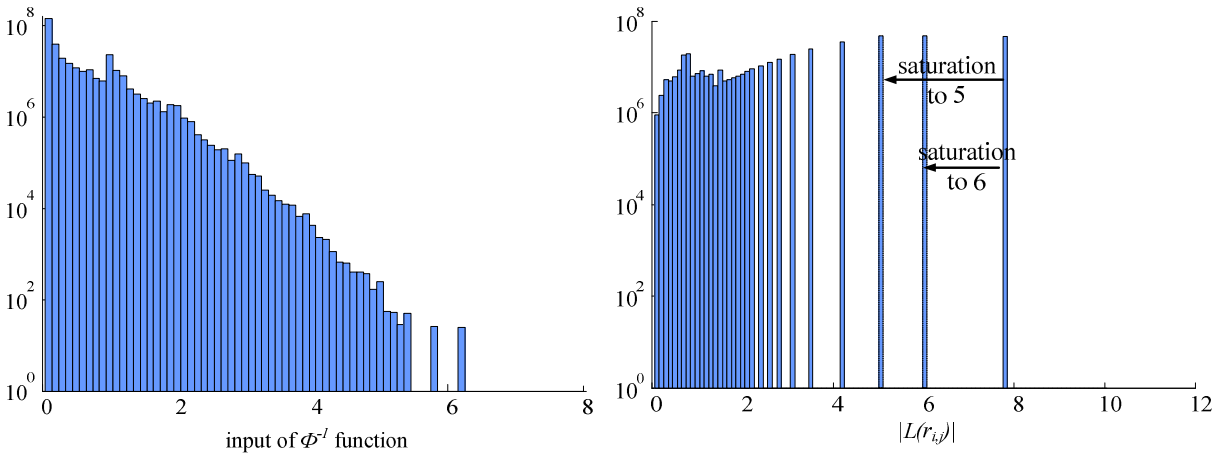


Fig. 4-8 Decoding performance of fixed-point implementation of Φ function (10 iterations)

In an analogue way a mapping for the Φ^{-1} function can be derived. The optimal quantization of this function depends on the quantization scheme chosen for the Φ function. In Fig. 4-9 a) the histogram of the input values is illustrated considering an output quantization of the Φ function of zero integer and five fractional bits.

Input values larger than three would be mapped to zero as the minimum magnitude of the check-node messages is 0.25. Due to the multi-operand adder and subtractors, '0' is possible as an input of the Φ^{-1} function. As the non-quantized output would be infinite, the input value has to be mapped to a certain high magnitude, e.g. the maximum representable value (here 7.75). On the other hand the value can also be saturated to a certain value as shown in Fig. 4-9 b).



a) Input of Φ^{-1} function **b) $|L(r_{i,j})|$**

Fig. 4-9 Histograms of quantized Sum-Product check node

The effect of the saturation value on the decoding performance is illustrated in Fig. 4-10. Therein, the decoding performance for a saturation to 7.75, 6, and 5 is depicted.

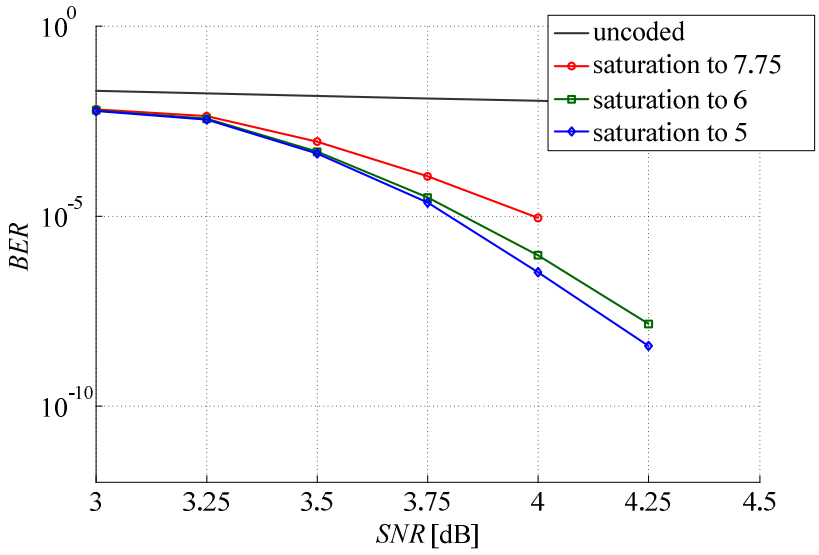


Fig. 4-10 Decoding performance of fixed-point implementation of Φ^{-1} function (10 iterations)

The decrease of the saturation value increases the decoding performance. The best decoding performance is obtained using a saturation value of five. A further reduction of the value would reduce the decoding performance. In such a case the sum of all A-posteriori information is small in comparison to the channel information which would impede a proper error correction.

4.2.3 Approximate Sum-Product decoder

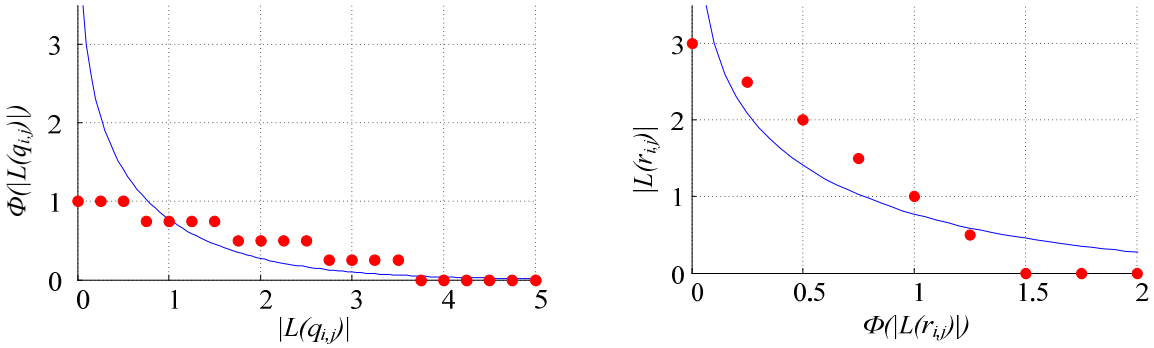
As the fixed-point realization of the Φ and Φ^{-1} functions in fact is nothing but an approximation of the accurate functions, it is also possible to find other hardware-efficient

approximations which model the distribution of the $|L(r_{i,j})|$ values. By gently modifying the mapping between the input and output values the hardware complexity can be reduced. Nevertheless, during these modifications a reduction of the decoding performance has to be avoided.

On the other hand a mapping function which allows for an efficient implementation e.g. by using FAs would be favorable in terms of hardware complexity. By changing the mapping to the functions shown in Fig. 4-11 a realization using FAs is possible. The corresponding functions are

$$\Phi'(x) = 1 - \frac{1}{4}x \quad \text{and} \quad \Phi'^{-1}(x) = 3 - 2x \,. \tag{4.12}$$

Both functions require only one adder stage as the coefficients are in the form 2^i. To avoid negative values the input values have to be saturated. Therefore, the existing saturation stages at the output of the bit node and between the subtractor stage and the Φ^{-1} function in the check node can be used leading to no additional hardware effort. In comparison to the check nodes in 4.2.1 the approximate check node allows for a hardware reduction.



a) Φ function **b) Φ^{-1} function**

Fig. 4-11 Linear approximation

In comparison to the quantized Sum-Product decoder the maximal value of $L(r_{i,j})$ is smaller. However, the decoding performance of this decoder is even better than the best decoding performance of chapter 4.2.2 (Fig. 4-12). The error floor of this fixed-point realization is comparable to the ones observed in [48].

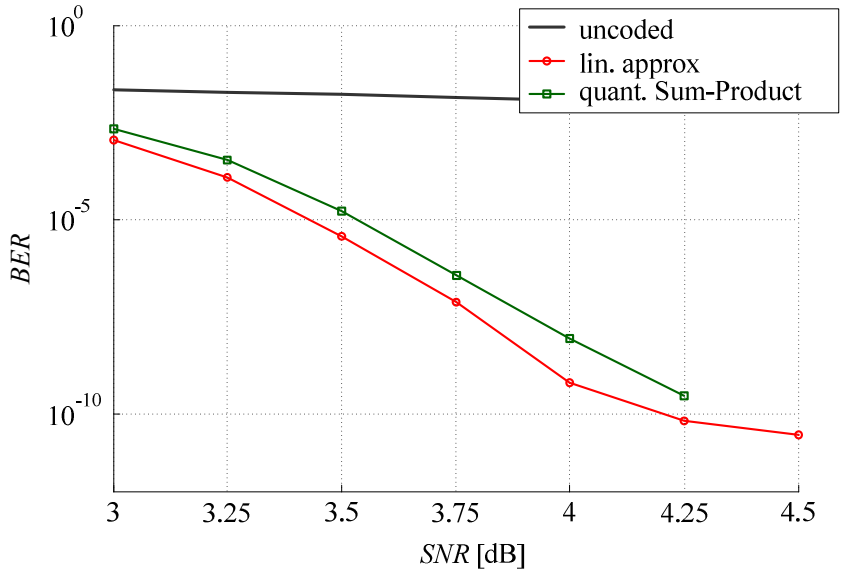
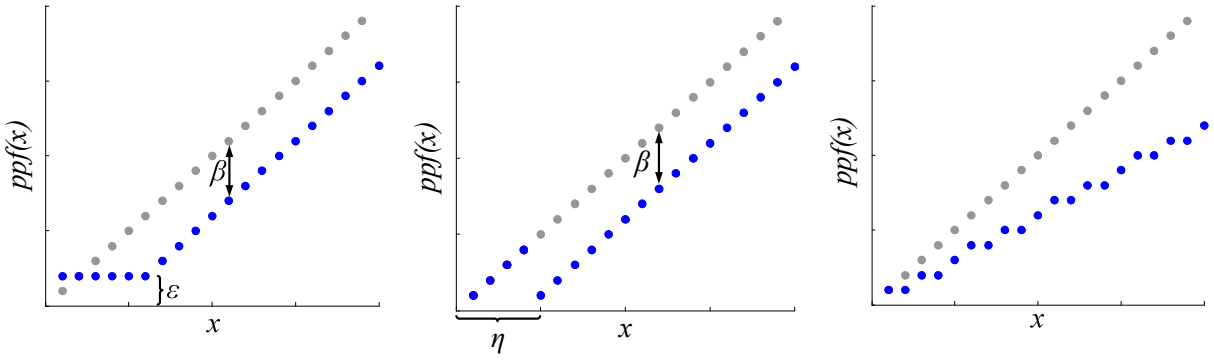


Fig. 4-12 Decoding performance of linear approximation of Φ and Φ^{-1} function (8 iterations)

4.2.4 Fixed-point Min-Sum decoder

As discussed in chapter 2.2 the Min-Sum algorithm is an approximation of the original Sum-Product algorithm. Due to the high magnitudes of the check-node messages in this algorithm, the decoding performance is decreased. Therefore, it is possible to increase the decoding performance of Min-Sum-based decoder by applying a post-processing function to these values.

Mainly two types of post-processing functions are known. The first type reduces the messages by subtraction and the other type is reduction by normalization. For the first type a constant value is subtracted from the A-posteriori information. Without any further modification this could lead to negative values for small input values. Therefore, this subtraction is performed only for values which are larger than a certain boundary. For smaller values it is possible to apply a certain constant output value (Fig. 4-13 a)) or keep the input value (Fig. 4-13 b)). For the latter post-processing function an optimal boundary and optimal value which is subtracted in dependency of the node degree is derived analytically in [55]. However, in this kind of post-processing function the cardinality is not kept. This can e.g. lead to a post-processed minimum which is larger than the post-processed second minimum.



a) Subtraction I **b) Subtraction II** **c) Normalization**

Fig. 4-13 Post-Processing Types

The second type of post-processing function reduces the minimum values by normalization. The optimal normalization factor for a certain code can be derived analytically using EXIT-charts [56]. E.g. for the IEEE 802.3an-compliant LDPC code this normalization factor is 0.6. The mapping function for this factor is depicted in Fig. 4-13 c).

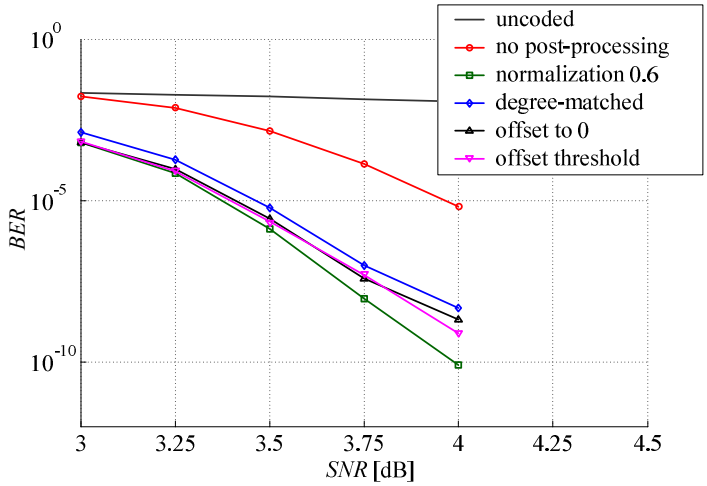


Fig. 4-14 Decoding performance of post-processing types (10 iterations)

Fig. 4-14 compares the decoding performance for the three different types of post-processing functions. Thereby, two post-processing functions of type Subtraction I with different threshold values ε ($\varepsilon = 0$ and $\varepsilon = 5$ LSBs, $\beta = 3$ LSBs) are analyzed. Additionally, the decoding performance using a fixed-point post-processing function of type Subtraction II with the parameters ($\eta = 5$ LSBs, $\beta = 3$ LSBs) derived based on [55] is depicted. In comparison to these post-processing functions the normalization results in an optimal decoding performance. The difference to other post-processing functions varies between 0.13 to 0.21 dB considering a BER of 10^{-8}.

All considered post-processing functions increase the check-node complexity. For the subtraction types at least a comparator and a subtractor need to be implemented while the

realization of the normalization requires multiple adder stages. The impact of the post-processing function on the node complexity is reduced, if the normalization can be performed in a hardware-efficient way [57]. In digital circuits normalization factors of 0.25, 0.375, 0.5, 0.625, and 0.75 are favorable, as they require at most one adder stage. In Fig. 4-15 the decoding performances for these factors are plotted in comparison to the analytical derived factor 0.6.

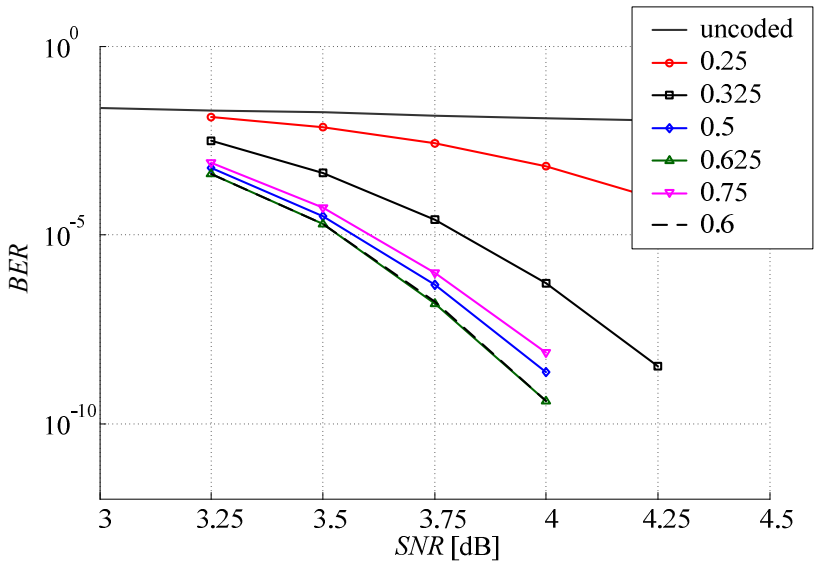
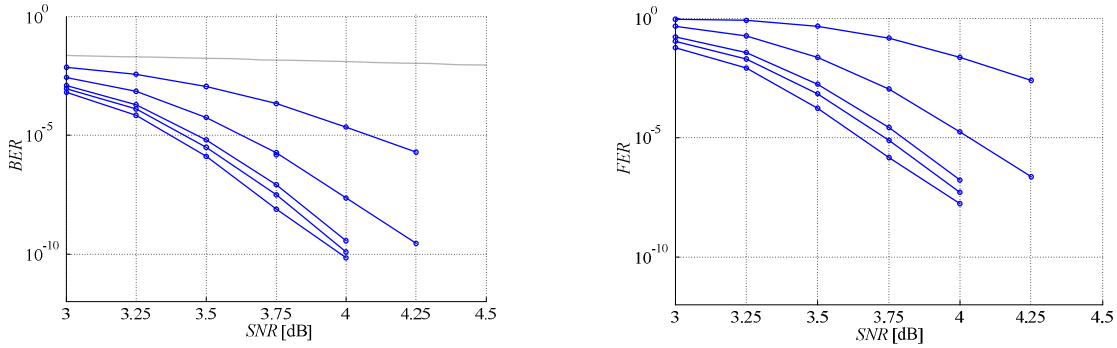


Fig. 4-15 Hardware-efficient post-processing factor

As expected, the simulations show an optimal decoding performance for a normalization factor of 0.6. When comparing the performance for the factor 0.6 and 0.5, it can be seen that the decoding performance decreases by less than 0.1 db. However, the implementation of the post-processing factor of 0.5 results in no hardware overhead. Fig. 4-16 shows the achievable BER and FER of this hardware-efficient normalization factor in more detail.



a) BER **b) FER**

Fig. 4-16 Decoding performance of normalization factor 0.5 after 2, 4, 6, 8, and 10 iterations

Typically, the post-processing function is interpreted as a correction of the large A-posteriori values in a Min-Sum-based decoder. Therefore, it is realized at the output of the

check nodes (e.g. [16]). This suits especially well because the post-processing function typically requires additional hardware and it can be realized efficiently when applying the function not to the outputs of the check node but to the minimum and second minimum (see initial position of the post-processing block in Fig. 4-17).

In contrast to other post-processing functions the realization of a normalization factor of 0.5 is only a shift and, thus, requires no additional hardware. Therefore, there is no advantage when applying it to the two minima. Furthermore, this normalization factor yields in a reduction of hardware complexity, as it reduces the message word length to *w-1*. To maximize the benefit of this property the normalization should be applied as soon as possible in one decoding iteration [57]. Therefore, the post-processing block can be moved to the output of the bit nodes as illustrated in Fig. 4-17.

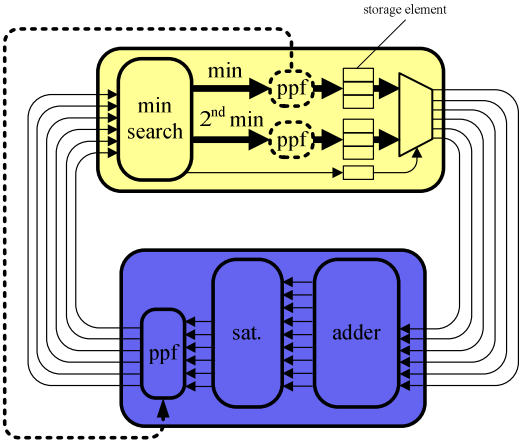


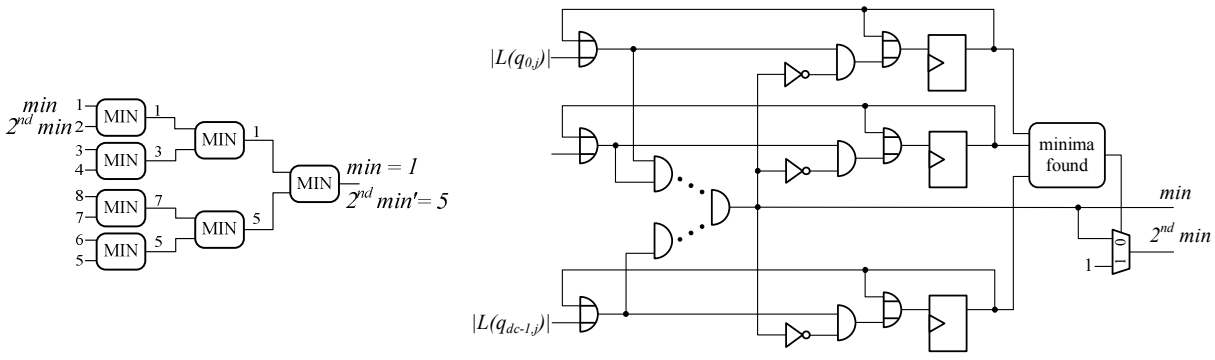
Fig. 4-17 Optimal position of hardware-efficient post-processing function

In a bit-parallel decoder implementation a normalization factor of 0.5 would result in a reduced bit- and check-node complexity and a reduced interconnect complexity. Especially the reduction of the interconnect is promising as based on (3-17) this reduces the silicon area significantly. For a IEEE 802.3an-compliant decoder with a word length of $w = 6$ the silicon area is reduced by about 27 % using this post-processing factor. For bit-serial decoder this normalization factor results in a smaller iteration period as the number of clock cycles per iteration can be reduced. Considering a word length of $w = 6$ the reduction of the iteration period is about 15 %.

4.2.5 Approximate Min-Sum decoder

The Min-Sum decoder requires the calculation of the minimal and the second minimal magnitude. To reduce the decoder complexity it is possible to calculate the minimal magnitude and then approximate the second minimum based on the results of the minimum

search. E.g. in [15] the approximation of the second minimum is done by increasing the minimum value by one LSB. Whenever the absolute and the second minimum are nearly equal, the approximation error is insignificant. Otherwise, the error is large and since the second minimum typically corrects errors, an inadequate approximation results in a significant reduction of the decoding performance. However, more sophisticated approximations are possible. E.g. if the minimum is determined using a tree of compare-and-swap elements as shown in Fig. 4-18 a), a possible approximation is the use of the maximum in the last stage of the tree, as an approximation for the second minimum. If the minimum and the second minimum origin from other parts of the tree this approximation is equal to the correct second minimal value. Otherwise, the second minimum is the absolute minimum of the $L(q_{i,j})$ values of the other part of the tree like in the example in Fig. 4-18 a). In this case the approximated second minimum is larger than the correct second minimum.

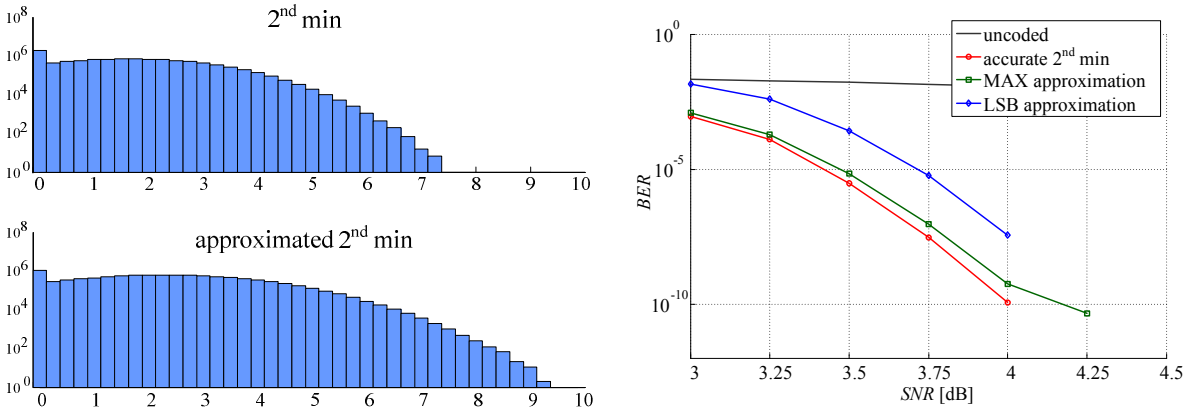


a) MAX approximation **b) LSB approximation**

Fig. 4-18 Second-minimum approximations

The second approximation targets the minimum-search architecture which is used in [15] and depicted in Fig. 4-18 b). In this bit-serial minimum search all input signals are combined by a tree of AND-gates. Based on a comparison of the resulting minimum bit and each input bit it can be calculated whether the corresponding input can still be the minimum. If it cannot be the minimum anymore, a status signal of this input is set to one. This status signal then disables the corresponding input. If all but one status signal is one, the minimum input signal is found.

The second minimum is set equal to the minimum as long as the minimum is not found. Beginning from that bit weight which finally defines which input signal is the minimum the remaining bits of the second minimum are set to one. Therefore, additional logic is required to determine whether the minimum input is found yet. A histogram of the resulting second minima approximations in comparison to the real second minima is shown in Fig. 4-19 a).



a) Second minimum of LSB approximation **b) Decoding performance (30 iterations)**

Fig. 4-19 Analysis of decoding performance for second-minimum approximation

In Fig. 4-19 b) the decoding performances of the two proposed approximations and the minimum search without approximation are shown. The LSB approximation results in a significant reduction of the decoding performance. In contrast the approximation which uses the maximum of the last compare-and-swap stage results in almost the same decoding performance. However, for the approximation an error floor is noticeable in Fig. 4-19 b) which starts at 10^{-9}.

Using the ATE-cost models derived in chapter 3 the impact of these approximations on the decoder complexity can be quantitatively analyzed. As the bit-parallel decoder is highly dominated by the interconnect, the achievable reduction is small. However, for the bit-serial decoder the silicon area, as well as the iteration period can be reduced significantly. Considering for example the IEEE 802.3an-compliant LDPC code and a message word length of $w = 6$ the silicon area of a bit-serial decoder can be reduced by 12 % using the MAX approximation. The LSB approximation would reduce the silicon area even by 17 %. As the iteration period of a bit-serial decoder is dominated by the critical path of the check node, these approximations also allow for a reduction of the iteration period. For both approximations the resulting critical path of the check node is smaller than that of the bit node. Thereby, the iteration period is reduced by 35 %.

5 Hardware-efficient decoder architectures

The next step in designing an LDPC decoder after the derivation of an efficient fixed-point realization of the algorithm is the optimization of the decoder architecture. Two popular decoder architectures have been already discussed in chapter 3. The quantitative comparison of the decoder features (Fig. 3-19) showed that the bit-parallel architecture is more efficient for small code complexities while the bit-serial architecture features the smaller ATE-complexity for large code complexities. In general, the bit-parallel architecture suffers from a high silicon area which is mainly due to the high global-interconnect complexity. On the other hand, the bit-serial approach leads to a low decoder throughput.

In the following two architectural concepts will be presented to overcome the drawbacks of the bit-parallel and the bit-serial decoder on architectural level. Therefore, instead of realizing bit- and check nodes fine granular modules are used in the decoder macro resulting in a reduced silicon area when considering a bit-parallel communication between the nodes. Furthermore, a systematic architecture analysis for the bit- and check-node architecture is performed resulting in a new partially bit-serial architecture with a significant improvement in decoder throughput.

5.1 Area-efficient bit-parallel decoder architecture

The highest possible decoder throughput is achieved when realizing a message-, word- and bit-parallel LDPC decoder like the one in [5]. However, the required silicon area is very high due to the complex interconnect between the bit- and the check nodes. Although an optimization of the node placement inside the macro is performed, the decoder in [5] still shows a utilization of the silicon area of just 50 %. Therefore, a sole optimization on circuit level does not seem to be sufficient to overcome this drawback.

One idea to reduce this drawback is to get away from the typical bit- and check-node architecture which is given by the Tanner graph, and find a new clustering of the logic. When looking at the parity-check matrix H, a one entry defines the smallest reasonable operation in an LDPC decoder. Therefore, instead of implementing one check node per row and one bit node per column one so called hybrid cell (HC) can be realized for each one entry of the matrix. Such a cell contains part of the bit- and part of the check-node logic. The logic of one check node e.g. is distributed to d_C hybrid cells as shown in Fig. 5-1. Therefore, the hybrid cells of one row need to be connected. Furthermore, the connected d_V hybrid cells of one column form one bit node.

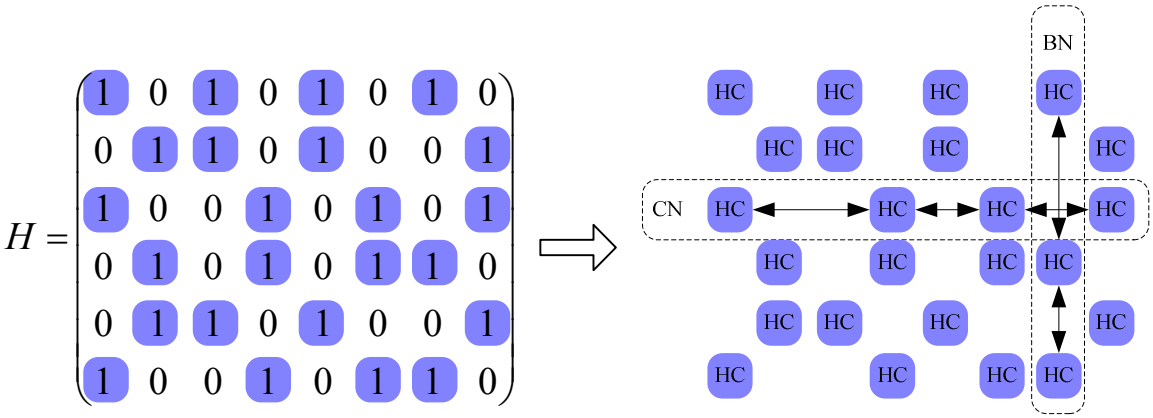
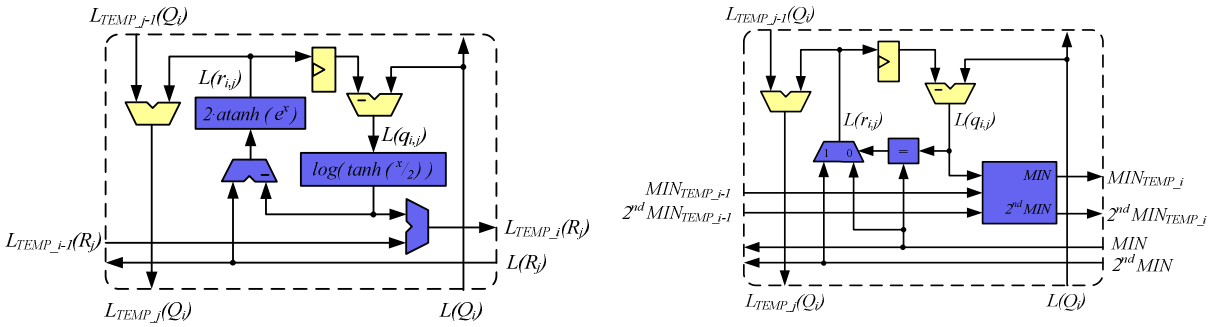


Fig. 5-1 Hybrid-cell-based decoder architecture

Fig. 5-2 a) illustrates the internal structure of a hybrid cell which bases on the Sum-Product algorithm. For the sake of clarity only the blocks of the magnitude calculation are depicted. The $L(Q_i)$ values are distributed via broadcasting wires to all hybrid cells of column i. In the HC the new A-priori information $L(q_{i,j})$ is calculated by subtracting the A-posteriori information $L(r_{i,j})$. Subsequently, the Φ function is invoiced. The multi-operand adder to calculate $L(R_j)$ is distributed over the HCs of one matrix row. Therefore, in each HC the actual value is added to the subtotal $L_{TEMP\_i-1}(R_j)$ which is received from the neighbored HC. Then the sum is sent to the next HC. At the end of the HC chain $L(R_j)$, as the sum of all $\Phi(L(q_{i,j}))$ values, is calculated and broadcasted to all d_C HCs. In the cells the required subtraction and the reconversion from the log-domain is performed to calculate the A-posteriori information $L(r_{i,j})$. The resulting value is stored in a register for the next decoding iteration and is added to the subtotal $L_{TEMP\_j-1}(Q_i)$ which is received from the neighbored HC from a column perspective. The sum is sent to the next HC. Considering that the channel information $L(c_i)$ is the input of the first HC of one column the last cell of that column calculates the $L(Q_i)$ information which is broadcasted to all HC of that column.



a) Sum-Product based **b) Min-Sum based**

Fig. 5-2 Structure of hybrid cell

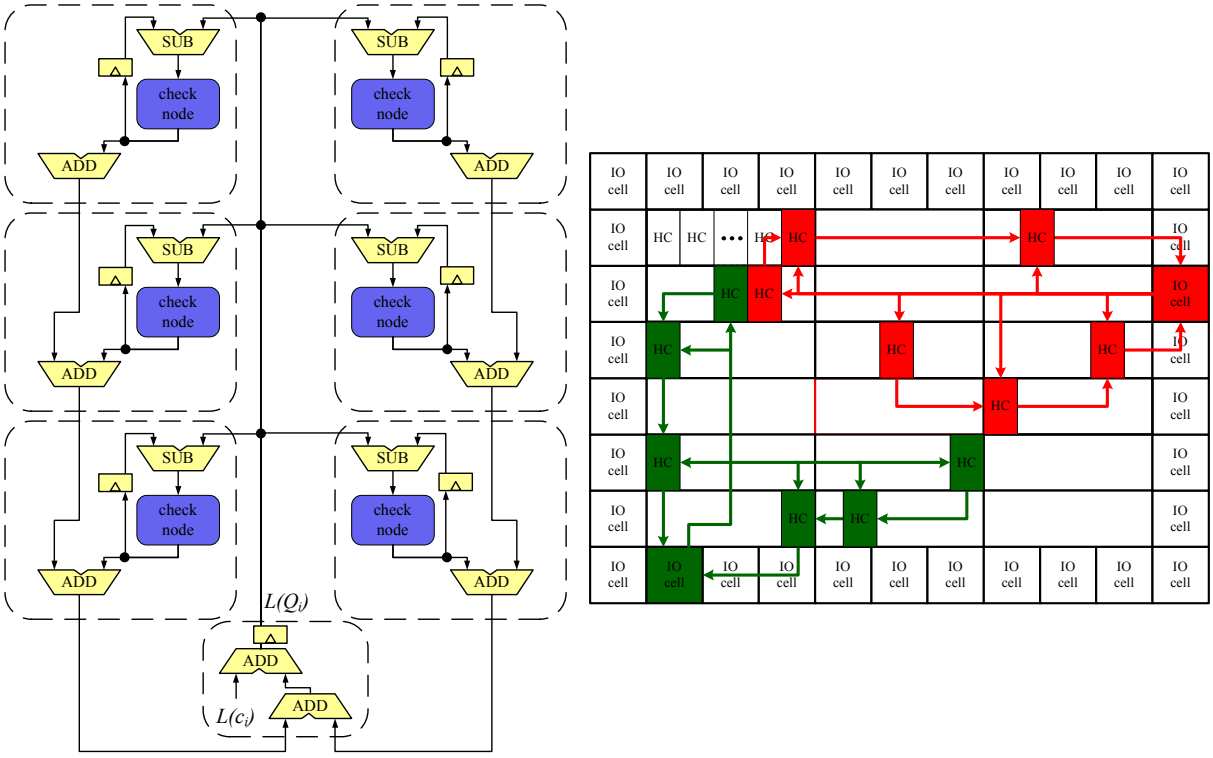
The same decoder structure can be applied to a Min-Sum based decoder. The internal structure of a HC for this algorithm is depicted in Fig. 5-2 b). Instead of converting the $L(q_{i,j})$ value into the log-domain, the value is used to compare the magnitude of the actual cell with the values of the previous cells in that row. Thereby, only the absolute and the second minimum have to be regarded. The resulting minimum and second minimum is sent to the next cell. After the minimum and the second minimum of all d_C $L(q_{i,j})$ values have been calculated and broadcasted to all connected cells, a comparison of the local $L(q_{i,j})$ signal with the minimum is performed. If these values are equal, and, thus, the local $L(q_{i,j})$ value is the minimal value of the d_C cells, the second minimum is chosen as the new $L(r_{i,j})$. Otherwise, $L(r_{i,j})$ is equal to the minimum.

The complete global interconnect consists of $2 \cdot n \cdot (d_C - 1)$ horizontal and $2 \cdot m \cdot (d_V - 1)$ vertical connections. Initially, the required number of interconnects have been doubled in comparison to the bit- and check-node architecture. Nevertheless, it is possible to merge some of the nodes into blocks, for example all HC of one row of the parity-check matrix leading to an extended check node. Therefore, the multi-operand adder of one column of the matrix is distributed over d_V extended check nodes and the order of the adders can be chosen freely.

Cascading the two-operand adders would result in a long critical path. But, it is also possible to use a tree adder structure to decrease the critical path of the multi-operand adder. Fig. 5-3 a) exemplarily shows the adder structure when the multi-operand adder contains $ts_V = 1$ tree stage for a bit-node degree $d_V = 6$. The sum of these two subtotals is summed in an additional cell which also adds the channel information $L(c_i)$. Thus, it is the io-cell of the decoder. Furthermore, the critical path of the decoder can be further reduced, when using a tree structure in the check-node operation, as well. Here, ts_C defines the number of tree stages used for calculating $L(R_j)$ or the minimum and second minimum, respectively.

In the decoder macro the hybrid cells are placed in the center of the macro like the check nodes in the standard architecture surrounded by the io-cells as shown in Fig. 5-3 b) for a small sample code. Additionally, the connections of two distributed bit nodes are outlined. Each io-cell sends the $L(Q_i)$ value to the six connected HCs. Then, beginning from two HCs, there are two tracks back to the io-cell in which the A-posteriori information are accumulated. The word length of the communicated messages can be optimized for each connection in the distributed bit node separately. E.g. the word length of the message sent from the first check node in each branch is smaller than the word length sent from the last check node to the io-cell. In the following the word length of the communicated messages is chosen to equal the

decoding performance of a bit- and check-node decoder with a message word length of five bit.



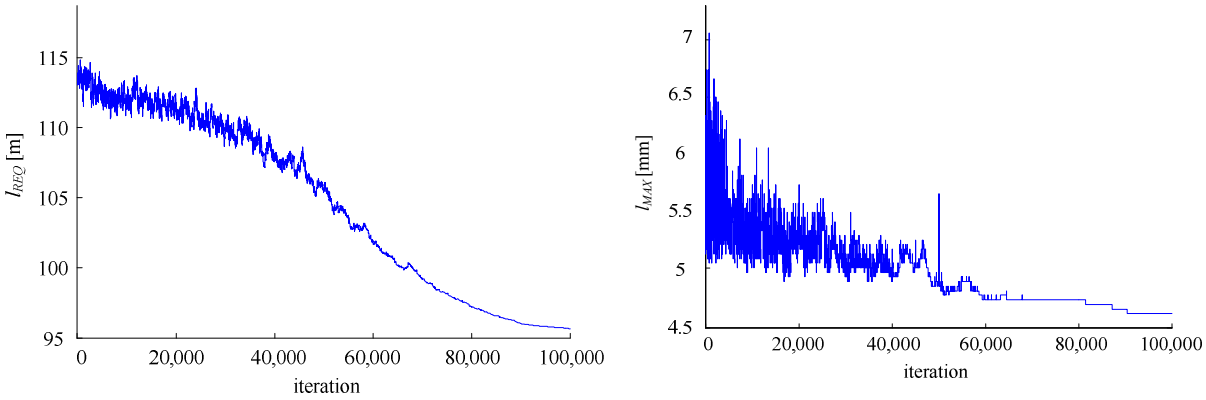
a) Distributed bit node **b) Decoder macro with bit-node connections**

Fig. 5-3 Hybrid-cell-based decoder structure

The total Manhattan length of the hybrid-cell decoder depends on the actual position of the hybrid- and io-cells as the Manhattan length in a bit- and check-node decoder does. Here, the simulated annealing algorithm discussed in chapter 3.1.2 can be used again. The minimization of the Manhattan length reduces the complexity of the interconnect and, thus, the decoder area and the energy per decoded bit. In contrast the iteration period is determined by the Manhattan length of the longest connection or in the case of the hybrid-cell architecture, the sum of the connections required to establish one half of the distributed bit node. The simultaneous minimization of the routing complexity and of the impact on the decoder throughput requires a combined cost function which e.g. can be the weighted sum of the total Manhattan length and the Manhattan length of the longest connection

$$c = \omega_1 \cdot \sum_{\forall connections} length + \omega_2 \cdot \max_{\forall connections} length \,. \tag{5-1}$$

The optimization of the maximal Manhattan length affects a small number of interconnects which only have a minor impact on the total Manhattan length. Therefore, the combination of both cost functions is unproblematic. The progress of both cost functions depicted in Fig. 5-4 a) and b) are e.g. taken from a simulated annealing algorithm with a combined cost function.

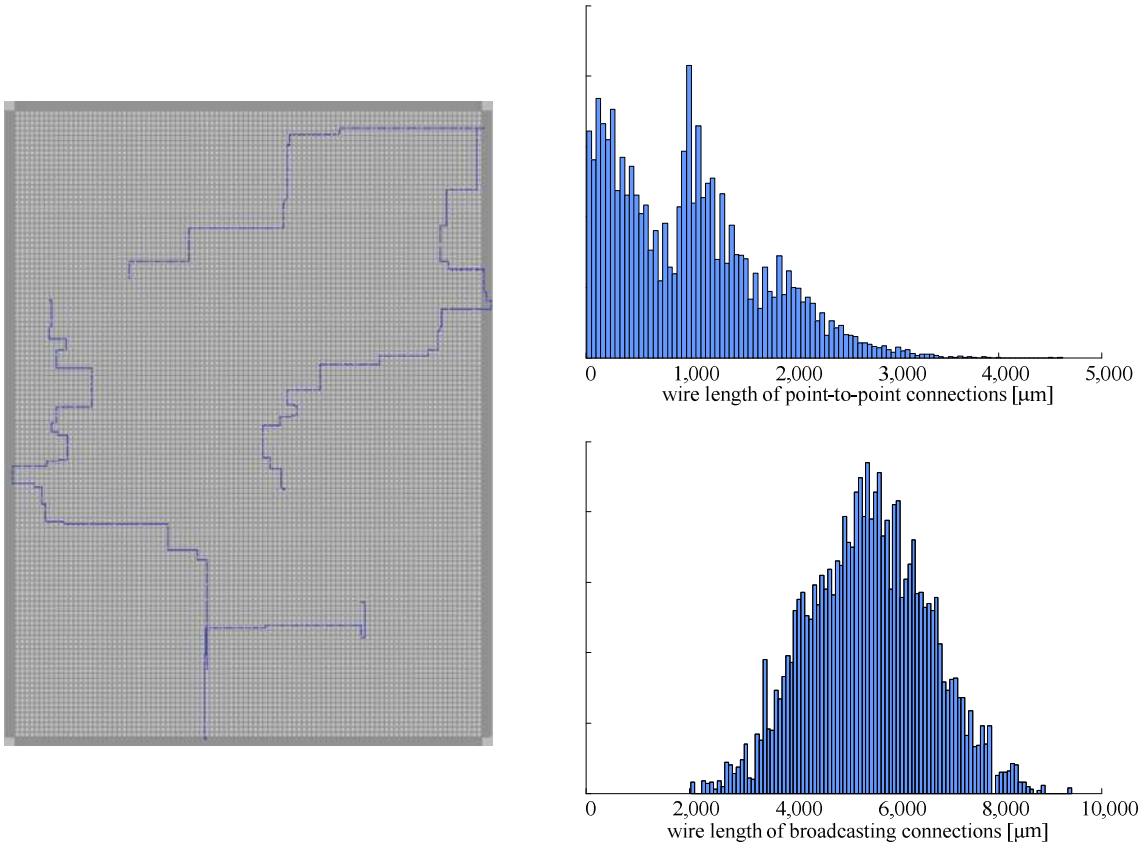


a) Summed Manhattan length of all connections **b) Longest Manhattan length**

Fig. 5-4 Cost function for hybrid-cell architecture during simulated annealing algorithm

To show the advantages of the hybrid-cell architecture in comparison to the standard bit- and check-node architecture with respect to silicon area the placement of the nodes for the IEEE 802.3an-compliant code has been optimized using the described simulated annealing algorithm. In contrast to the average of ten random placements the interconnect length has been reduced from 123 to 105 m. Then the global interconnect has been realized in a 90-nm CMOS technology using five metal layers. Therefore, node abstracts have been connected using the IC-Craftsman [58] which is integrated in the cadence design framework. To determine the minimum silicon area of the decoder the size of the node abstracts has been shrunk until the router could not establish all the required interconnect lines.

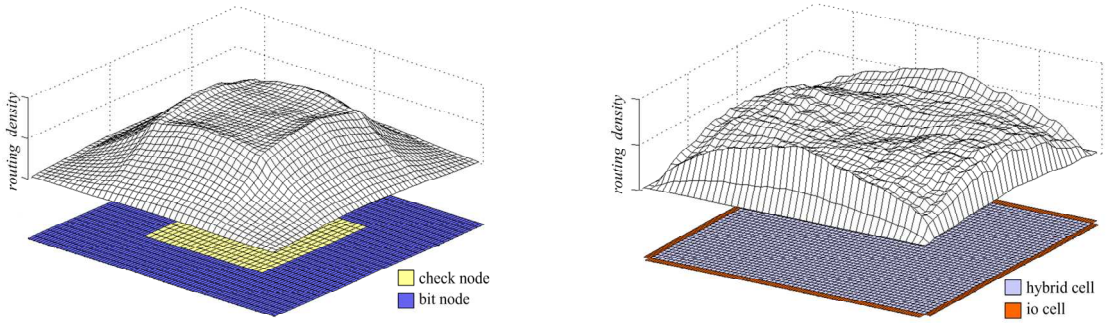
The interconnect has been successfully routed on a decoder size of just 8.5 mm^2. The connections of two sample bit-node connections and the post-routing histogram of the point-to-point connections and of the broadcasting wires are shown in Fig. 5-5 a) and b), respectively. The resulting interconnect length is 79 m. The difference to the estimated Manhattan length during the simulated annealing algorithm is due to the optimized routing of the broadcasting signals. In the cost-function of the simulated annealing algorithm these broadcasting signals have been modeled as point-to-point connections for the sake of simplicity.



a) Sample connection in decoder macro **b) Post-route wire-length histograms**

Fig. 5-5 Post-routing interconnect analysis of hybrid-cell decoder

The advantage of the hybrid-cell architecture becomes obvious when comparing the interconnect density of the hybrid-cell architecture to the one of the bit- and check-node architecture (Fig. 5-6). An analysis of the routing density for the bit- and check-node architecture shows a high density at the boundary between the check-node array and the surrounding bit nodes. The perimeter of that boundary and the number of metal layers define the minimum size of the check-node array which typically limits the silicon area. In contrast the equivalent boundary lies nearly at the perimeter of the whole decoder macro for the hybrid-cell architecture, as the io-cells are small in comparison to the bit nodes. Therefore, the routing density is distributed more uniformly leading to a higher utilization of the routing layers which is $u = 0.6$ for the realized global interconnect. In a bit-and check-node architecture the utilization for the same code is approximately only 0.4 (Tab. 2-1).

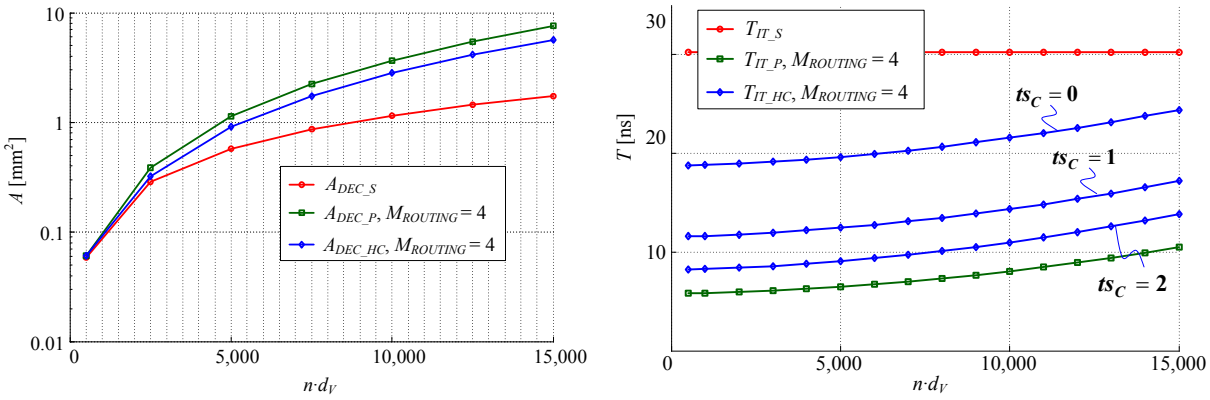


a) Bit- and check-node architecture **b) Hybrid-cell architecture**

Fig. 5-6 Interconnect density

For a quantitative comparison the area and timing models of the bit-parallel decoder architecture from chapter 3.1 have been adapted to the HC architecture. Therein, for the new architecture a utilization of $u = 0.6$ and a worst-case scenario for the highest possible interconnect delay is assumed. The resulting silicon area A_{DEC_HC} and iteration period T_{IT_HC} are compared to the basic bit-serial and bit-parallel architectures in Fig. 5-7. The silicon area is reduced in comparison to the bit-parallel architecture while maintaining a low iteration period. As the iteration period is highly affected by the cascaded HCs for the check node operation the number of tree stages ts_C should be increased.

As the bit-serial decoder architecture features a significantly lower silicon area, the HC architecture is attractive for application demanding a decoder throughput which can not be met using a bit-serial approach. In this case, the silicon area can be reduced in comparison to the bit-parallel architecture. Considering for example a code complexity $n \cdot d_V = 12{,}000$ the cost model predicts a reduction of the silicon area by approximately 25 %.



a) Silicon area **b) Iteration period**

Fig. 5-7 Quantitative comparison of hybrid-cell architecture with bit- and check-node architectures ($d_V = 6$, $d_C = 32$, $M_{ROUTING} = 4$, $w = 6$, ts_V=0, $\lambda = 40$nm)

In [59] a fully parallel realization of the IEEE 802.3an-compliant decoder ($n \cdot d_V = 12.288$) requires a silicon area of 34.9 mm^2 (quadratically scaled to 90-nm). Thereby, a message word length of five bit and a technology with six metal layers is used. The number of routing metal layers used for the hybrid-cell decoder is higher than the one used in [59]. As the number of metal layers which are used for the local connections are not stated it is assumed that three of the six metal layers are utilized for this purpose. Considering that the routing area scales quadratically with the number of metal layers ((3-17)) the normalized area would be 12.6 mm^2. Therefore, using the hybrid-cell architecture the silicon area can be reduced by more than 30 %.

However, the silicon area of a hybrid-cell decoder is still significantly larger than that of the bit-serial architecture (see Fig. 5-7 a)). As the reduced area is mainly due to the bit-serial communication between the nodes, decoder architectures with such a bit-serial interconnect are analyzed in more detail in the following subchapter.

5.2 High-throughput partially bit-serial decoder architecture

Bit-serial decoders gain from a quadratic reduction of the silicon area and suffer only from a linear increase of the iteration period in comparison to their bit-parallel counterparts. Therefore, they feature the smallest ATE-complexity in a wide code complexity range. Although it is shown in [15] that even the high throughput specified in the IEEE 802.3an standard can be met, the achievable timing features are limited. The high throughput in [15] is for example achieved by introducing block interleaving leading to a high throughput, but the block latency of such a decoder is still high. This complicates the meeting of tight latency specifications of certain standards. Moreover, the supported number of decoding iterations is limited. Although the gain of an additional iteration gets smaller the higher the number of iteration is, it might be attractive to perform 20 iterations as the error floor is reduced significantly in comparison to ten iterations as it is shown in [48].

The two decoder architectures described in chapter 3 only form a part of the bit- and check-node based decoder design space. Additionally, it is possible to introduce a bit-serial data flow into the decoding loop carefully. The loop can be split into four blocks: the communication from bit to check node, the arithmetic in the check node, the communication to the bit node, and the arithmetic in the bit node. Each of these blocks can be realized either bit-parallel or bit-serial leading to a total of 16 possible decoder architectures in which the two architectures of chapter 3 build the two extremes [41].

As the realization of a bit-parallel interconnect results in a high area overhead only those architectures with a bit-serial interconnect are considered in the following. The tree in Fig. 5-8 shows the four remaining combinations. Additionally, the number of clock cycles as a first order metric for the iteration period is quoted.

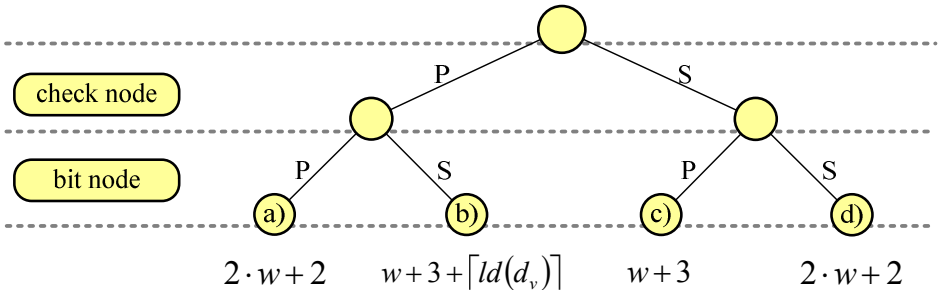


Fig. 5-8 Decoder architectures with bit-serial interconnect

In chapter 3 it has been shown that a bit-serial decoder with both nodes implemented bit-serially (architecture d)) requires *(2·w+2)* clock cycles. A block diagram of such a decoder loop and the corresponding timing diagram are depicted in Fig. 5-9 d).

Realizing both nodes bit-parallel (architecture a)) also leads to an iteration period of *(2·w+2)* clock cycles as is illustrated in Fig. 5-9 a). At the beginning of the decoding iteration the bit-node messages $L(q_{i,j})$ are sent bit-serially to the check node where they are stored in a serial-to-parallel converter. In clock cycle six the check-node operation is performed bit-parallel and the results are sent to the bit node bit-serially. In the last clock cycle the bit-node operation is performed bit-parallel.

Beside these two architectures there are also two mixed architectures realizing only one of the nodes in a bit-serial and the other in a bit-parallel fashion. The decoder loops for these two architectures are depicted in Fig. 5-9 b) and c). The architecture of Fig. 5-9 b) consists of a bit-parallel check node and a bit-serial bit node. Due to the bit-serial LSB-first data flow in the bit node, both node messages are sent bit-serially LSB-first using a two's-complement representation. Beginning with clock cycle one, the multi-operand adder in the bit node calculates one bit of the bit-node messages $L(q_{i,j})$ in each clock cycle. After clock cycle seven all check-node message bits are processed and sent back to the check node. Because of carries to higher bit weights the word length needs to be extended. Due to the LSB-first data flow in the bit node, a saturation of the bit-node messages is not possible. Therefore, the unsaturated messages have to be sent to the check node. Based on (3-1) this word length depends on the degree of the bit node d_V leading to a total of $\left(w + \lceil ld(d_V) \rceil + 3\right)$ clock cycles.

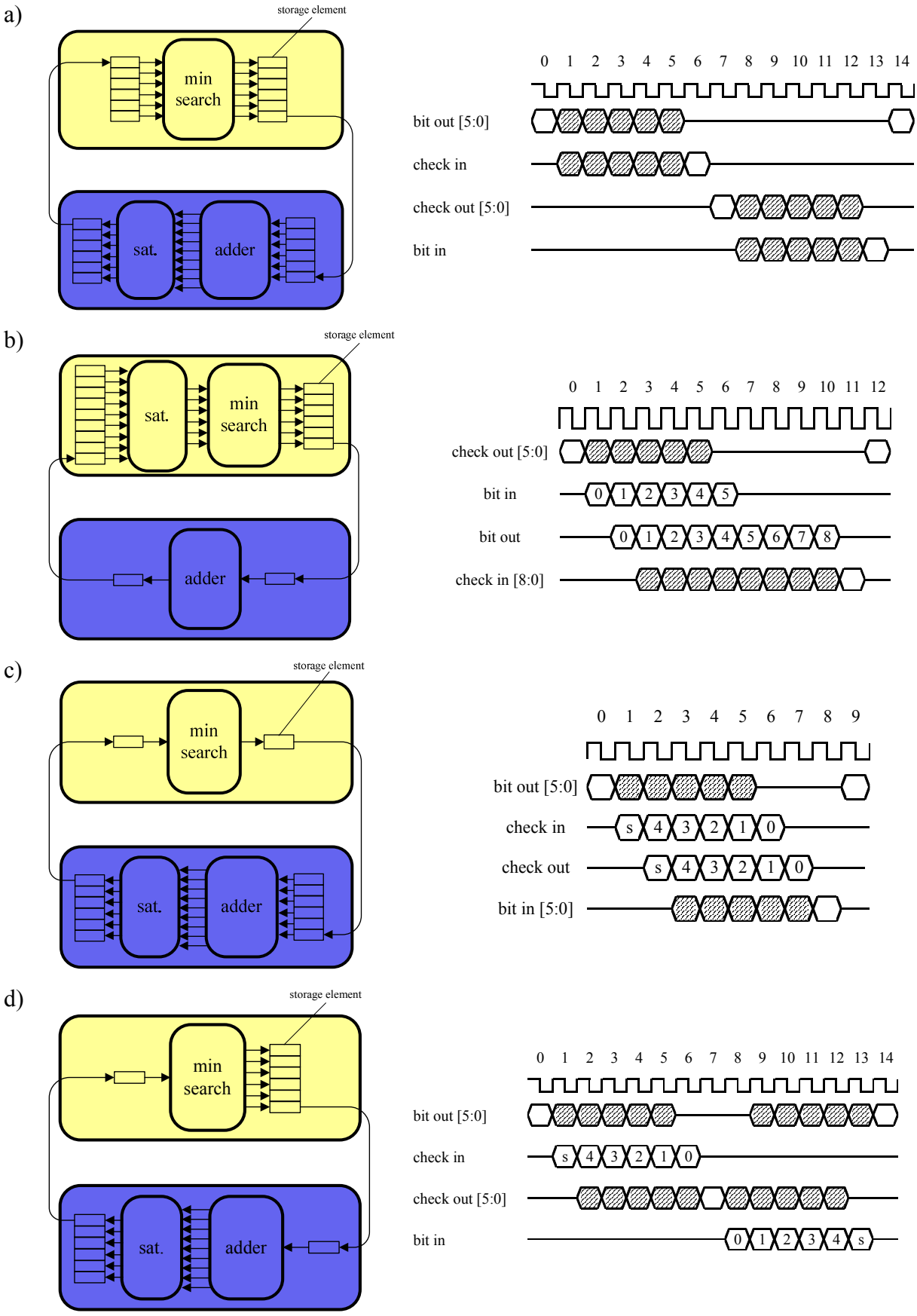


Fig. 5-9 Timing of decoder architectures with bit-serial interconnect

Architecture c) consists of a bit-parallel bit- and a bit-serial check node. Due to the bit-serial MSB-first minimum search in the check node, both node messages are sent bit-serially using a MSB-first data flow and a sign-magnitude representation. At the beginning of the iteration the bit-node messages are stored in the output registers of the bit node. Subsequently, the messages are sent bit-serially to the check node where they are immediately processed. The resulting bit is sent back to the bit node where the bits are accumulated in an input registers. As no extension of the word length is required in the check nodes, only w bits are sent back to the bit node. In total $(w+3)$ clock cycles are required for one decoding iteration.

In comparison to architectures a) and d) the mixed architectures allow for a smaller number of clock cycles per iteration. Thereby, architecture c) realizes the lowest number of cycles. However, the mixed architectures obviously suffer from a longer critical path in the nodes. This is due to the bit-parallel realization of one of the two nodes leading to a ripple path in either the minimum search (architecture b)) or the multi-operand adder (architecture c)). The critical paths of both nodes in architecture d) do not include this ripple path over the word length. Thus, this architecture allows for a higher clock frequency and additionally a smaller silicon area of the nodes. However, as will be shown in the following by applying some arithmetic optimizations to the nodes of architecture c), it is possible to reduce the silicon area and the critical path significantly.

5.2.1 Arithmetic optimization of nodes

Architecture c) in Fig. 5-8 requires a bit-serial realization of the check node. The incoming bit-serial messages in sign-magnitude representation are stored in input registers. In clock cycle one of the iteration the signs of the check-node messages are calculated. This is done using XOR-gates as it has been described in chapter 2.2. In the other clock cycles the minimum search of the bit-node messages is performed. The multiplexers at the output of the check node select the output of the sign calculation in clock cycle one and in the following clock cycles the output of the bit-serial minimum search. A block diagram of such a bit-serial check node is depicted in Fig. 5-10.

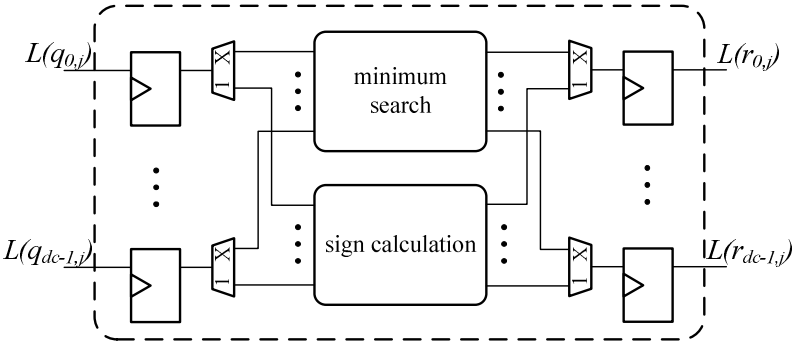


Fig. 5-10 MSB-first check node

In contrast to other bit-serial implementations as e.g. [15] the minimum search has to calculate the bits of the d_C check-node messages immediately. A possible architecture consisting of two-operand minimum cells is shown in Fig. 5-11 for a check-node degree $d_C = 32$. The two-operand minimum cells can be derived from the bit-serial compare-and-swap cell illustrated in Fig. 3-3 b) by removing the maximum output circuit. The minimum search consists of eight stages. In the first four stages the minimum of the upper 16 and the lower 16 input values are calculated in a tree structure. Therefore, the output signals of the first stage consist of the minima of two neighboring inputs $min_{0\text{-}1} = min(|L(q_{0,j})|, |L(q_{1,j})|)$, $min_{2\text{-}3} = min(|L(q_{2,j})|, |L(q_{3,j})|)$ … $min_{30\text{-}31} = min(|L(q_{30,j})|, |L(q_{31,j})|)$. In the second stage the minima of four inputs $min_{0\text{-}4} = min(|L(q_{0,j})|, |L(q_{1,j})|, |L(q_{2,j})|, |L(q_{3,j})|)$, … $min_{28\text{-}31} = min(|L(q_{28,j})|, |L(q_{29,j})|, |L(q_{30,j})|, |L(q_{31,j})|)$ are calculated etc.

In the second part of the minimum search the intermediate minima are combined using minimum cells to calculate the outputs $L(r_{i,j})$. The magnitude of the check-node message $L(r_{31,j})$ is e.g.

$$\left| L\left(r_{j,31}\right) \right| = \boldsymbol{min}_{0-30} = \boldsymbol{min}\left(\underbrace{\boldsymbol{min}_{0-15}}_{tree}, \underbrace{\boldsymbol{min}_{16-23}, \boldsymbol{min}_{24-27}, \boldsymbol{min}_{28-29}, \left| L\left(q_{j,30}\right) \right|}_{inverse\ tree} \right). \tag{5-2}$$

Thereby, internal minima for groups of inputs are reused to minimize the number of minimum cells. For a check-node degree $d_C = 32$ in total 90 two-operand minimum cells are required.

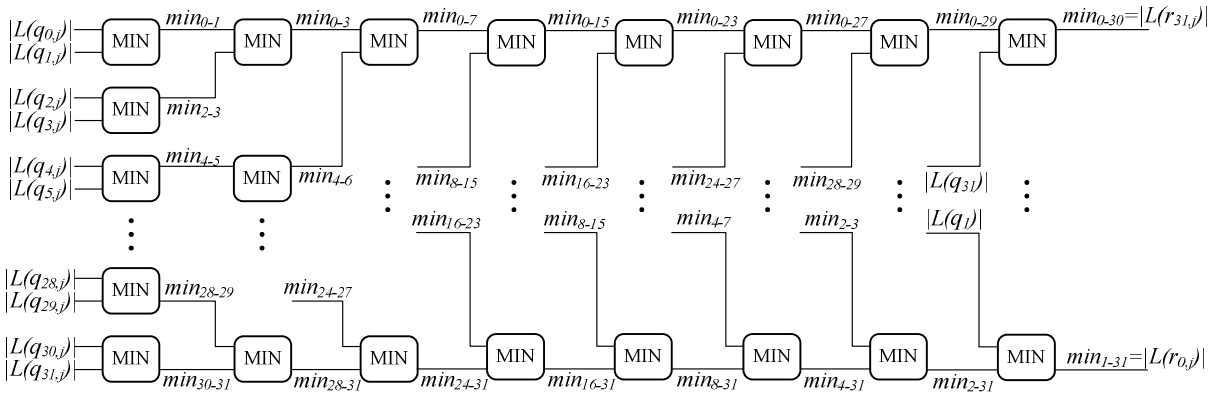


Fig. 5-11 MSB-first minimum search 1

To reduce the hardware complexity the smallest and the second smallest input value is typically searched with a subsequent choice of one of the two values. If the output signal has to be sent immediately to the bit node, it is not possible to wait for the end of the comparison. A circuit which performs this immediate output is shown in Fig. 5-12 for a check-node degree $d_C = 8$. The search for the minimum and second minimum is done using the circuit of Fig. 3-3 b). Additionally, for each output a multiplexer which either selects the minimum or the second minimum has to be implemented for each output.

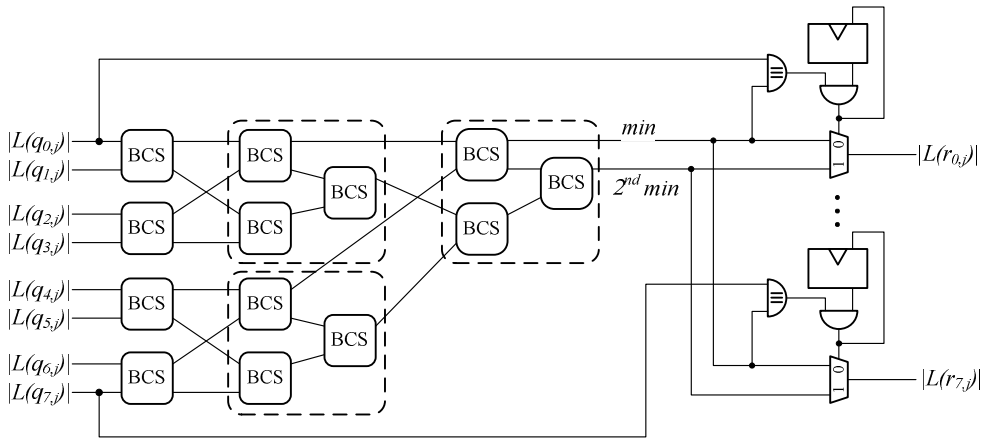


Fig. 5-12 MSB-first minimum search 2

At the beginning of a comparison the registers in the multiplexer logic are initialized to *'1'*. The signal remains on its high level, as long as the corresponding input possibly is the minimum. In this case the second minimum is selected. If for the first time the input value differs from the minimum, the circuit consisting of an XNOR- and an AND-gate sets the signal to *'0'*. Subsequently the minimum would be selected as the output.

The number of bit compare-and-swap cells is given in (3-5). These cells are slightly more complex than a minimum cell which is used in Fig. 5-11. Additionally, d_C multiplexer cells are required. Therefore, the hardware complexity of both architectures is comparable.

A first approach to realize the bit-parallel bit node would be to use the bit-parallel node discussed in chapter 3.1. However, this would lead to a large silicon area, as well as a long critical path with a ripple path over the extended word length. However, it is possible to exploit the fact, that the check-node messages are received in a MSB-first fashion. The basic idea directly becomes clear when looking at the way how to sum-up e.g. seven operands in two's-complement representation manually as done in Fig. 5-13 a). Usually, we start at the LSB and calculate the partial sum which is three in the example. In the next step the partial sum of the next bit position is calculated and added to the partial sum of the LSB.

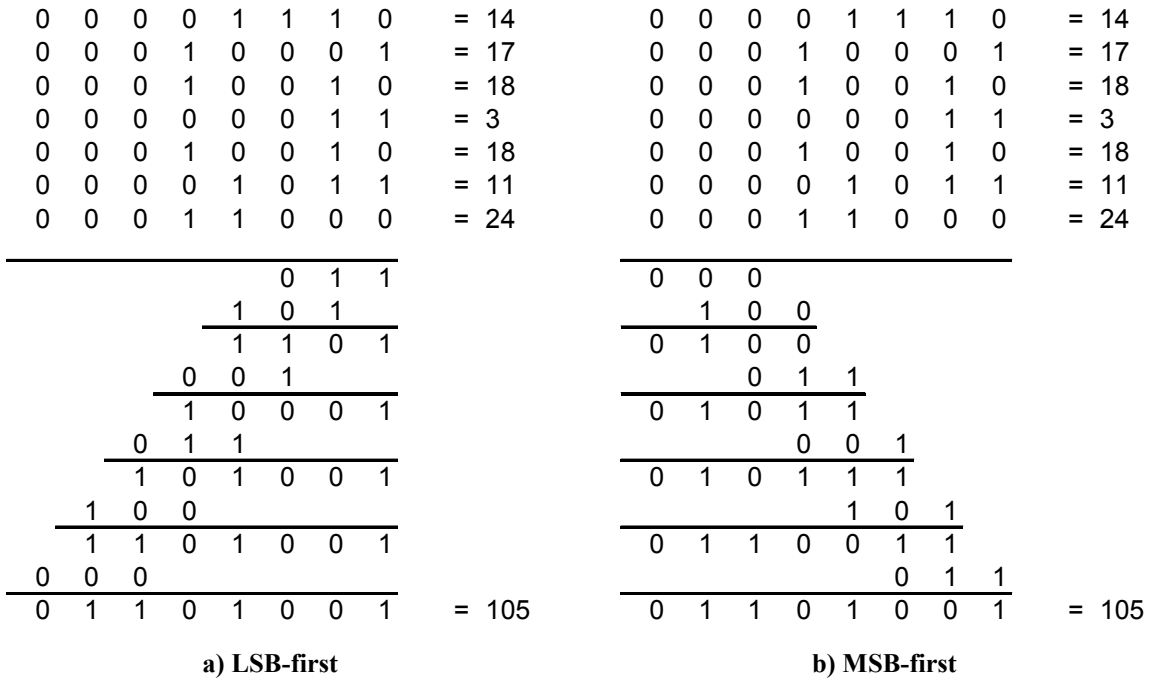


Fig. 5-13 Summation of multiple operands

As the input values are received MSB-first, in Fig. 5-13 b) the adding starts by calculating the partial sum of the MSB. In the next step the partial sum for the next lower bit position is calculated. Again, the partial sums are accumulated over all steps until the LSB bit of the inputs is processed. Fig. 5-14 illustrates a hardware realization of such an MSB-first adder. The adder consists of a partial-sum generator which summarizes the bits of all the operands for one bit position and an accumulator unit. The registers in that unit are initially set to *'0'*. Then the register data is multiplied by two in each step which is realized by a left shift of one bit position. Subsequently, the actual partial sum is added and the result is stored in the registers again.

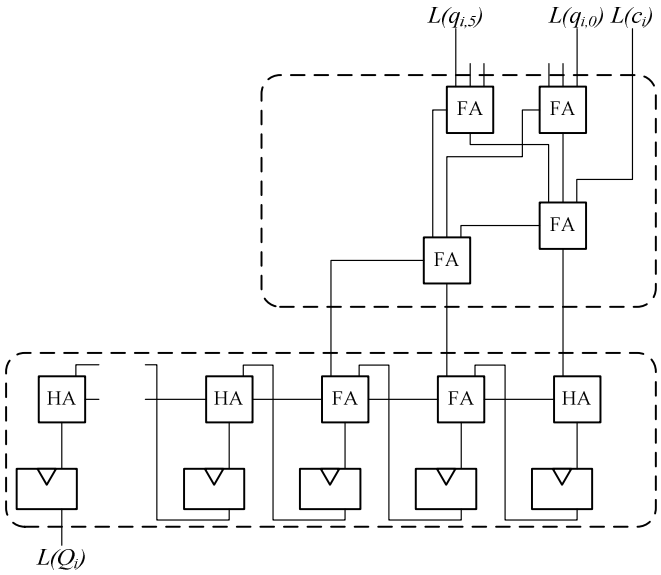


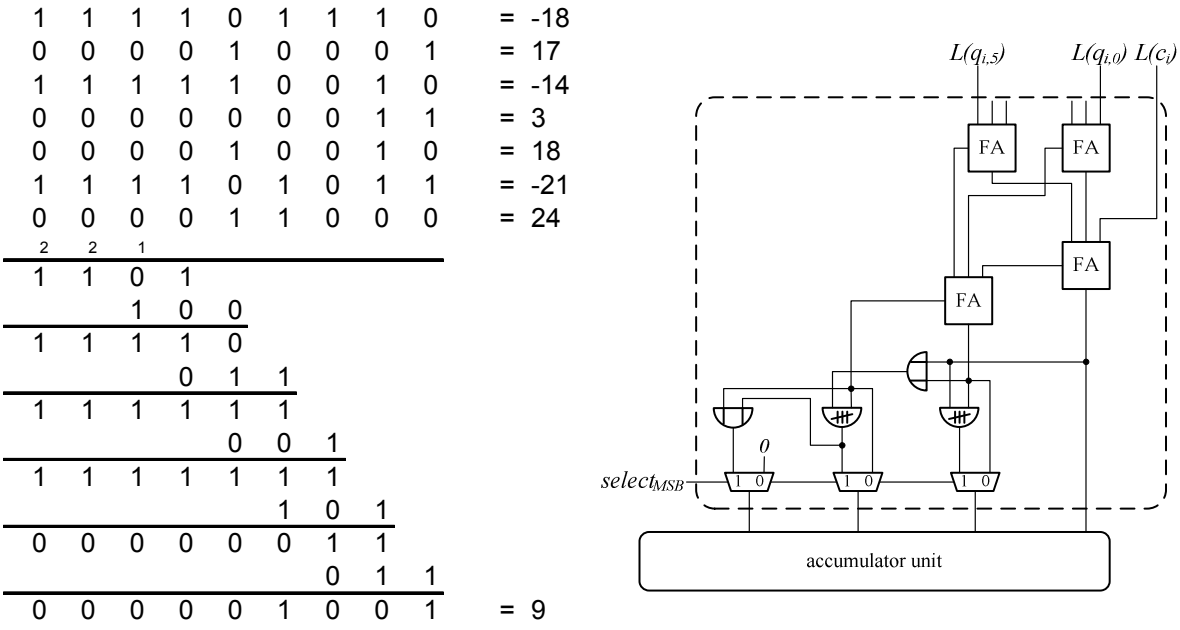
Fig. 5-14 Partial-sum generator and accumulation unit

In contrast to a fully bit-parallel realization of the bit node the area can be significantly reduced. However, there are some modifications which are required to use this adder in the bit node. The main modifications base on the fact that

1) The adder has to support negative input values.
2) The adder in Fig. 5-14 allows for a calculation of $L(Q_i)$. However, the calculation of the d_V bit-node messages $L(q_{i,j})$ requires the integration of a subtractor.
3) The input of the bit node is in sign-magnitude representation but the adder requires a two's-complement representation.
4) The bit-node messages have to be saturated before sent to the check node.
5) The critical path of the adder is comparable to the critical path of a bit-parallel node implementation and, thus, will limit the achievable clock frequency.

Ad 1)

In the example of Fig. 5-13 only positive operands are considered. Therefore, the required extension of the operands to the word length does not affect the result. However, if at least one of the operands is negative, the partial sums for the bit weights larger than w are not zero. In the example in Fig. 5-15 a) the previously discussed partial-sum calculation would fail, as the partial sum does not only have to include the bit weight w in the first step but also the higher bit weights.



a) MSB-first summation with negative operands **b) Block diagram**

Fig. 5-15 Sign extension in partial-sum generator

As the bits w to w_{EXT_BN}-*1* of the operands are equal, a direct mapping of the partial sum without sign extension to the partial sum with sign extension is possible. The resulting logic is shown in Fig. 5-15 b). In the clock cycle in which the MSB is processed the output of the sign extension logic is connected to the accumulator unit. In the other cycles this logic is bypassed.

Ad 2)

The circuit in Fig. 5-14 only calculates the $L(Q_i)$ value. However, for the next decoding iteration all d_V bit-node messages $L(q_{i,j})$ have to be calculated. Thereby, again the information of all check-node messages is summarized, before the corresponding check-node message is subtracted for each bit-node message. But instead of calculating the complete $L(Q_i)$ value the subtraction is done bit wise. Therefore, a separate partial sum for all bit-node messages can be generated by decrementing the partial sum, if the bit of the corresponding check-node message is one. Otherwise, the partial sum is not modified. Thus, only two different partial sums are possible. To reduce the hardware complexity the partial sum can be decremented and, afterwards, the partial sum for each bit-node message is derived by multiplexing. Fig. 5-16 shows the resulting structure.

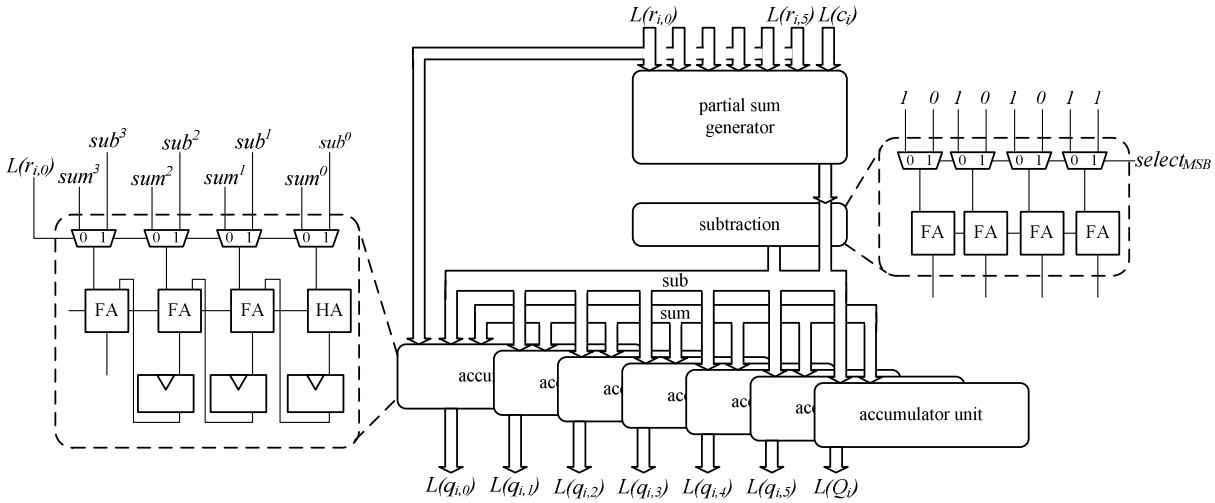


Fig. 5-16 $L(q_{i,j})$ and $L(Q_i)$ register stages

The sign extended partial sum is fed into the subtraction logic. Considering that in the MSB cycle the sum is extended, in this cycle the influence of a one at the input of the bit node is neutralized by incrementing the partial sum by one. In the other clock cycles the partial sum is decremented.

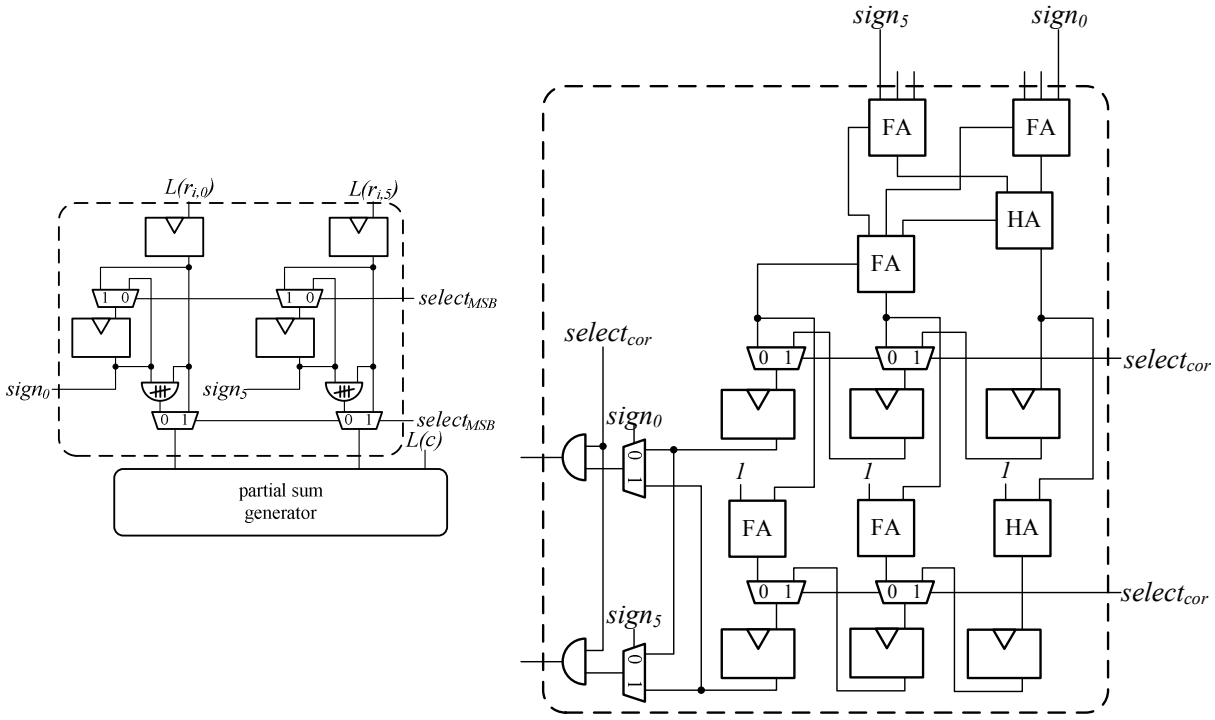
The sign extended partial sum (*sum*) and the result of the subtraction (*sub*) are sent to all d_V accumulation units. These units are extended by multiplexer. In dependency of the actual bit of the check-node message $L(r_{i,j})$ either the *sum* signal or the *sub* signal is accumulated.

Ad 3)

The received check-node messages are in sign-magnitude representation but the adder requires two's-complement representation. The conversion from sign magnitude to two's complement requires the inversion of all bits with a subsequent addition of one LSB. This is not integrable into the bit-serial MSB-first data flow. The basic idea to overcome this problem is to use an one's-complement number representation in the multi-operand adder. The conversion to one's complement only requires an XOR-operation with the actual sign bit. Therefore, in the MSB cycle the sign is stored in an additional register and in the following clock cycles the bits are combined in an XOR-gate with that sign, as shown in the block diagram in Fig. 5-17 a).

The two's-complement correction is done cumulated in the last three clock cycles. Therefore, the signs of all A-posteriori values are summarized, as shown in Fig. 5-17 b). The resulting three bit value is fed bit-serially into the accumulator unit in the correct clock cycles indicated by a high level on the $select_{cor}$ signal. Again, the influence of the sign of operand $L(r_{i,j})$ on the two's-complement correction for the bit-node message $L(q_{i,j})$ has to be cancelled out. Therefore, two different correction terms have to be calculated, one which summarizes all

signs and one which is decremented by one. In dependency of the $sign_i$ signal either the sum of all signs or the decremented sum is chosen.



a) One's-complement conversion **b) Two's-complement correction**

Fig. 5-17 Two's-complement conversion

Ad 4)

The summing-up of the check-node messages and of the A-priori information results in an extended word length which can be calculated based on (3-1). Therefore, a saturation logic is required. Additionally, the bit-node message needs to be converted from two's-complement into sign-magnitude representation. Both operations do not affect the sign of the message. Therefore, to reduce the critical path it is possible to perform both operations, while the sign is sent to the check nodes and, thus, in the first clock cycle of the next iteration.

Based on the unsaturated operand a detection logic determines, if the magnitude can be represented using five bits as shown in Fig. 5-18. If there is an overflow, a control signal is set which selects the maximal possible magnitude ('11111'), as the magnitude of the saturated bit-node message. Otherwise, the conversion from two's complement to magnitude is performed. If the bit-node message is positive and no overflow occurred the lowest five bits form the magnitude of the message. Otherwise, if the message is negative, the sign which is stored in the MSB register of the accumulator unit is one. In this case the magnitude is calculated by performing a two's-complement conversion. Therefore, the lower five bits of the operand are inverted and the sign is additionally connected with the LSB HA.

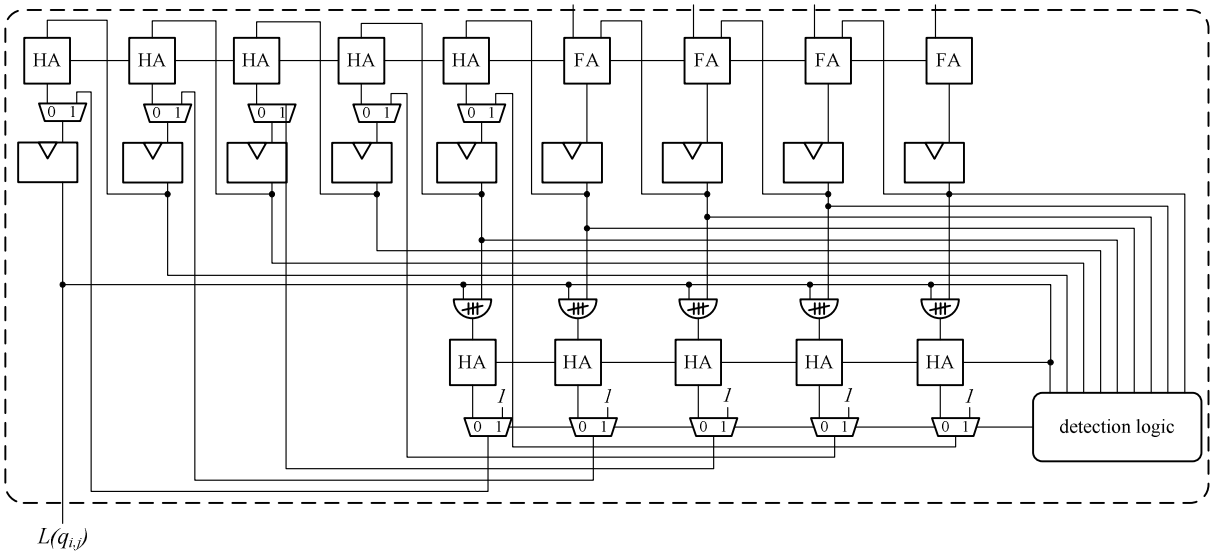


Fig. 5-18 Accumulator unit with saturation and two's-complement to sign-magnitude conversion

Ad 5)

Although the realization of a MSB-first adder in the bit node reduces the node complexity in comparison to a bit-parallel node implementation, the critical path still contains the ripple path running over the complete extended word length. The partial sum is only added in the four lowest bit weights. Nevertheless, there can be a carry overflow which affects the upper word consisting of the bit weights five to w_{EXT}-*1*. To reduce the critical path it is possible to use a carry select adder for the higher bits. It is possible to assume that there is a carry overflow in bit weight four. Subsequently, the actual carry signal of stage four selects either the result or just the shifted version of the register input.

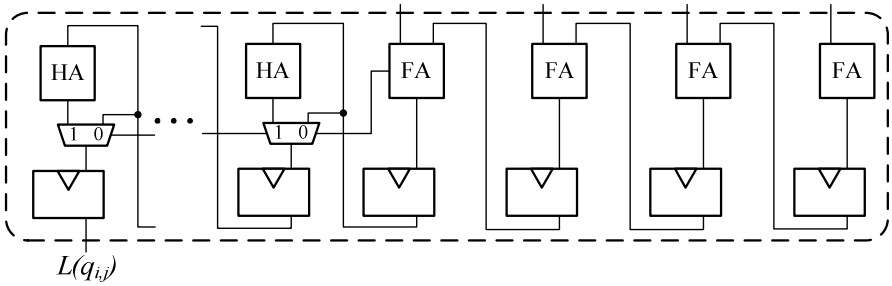


Fig. 5-19 Accumulator unit using a carry-select adder for critical-path reduction

Now the critical path only runs through the ripple path in the lower word and a multiplexer. In contrast to the carry ripple based accumulator unit the ripple path and, therefore, the critical path of the check node are reduced by five HA delays.

5.2.2 ATE-cost models

Again, it is possible to derive ATE cost models for an early estimation of the decoder features and for a comparison of the proposed architecture with the bit-serial and bit-parallel architecture. The logic area of the decoder is just a gate count. Considering the gate areas stated in Tab. 3-2, the logic area of the optimized decoder can be approximated to

$$\begin{aligned} A_{L\_PS} \approx \{&28000 \cdot m \cdot d_C \cdot +500 \cdot n \cdot (9 \cdot d_V + 30 + 5 \cdot w \\ &+ 10 \cdot w \cdot d_V + 15 \cdot \lceil ld(d_V) \rceil + 6 \cdot d_V \cdot \lceil ld(d_V) \rceil)\} \cdot \lambda^2. \end{aligned} \tag{5-3}$$

As a standard placement of the bit- and check node is considered, the routing area can be estimated using (3-39). The decoder area A_{DEC_PS} again is the maximum of the logic and the routing area.

As one iteration is performed in *(w+3)* clock cycles, the iteration period can be estimated to

$$T_{IT\_PS} \approx (w+3) \cdot \max(T_{CN\_S}, T_{BN\_S}, T_{INT\_S}) \tag{5-4}$$

Here, either the interconnect delay which can be approximated using (3-18) or the critical path in the check node which approximately is

$$T_{CN\_PS} \approx [2 \cdot ld(d_c) - 2] \cdot T_{BCS} \tag{5-5}$$

limits the iteration period.

The energy per iteration can be derived in analogy to the bit-serial decoder in chapter 3.2. Therefore, the energy can be approximated by

$$E_{DEC\_PS} = E_{L\_PS} + E_{INT\_PS} + E_{CLK\_PS}. \tag{5-6}$$

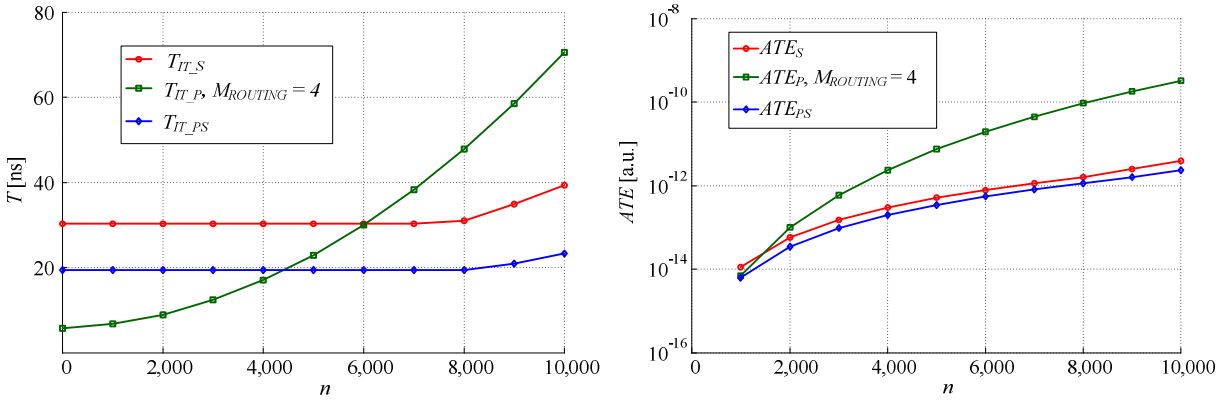
E_{L_PS} and E_{INT_PS} can be determined using (3-24) and (3-28) using the interconnect length of the considered decoder. As the number of registers in the decoder can be estimated to

$$\begin{aligned} N_{DEC\_PS\_REG} &\approx n \cdot d_V \cdot (2 + w + \lceil ld(d_V) \rceil) + m \cdot (8 \cdot d_C - 12) \\ &\approx n \cdot d_V \cdot (10 + w + \lceil ld(d_V) \rceil), \end{aligned} \tag{5-7}$$

the clock energy per iteration approximately is

$$E_{CLK\_PS} \approx n \cdot d_V \cdot (w+3) \cdot (10 + w + \lceil ld(d_V) \rceil) \cdot C_{REG\_IN} \cdot V_{DD}^2. \tag{5-8}$$

The iteration period for various block lengths is shown in Fig. 5-20 a). In comparison to the bit-serial architecture the iteration period can be reduced significantly. As can be seen in Fig. 5-20 b) this reduction is not achieved by trading throughput with area or energy, as the ATE-complexity ATE_{PS} is smaller in comparison to the bit-serial architecture over the whole considered block length range.



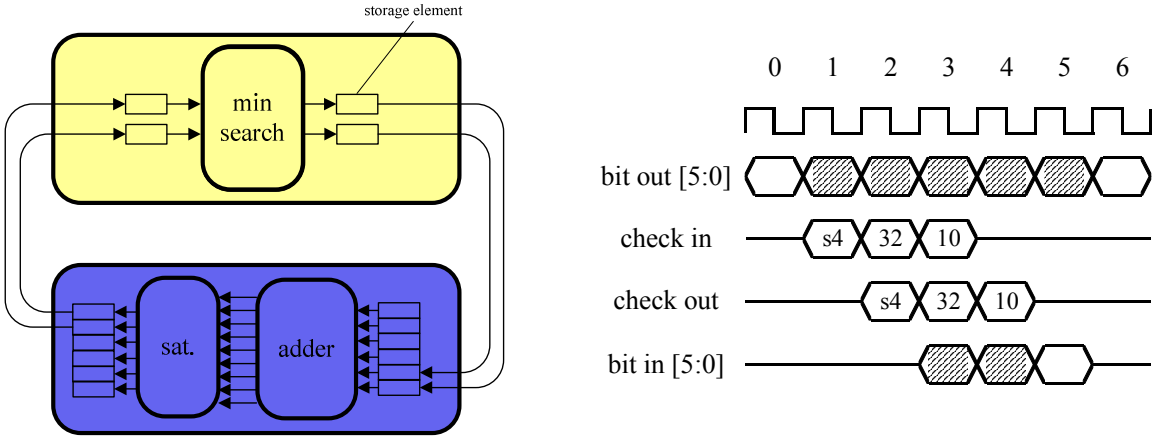
a) Iteration period **b) ATE-complexity**

Fig. 5-20 Quantitative comparison of proposed decoder architecture with bit- and check-node architectures ($d_V = 6$, $d_C = 32$, $M_{ROUTING} = 4$, $w = 6$, $\lambda = 40$nm)

Considering for example the IEEE 802.3an-compliant LDPC code which has a block length of $n = 2048$ the iteration period of the proposed architecture can be reduced by 35 % in comparison to the bit-serial architecture. Additionally, the ATE-efficiency is improved by a factor of 1.6 and 2.8 compared to the other two architectures.

5.2.3 Digit-serial decoder architectures

The cost models show that for small block lengths the iteration period of the bit-parallel decoder is significantly smaller than for the proposed decoder. However, as has been discussed in chapter 3.3.2 for code complexities lower than 40,000, the bit-serial decoder is logic dominated and the metal layers are not optimally utilized.



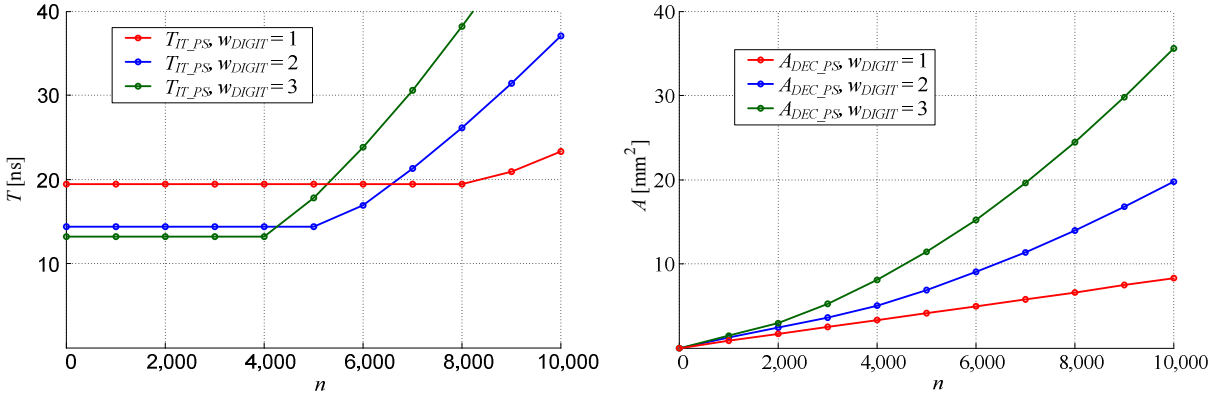
a) Block diagram of decoder loop **b) Timing diagram**

Fig. 5-21 Digit-serial architecture

To reduce the iteration period of such a decoder it is possible to stepwise increase the word length w_{DIGIT} of the interconnect until the logic and the routing area match as proposed in [41]. Considering e.g. the realization of a digit-serial interconnect with a digit word length of

two bit as shown in the block and timing diagram in Fig. 5-21, the number of clock cycles can be reduced from nine to six clock cycles. A further reduction to five clock cycles can be achieved by choosing a digit word length of three bits.

The effect on the decoder features can be analyzed quantitatively using the ATE-cost models. The achievable reduction of the iteration period is outlined in Fig. 5-22 a). For small block lengths the iteration period can be reduced by 25 and 33 % using a digit word length of two and three bit, respectively. The draw back is the increased decoder area which is outlined in Fig. 5-22 b). However, the impact on the silicon area is weaker than for the fully bit-parallel decoder architecture. Such a decoder would require a silicon area of about 5 mm^2 for a block length of 2,000, while the area of the digit serial decoder with a digit word length of three is just 3 mm^2.



a) Iteration period **b) Silicon area**

Fig. 5-22 Decoder features of digit-serial architectures (d_V = 6, d_C = 32, $M_{ROUTING}$ = 4, w = 6, λ = 40nm)

5.3 Quantitative architecture comparison

Two new decoder architectures have been presented in this chapter and the cost models derived in chapter 3 have been adapted allowing for a quantitative comparison of the decoder features for various code complexities. When comparing the iteration period of the hybrid-cell decoder (Fig. 5-7 b)) with the one of the partially bit-serial architecture (Fig. 5-20 a)) the advantage of the hybrid-cell becomes obvious (note the different abscissa scales). Considering a hybrid-cell architecture with ts_C=2 tree adder stages iteration periods of about 10 ns are possible while the iteration period of the partially bit-serial architecture is about twice as high. On the other hand, the silicon area of the partially bit-serial architecture is significantly smaller as is visible in Fig. 5-7 a). As the partially bit-serial architecture allows for a silicon area which is approximately the silicon area of the bit-serial architecture the silicon area can be reduced from about 4 mm^2 for the hybrid-cell architecture to about 1.4 mm^2 ($n \cdot d_V$=12,500).

The silicon areas and iteration periods for the different decoder architectures and for code complexities of 5,000, 10,000, and 20,000 are summarized in Fig. 5-23. For the smallest code complexity all architectures, and especially the hybrid-cell architecture, lie very close to the minimal AT-complexity which is defined by the bit-parallel architecture. When considering that the hybrid-cell architecture shows a reduced interconnect complexity and, thus, a reduced total interconnect capacitance this architecture is expected to allow for even a smaller ATE-complexity than the bit-parallel architecture. Therefore, this architecture is very attractive for applications with a high-throughput and / or low-latency specification and small to medium code complexities. Another advantage of the hybrid-cell architecture in contrast to bit- and check-node-based architectures is the higher regularity which reduces the design effort when considering for example a full-custom design approach. For a hybrid-cell decoder without any tree stages (ts_V=ts_C=0) only one cell needs to be designed which has a significantly smaller complexity than a bit or a check node. When considering tree stages as for example shown in Fig. 5-3 a) the design effort of the additional cells is also very small.

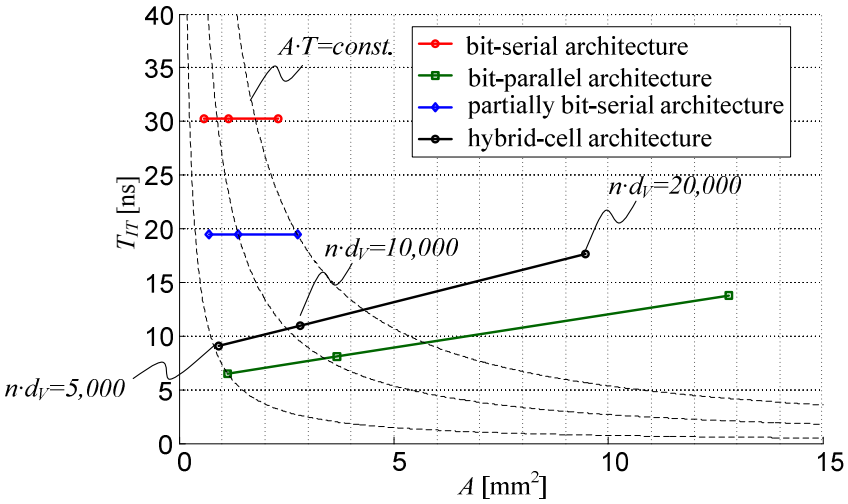


Fig. 5-23 Quantitative area and timing comparison of decoder architectures (d_V = 6, d_C = 32, $M_{ROUTING}$ = 4, w = 6, λ = 40nm)

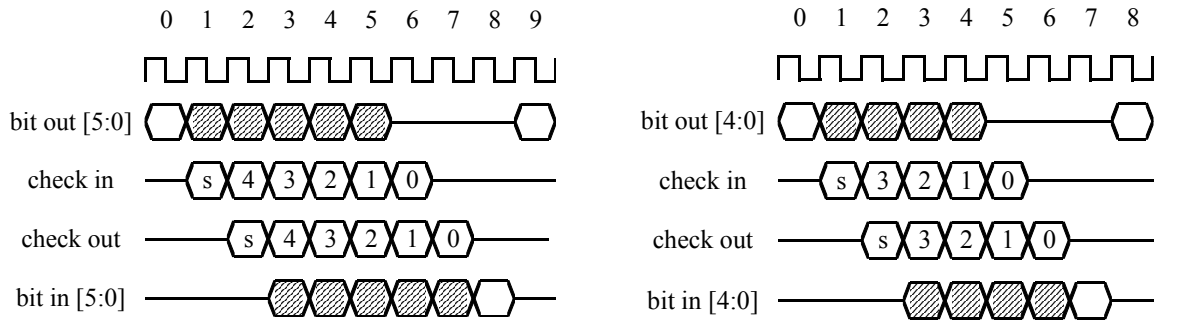
With an increasing code complexity the decoder architectures with a bit-parallel interconnect show a significant increase in silicon area and, additionally, the difference in iteration period to the partially bit-serial architecture gets smaller. This leads to a larger and larger distance to the minimal AT- and, thus, ATE-complexity which is achieved by the partially bit-serial architecture. As, except for very small code complexities, this architecture allows for the smallest ATE-complexity, it should be employed whenever the timing specification of the given application can be met.

6 Highly-optimized full-custom designed LDPC decoder

The optimization of high-throughput LDPC decoders started on algorithm and architecture level is continued on circuit and physical implementation level in the following. Therefore, a decoder macro has been further optimized and realized in a 40-nm CMOS technology [57]. As currently the IEEE 802.3 an [6] is the most challenging standard from a throughput perspective it has been chosen as the exemplary code. It has been shown, that for the code complexity of 12,288 the new partially bit-serial decoder architecture features the smallest decoder complexity (Fig. 5-20 and Fig. 5-23). Additionally, the hardware-efficient post-processing factor identified in chapter 4.2.4 needs to be integrated into this architecture. To continue the optimization on circuit and physical level the decoder is realized in a full-custom design approach. Thereby, the high modularity of the LDPC decoder reduces the design effort, while such a flow allows for an extensive optimization on lower design levels.

6.1 Decoder implementation

In chapter 4 it is shown that the Min-Sum algorithm in combination with a normalization factor of 0.5 allows for almost the optimal decoding performance and simultaneously reduces the hardware complexity. In the proposed decoder architecture the main advantage is the reduced iteration period. In contrast to the decoder in Fig. 5-9 c) the iteration period can be reduced from *w+3* to only *w+2* clock cycles as the timing diagrams in Fig. 6-1 reveal.



a) Without post-processing function **b) With hardware-efficient normalization factor**

Fig. 6-1 Timing diagram of optimized architecture

The check node is not affected by the introduction of the hardware-efficient normalization factor. Therefore, both check-node architectures described in chapter 5.2 can be used. As there is nearly no difference in silicon area and critical path, the minimum search of Fig. 5-11 is chosen in the following. A schematic of the check node including the input and output registers and the sign calculation is shown in Fig. 6-2. In the schematic the used partitioning of the check node to allow for a full-custom design scheme is illustrated, additionally. The

check node bases on three different cells: one cell contains one BCS-element, another cell two input and two output registers, and the third cell eight XOR-gates. The number of registers and XOR-gates per cell is chosen so that the size of these cells equals the size of one BCS-cell. Considering a check-node degree of $d_C = 32$, the node consists of 90 BCS-cells, 16 register cells, and eight XOR-gate cells.

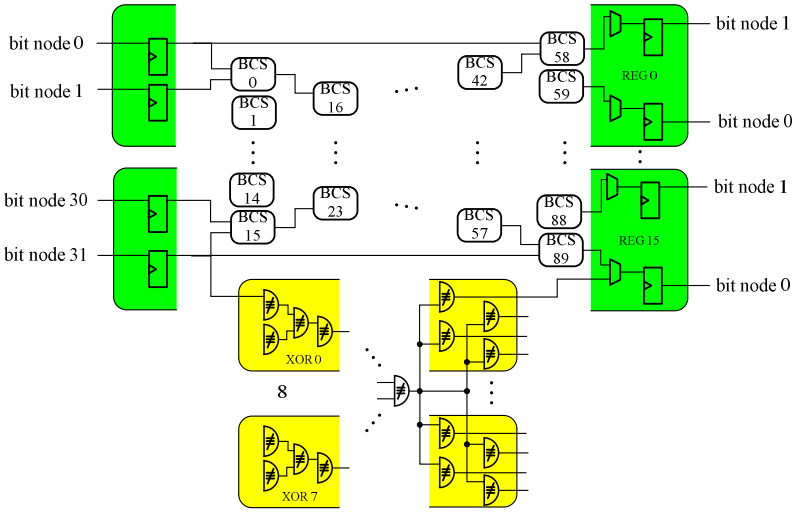
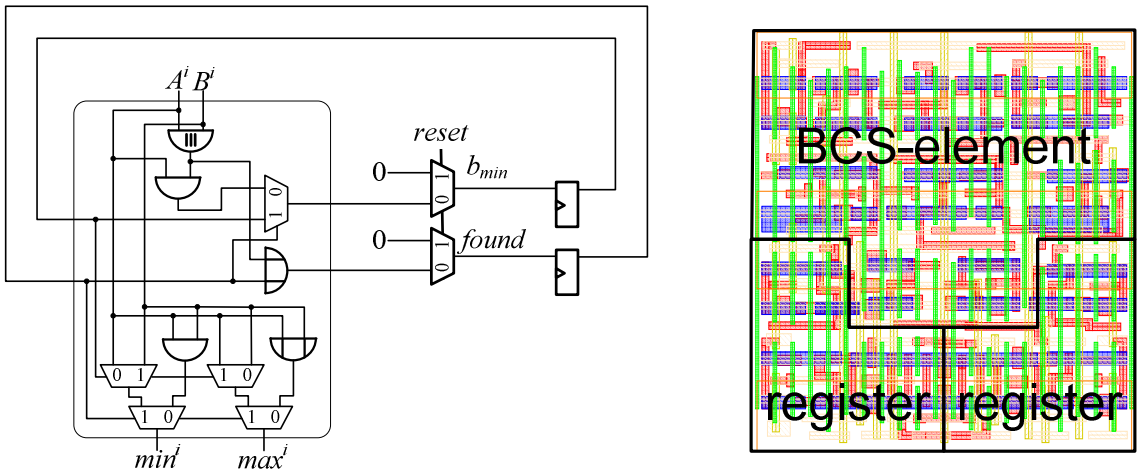


Fig. 6-2 Check-node structure

Fig. 6-3 illustrates the schematic and the physically optimized layout of the BCS-cell. In addition to the standard design rules, the layout follows DFM rules as e.g. the poly silicon is arranged periodically without any jogs. The cell occupies a silicon area of about 12 µm^2. The layouts of the two other cells are physically optimized following the same DFM rules, as well.



a) Schematic **b) Layout**

Fig. 6-3 Physically optimized BCS-cell

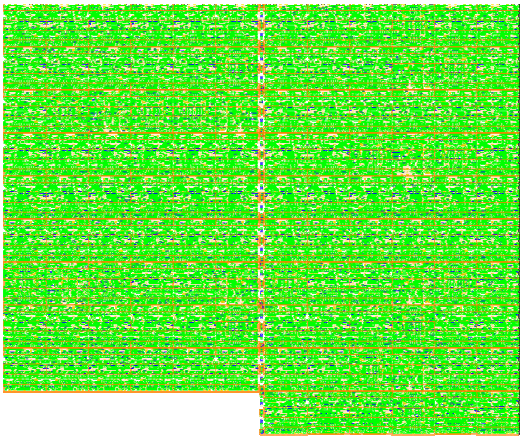
The next step in the design flow is the generation of the check-node layout by combining multiple instances of the three different leaf cells. Therefore, the data-path-generator presented in [60] is used. Thereby, the placement of each individual cell instance highly

affects the routing requirement inside the nodes. The more routing resources are occupied by internal node interconnects, the lower is the available routing length to realize the global interconnect. Therefore, the placement of the leaf cell instances inside the node is critical, and an optimization is mandatory.

The simulated annealing algorithm discussed in 3.1.2 has been adapted to this placement problem and has been used to optimize the position of each of the 114 leaf-cell instances leading to the routing optimized placement illustrated in Fig. 6-4 a). As one XOR-cell is connected to two register-cells (e.g. ‘xor 0’ to ‘reg 14’ and ‘reg 15’), the algorithm places these cells next to each other. Moreover, the min-cells which are directly connected to the register cells are also located close to the respective register cell (‘min 58’ – ‘min 89’ to ‘reg 0’ – ‘reg 15’).

BCS 80	BCS 20	BCS 8	BCS 51	BCS 77	BCS 9	BCS 76	BCS 65	BCS 64	REG 3	BCS 3	BCS 44
BCS 53	BCS 81	BCS 10	BCS 50	BCS 74	REG 9	BCS 35	BCS 45	BCS 2	XOR 6	REG 2	BCS 63
BCS 79	REG 11	XOR 2	REG 10	REG 8	XOR 3	BCS 30	BCS 17	BCS 34	BCS 42	BCS 58	BCS 62
BCS 52	BCS 11	BCS 21	BCS 78	BCS 75	BCS 24	BCS 16	BCS 1	REG 1	XOR 7	REG 0	BCS 0
BCS 39	BCS 88	BCS 26	BCS 38	BCS 32	BCS 29	BCS 31	BCS 36	BCS 60	BCS 59	BCS 61	BCS 43
BCS 57	BCS 89	BCS 33	BCS 27	BCS 28	BCS 83	BCS 25	BCS 18	BCS 46	BCS 5	BCS 4	BCS 47
REG 14	XOR 0	REG 15	BCS 82	BCS 54	XOR 1	REG 12	BCS 37	BCS 67	XOR 5	REG 4	BCS 66
BCS 14	BCS 87	BCS 41	BCS 22	BCS 13	REG 13	BCS 12	BCS 19	BCS 71	REG 5	BCS 69	BCS 68
BCS 86	BCS 56	BCS 15	BCS 23	BCS 40	BCS 85	BCS 55	BCS 6	REG 6	XOR 4	BCS 48	BCS 70
						BCS 84	BCS 49	BCS 72	REG 7	BCS 7	BCS 73

a) Routing optimized leaf-cell placement



b) Layout

Fig. 6-4 Physically optimized check node

The optimized placement scheme has been integrated into the HDL description for the DPG. After placing the leaf cells using the DPG, the interconnect is realized using the IC-Craftsman [58]. The resulting layout of one check node is depicted in Fig. 6-4 b). It consists of about 12,000 transistors and has a size of about 1,350 μm^2. The layout utilizes the lower three and part of the forth metallization level.

Based on circuit simulations of an extracted netlist the critical path of the node has been determined for a nominal supply voltage range between 700 and 900 mV and is depicted in Fig. 6-5. The simulations are carried out using slow transistor models, a temperature of 0° and a supply voltage of 90 % of the nominal V_{DD}.

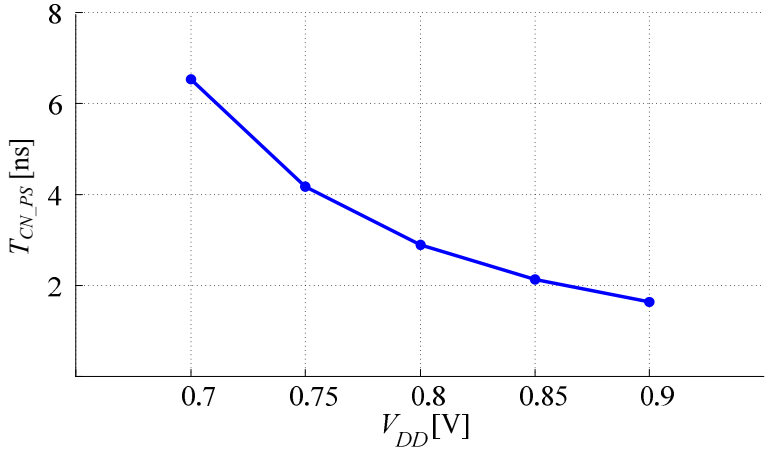


Fig. 6-5 Check-node critical path (ss, 0°, 90% supply voltage)

For a supply voltage of e.g. 0.8 V the critical path is about 3 ns. Test vectors which are generated using a bit-true equivalent C-model of the decoder (5 dB) are used in a circuit simulation to determine the power consumption at this clock period. The total power of one check node (typical corner, 25°) approximately is 0.17 mW. Fig. 6-6 gives a more detailed view on the power figures of the check node. 62 % of this power is the AC power of the local interconnect including all parasitic capacitances in the layout. Only 38 % of the total power is assigned to the actual transistors. Thereby, 13 % of the transistor power or 5 % of the total power is due to leakage current. Considering the fast corner in a hot environment, the leakage power increases significantly and would account for more than 70 % of the total power. Therefore, it is mandatory to reduce the leakage power by applying back biasing. A back-bias voltage of 0.5 V would e.g. reduce the leakage power by 60 % leading to a total power of the check node in the fast corner of 0.44 mW.

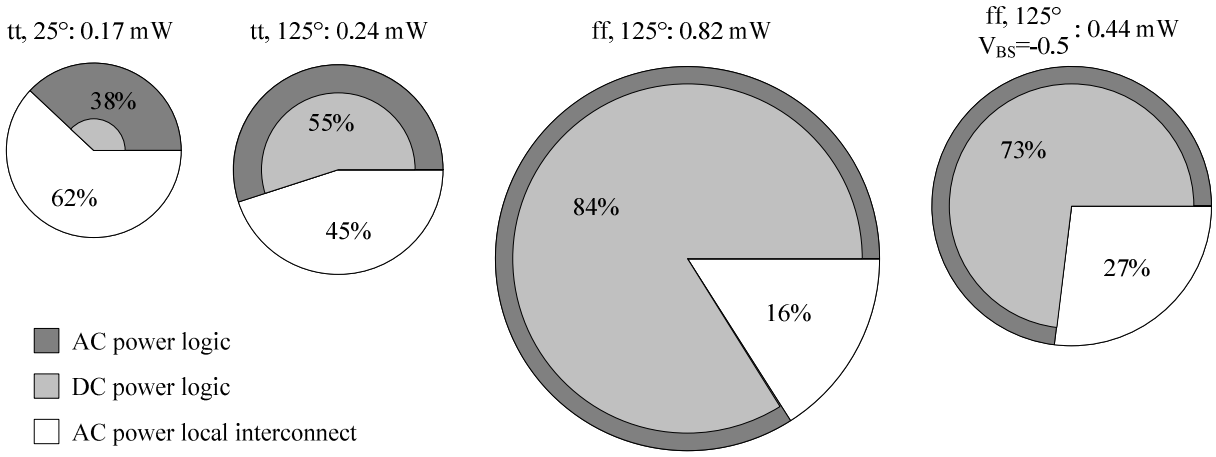


Fig. 6-6 Power break-down check node

In contrast to the check node the hardware-efficient normalization factor allows for a further reduction of the bit-node complexity. As the word length of the six check-node

messages $L(r_{i,j})$ is reduced by one bit, the multi-operand adder has to sum-up the channel information with a word length of w and d_V check-node messages with a reduced word length of $w\text{-}1$ instead of d_V+1 operands with a word length of w. Considering e.g. a message word length of $w = 6$ bit, the maximum sum is 249. Therefore, eight instead of nine bits are sufficient and each register stage requires only eight registers.

However, an additional modification to the bit-node architecture of chapter 5.2 is required. The reason is the calculation in clock cycle three (see Fig. 6-1). In the original bit-node architecture the MSB of the channel information and of the check-node messages is processed. In contrast the modified bit node needs to process the two MSBs (bit weights $w\text{-}1$ and $w\text{-}2$) of the channel information and the MSB of the check-node messages (bit weight $w\text{-}2$) due to the different word lengths of these messages.

If the two leading bits of the channel information are equal, the original circuit (Fig. 5-15 b)) would be sufficient. However, the circuit would calculate an incorrect partial sum in this clock cycle when the two bits differ. In the case the two leading bits of the channel information are ‘10’ (‘01’), the original circuit would calculate a partial sum which is smaller (larger) by two. The correction can be performed e.g. in the subtraction stage as depicted in Fig. 6-7. Therein, based on two control signals (‘A-priori correction’ and ‘A-priori sign’) the partial sum is either incremented by two, decremented by two, or kept unchanged.

Two further optimizations have been introduced to the register stage in contrast to the decoder of chapter 5.2. A comparison of the active clock cycles of the adder in the register stage and the adders in the saturation logic (see Fig. 5-18) show that a reuse of these adders is possible. While the adders in the register stage are active only in clock cycles 4 to 7, those in the saturation logic are utilized in clock cycle 0.

The second modification targets the switching activity of the interconnect lines from the bit to the check nodes. To reduce the decoder power the switching activity in the decoder loop should be minimize. As can be seen in Fig. 6-1 b) the utilization of the interconnect and the check node is $^5/_8$. To benefit from this property the output of the bit node is held constant in clock cycles four to seven by an additional multiplexed register.

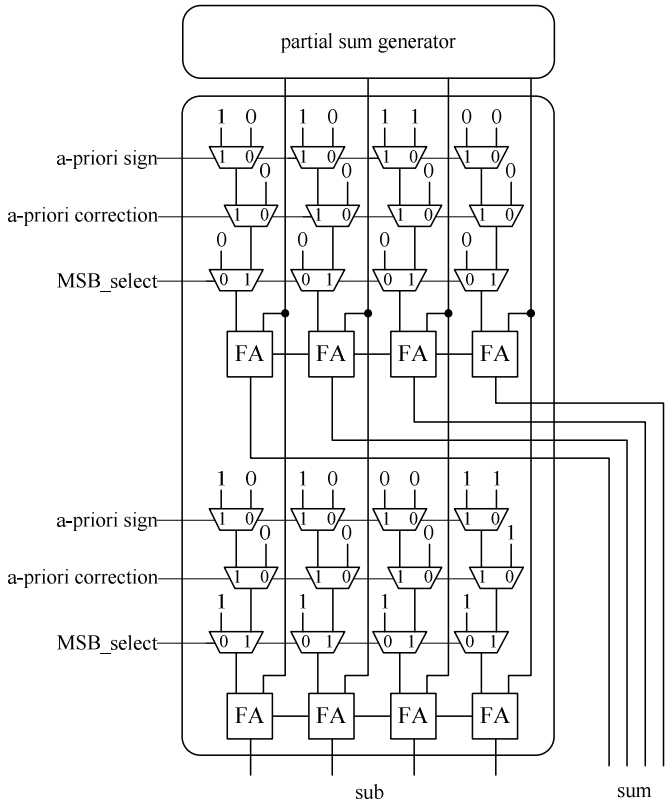


Fig. 6-7 Schematic subtraction logic

While the input register stage, the one's-complement conversion, the partial-sum generator, and the two's-complement correction logic are manually designed leaf-cells, the accumulator unit itself is generated using the DPG. The resulting bit-node layout which is placed using the DPG and routed using the IC-Craftsman is depicted in Fig. 6-8. The bit-node layout consists of about 6,000 transistors and has a size of 760 μm^2. Again, as for the check node, the lower three and part of the forth metallization levels are utilized.

A message parallel IEEE 802.3an-compliant decoder consists of 2048 bit and 384 check nodes. As discussed before, the placement of the 2432 node instances in the decoder highly affects the interconnect complexity and, therefore, an optimization is mandatory. By optimization the placement using the simulated annealing algorithm, the interconnect length can be reduced by about 20 %. The progress of the optimization has already been shown in Fig. 3-6 a). The total Manhattan length of the optimized placement is 19.8 m.

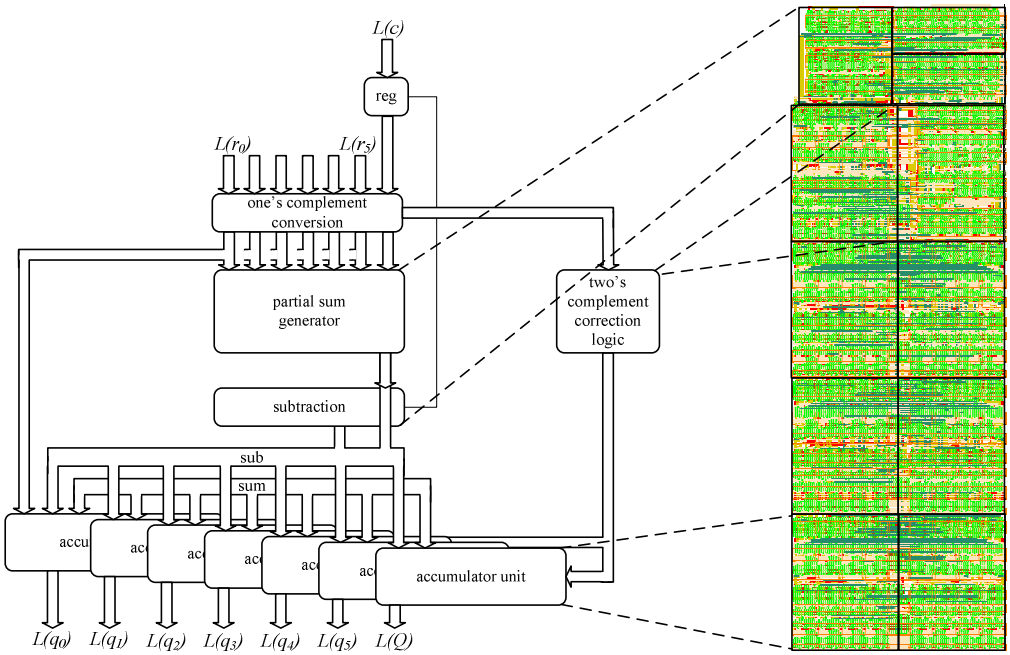


Fig. 6-8 Layout bit node

Using the DPG the 384 check and 2048 bit nodes have been instantiated in the decoder macro based on this optimized placement. Subsequently, the global interconnect is routed on 3.5 metallization levels. Due to the low number of metal layers used for the node layouts and the optimized placement, no artificial increase of the decoder area is required.

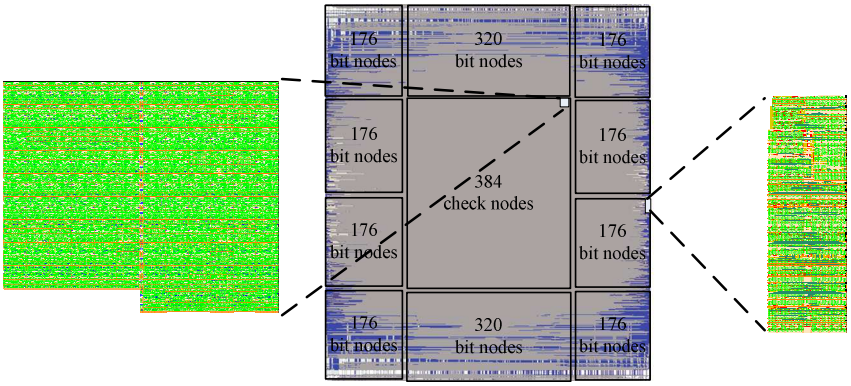


Fig. 6-9 Decoder layout

The resulting decoder layout is shown in Fig. 6-9. It occupies a silicon area of 2.3 mm^2 and consists of more than 18 million transistors. Therefore, a transistor density as high as eight million transistors per square millimeter is achieved. This density is very high, especially when considering the routing domination of LDPC decoders. In comparison, the quadratically scaled transistor density of [5] is only two million and the one of [15] is 3.8 million transistors per square millimeter when considering the average number of transistors per gate being four.

The routed wire length for the global interconnect is 19.92 m. Therefore, the overhead in comparison to the Manhattan length is less than 1 %. Fig. 6-10 outlines the post-routing wire-length histogram. The maximal wire length is 2.3 mm and the mean wire length is about 0.81 mm.

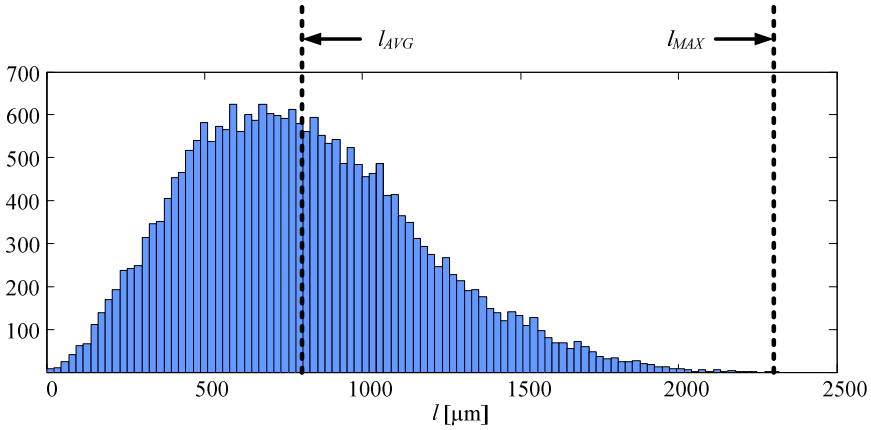
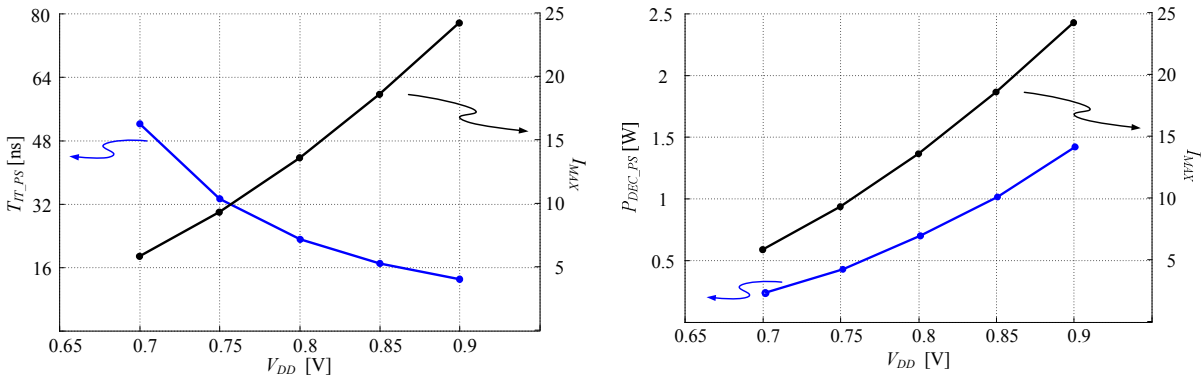


Fig. 6-10 Post-routing wire-length histogram

The critical path of the decoder runs through the check node and, therefore, the critical path depicted in Fig. 6-5 limits the clock frequency. Considering the specified block throughput rate of 3.125 million blocks per second, the decoder is able to perform 24 iterations at a nominal supply voltage of 900 mV realizing a block latency of just 320 ns (see Fig. 6-11 a)).



a) Iteration period ss (0°) **b) Decoder power tt (25°)**

Fig. 6-11 Decoder features

The decoder power P_{DEC_PS} is outlined in Fig. 6-11 b). Targeting e.g. 13 decoding iterations, the power dissipation is 0.75 W (V_{DD} = 0.8 V). The power break down (Fig. 6-12) shows that 26 % of that power is the AC power of the global interconnect. In total, the global and the local interconnect accounts for about 50 % of the decoder power.

In the fast corner the decoder power is 1.7 W when applying a back-bias voltage of V_{BS} = -0.5 V. In comparison to the typical corner (25°) where the DC power is very small the leakage power in the fast corner (125°) increases to 0.5 W which is 40 % of the total power.

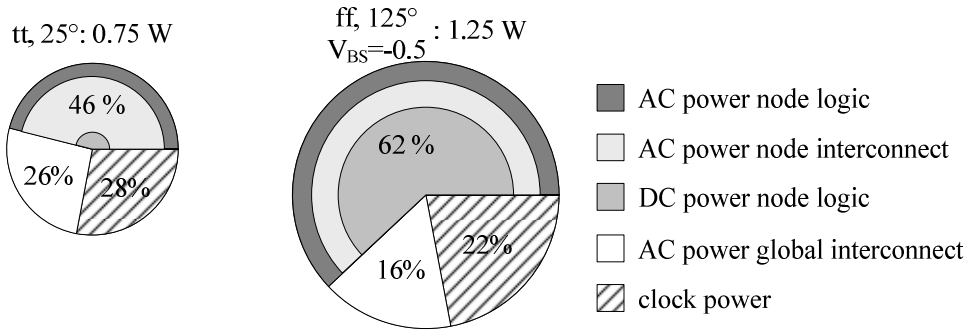


Fig. 6-12 Power break-down @ 13 decoding iterations, V_{DD} = 800mV

6.2 Implementation of stopping criteria

As can be seen in the FER diagram in Fig. 4-16 b), more than 99 % of the blocks are decoded within the first two iterations for high SNRs. Nevertheless, to achieve very low FERs for some blocks more than ten iterations are necessary. Thus, there is a large gap between the average and the maximum number of iterations. Therefore, it is possible to stop the decoding of the complete block as soon as the block is error free. There are two possible ways to benefit from this early termination. One of the possibilities is to increase the decoder throughput as is e.g. done in [61]. As typically a constant block latency has to be ensured this requires synchronization at the decoder in- and output which would increase the decoder complexity. On the other hand it is possible to reduce the decoder power by switching off the decoder or at least parts of the decoder as soon as the block is error free [62].

A block is error free, if the decoder output namely the signs of the $L(Q_i)$ information fulfill all parity checks defined by the parity-check matrix H. It is possible to implement an external check as sketched in Fig. 6-13 which consists of the calculation of m parity checks with a subsequent combination of the results. Although the additional active silicon area is comparatively small to the total decoder area, in total $n \cdot d_V$ additional interconnect lines are required which is an overhead of 50 % in comparison to the global interconnect in a bit-serial decoder.

To reduce the decoder complexity in [62] an approximate calculation is proposed. Instead of sensing the absence of errors using the sign of the $L(Q_i)$ values, the results of the sign calculation in each check node is used as is depicted in Fig. 6-13, as well. Therefore, the sensing bases on the $L(q_{i,j})$ values which are sent to the check nodes. This can lead to a

different conclusion about the decoding status of the block. In Fig. 6-13 the parity check using the $L(Q_i)$ value would, therefore, indicate a successful decoding, when considering the error-free symbol for the depicted bit node is '-1'. In contrast, the parity check performed on the signs of the bit-node messages sent to the different check nodes would still indicate an error as illustrated in the example. Thus, the false conclusion would be that the block still contains errors. However, if the block would be read out in that iteration, it would be error free.

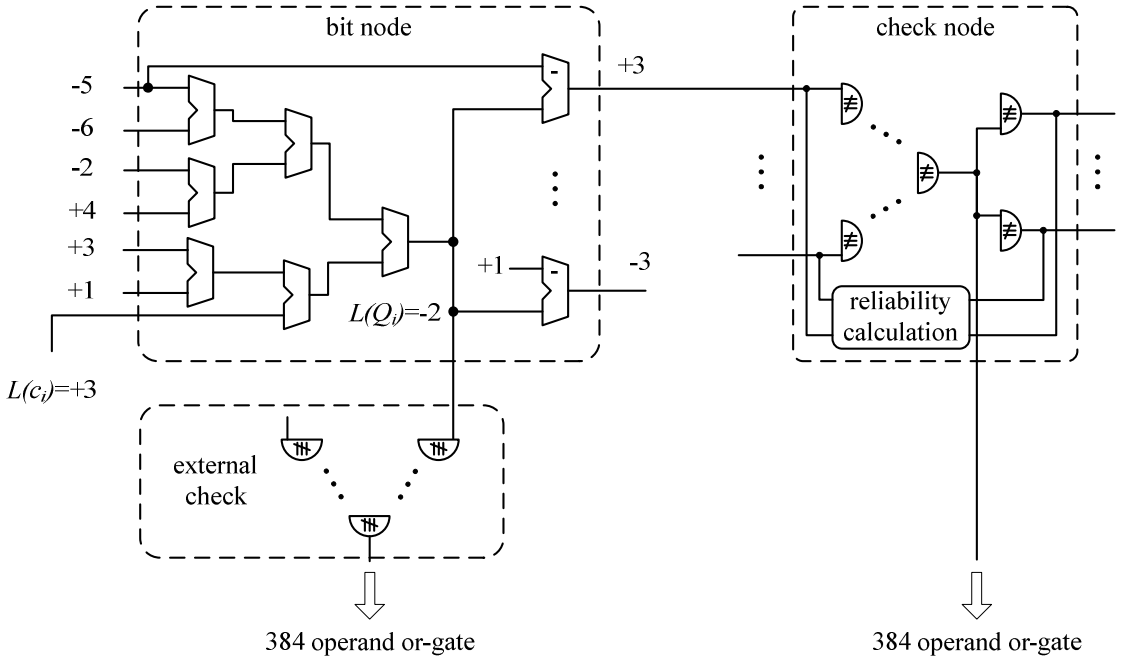


Fig. 6-13 Sensing of error-free blocks

The average delay of the estimated calculation can be determined by Monte-Carlo simulation. Fig. 6-14 shows the fraction of blocks for which both calculations sense the error-free block in the same clock cycle and with one clock cycle delay. For small SNRs about every second block would require one additional iteration as this fraction gets smaller for higher SNRs. In the whole analyzed SNR range only less than 1 % of the blocks require two additional iterations.

However, as the two checks differ, it might be possible that a block still contains errors with the sensing circuit indicating that the block is error free. Although this behavior can not be absolutely excluded, simulations of 5 million blocks do not show such a case. Thus, the probability is lower than $2 \cdot 10^{-7}$. Simulation results presented [62] underline these results.

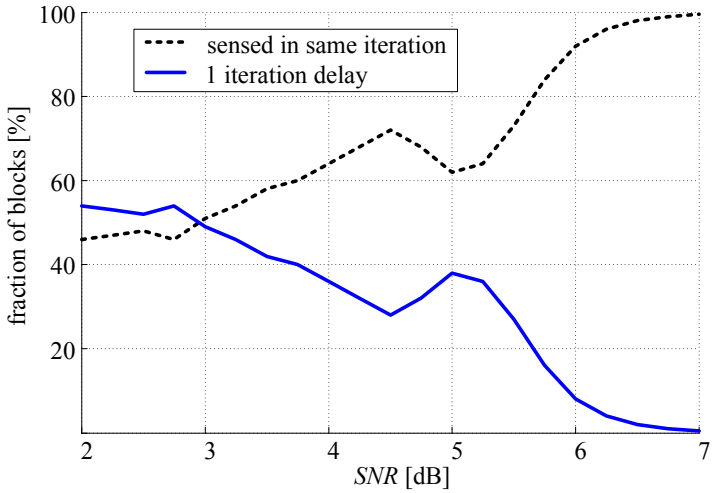
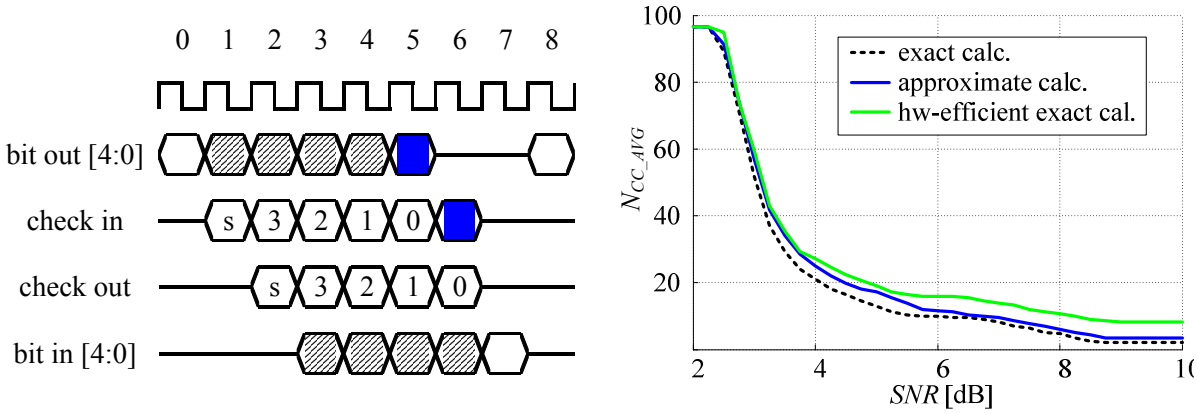


Fig. 6-14 Performance of error-free sensing

It is also possible to use the XOR-gates in the check nodes to calculate the correct parity check by time-multiplexing the existing interconnect lines. This possibility is discarded in [62], as it would result in an increased iteration period. However, in the proposed decoder architecture the utilization of the interconnect lines is only $^{5}/_{8}$. This allows for the transmission of the sign of $L(Q_i)$ without affecting the decoder throughput. Thereby, subsequent to the transmission of the bit-node messages $L(q_{i,j})$ the sign of the $L(Q_i)$ signal is sent to the connected check nodes in clock cycle five. Then the parity check is performed in clock cycle six. The resulting timing diagram is shown in Fig. 6-15 a).



a) Timing diagram hardware-efficient parity check **b) Clock cycle for different stopping criteria**

Fig. 6-15 Stopping criteria in proposed decoder architecture

If the calculation in the nodes is switched off after the block is error free, the reduction in decoder power can be expressed as [62]

$$P'_{DYN} = (1+\gamma)\cdot\left[P_{CLK} + \nu \cdot P_{DEC}\right]. \qquad (6\text{-}1)$$

ν is the fraction of the average number of clock cycles N_{CC_AVG} in comparison to the maximal number. γ is the power consumption overhead of the additional control logic which typically can be neglected, as it will be done in the following. The activity factor can be determined by Monte Carlo simulation. In Fig. 6-15 b) the resulting average number of clock cycles for the decoder for the three different sensing schemes is outlined. The activity factor ν can then be determined as

$$\nu = \frac{N_{CC\_AVG}}{8 \cdot I_{MAX}} . \tag{6-2}$$

As an external exact calculation of the stopping criteria results in the lowest number of iterations and as it can be performed at the beginning of each iteration, it results in the lowest average number of clock cycles (dashed line). In average the decoder is active between 21, 13.5, and 10 clock cycles for an SNRs of four, five, and six dB leading to an activity factor of 0.2, 0.13, and 0.1, respectively when considering a maximum of 13 decoding iterations.

For the approximate parity check the number of average active clock cycles is larger due to possible additional iterations and due to the fact, that the calculation of the parity check is performed with two clock cycles delay. For an SNR of five dB ν would be 0.17. Considering the hardware-efficient exact sensing, the calculation is performed in clock cycle seven. In this case ν would be 0.19.

In contrast to the decoder in [62] which requires interleaving to increase the decoding throughput in the proposed decoder, the complete decoder macro can be shut down by clock gating. Therefore, not only the dynamic power consumption of the nodes can be reduced by a factor *v* but also the clock power. The resulting decoder power can be estimated as

$$P''_{DYN} = (1+\gamma) \cdot \nu \cdot [P_{CLK} + P_{DEC}]. \tag{6-3}$$

The power dissipation of the decoder for the three sensing schemes with node disabling and clock gating is listed in Tab. 6-1. Thereby, 13 decoding iterations and a clock power of 28 % (Fig. 6-12) are assumed.

Starting from an energy per iteration of 16.3 nJ a disabling of the nodes would result in an energy reduction of about 60 %. Thereby, the exact external calculation results in the largest power reduction. However, as this would result in a significant hardware overhead the implementation is not efficient. As the difference in power reduction between the three sensing schemes is small, the hardware-efficient exact sensing scheme is favorable. Using this scheme the hardware impact is minimized and a false indication of a faulty block as error free is avoided. Although the clock fraction of the decoder power without early termination is only 28 %, in a decoder with node disabling this fraction is 67 %. Therefore, it is highly

recommended to realize clock gating as in comparison to a decoder with just node disabling a further power reduction of more than 50 % from 6.8 nJ to 3.3 nJ is possible.

Tab. 6-1 Early termination power reduction (typ, 25 °, SNR = 5dB, I_{MAX} = 13)

	E_{IT} w/o early termination	Node disabling		Node disabling + clk gating	
		E_{IT}	Reduction	E_{IT}	Reduction
Exact sensing	16.3 nJ	6.0 nJ	63 %	2.1 nJ	87 %
Approx. sensing	16.3 nJ	6.5 nJ	60 %	2.7 nJ	83 %
HW-Efficient exact sensing	16.3 nJ	6.8 nJ	58 %	3.3 nJ	80 %

6.3 Benchmarking

Published decoder implementations are typically developed for different applications. Therefore, a benchmarking of these implementations requires a fair scaling of the decoder features. Even today nearly one decade after the publication of the first integrated LDPC decoder it is still challenging to fairly compare decoders for different LDPC codes, as the three basic metrics throughput, energy, and silicon area are highly complex functions of the code parameters [63]. Therefore, it is customary to assume e.g. a linear scaling of the decoder area with the block length n [62], [63]. Even worse is that the energy per bit and iteration is typically assumed to be constant for various LDPC codes, as it is not scaled with respect to code complexity. In contrast the ATE-cost models derived in chapter 3 e.g. show that for a wide range of LDPC codes the silicon area scales quadratically with the block length leading to a linear dependency between the energy per bit and the block length.

These models are used to derive fair scaling rules which are listed in Tab. 6-2. Thereby, four types of decoders are distinguished:

A) Logic-dominated decoder with a bit-serial interconnect

B) Routing-dominated decoder with a bit-serial interconnect

C) Logic-dominated decoder with a bit-parallel interconnect

D) Routing-dominated decoder with a bit-parallel interconnect

Tab. 6-3 lists decoder implementations targeting a high decoder throughput and the code and technology parameters which are required for the feature scaling. Furthermore, the decoder features silicon area, iteration period, and energy per iteration are listed for the individual code. As block interleaving and early termination to increase the decoder

throughput or decrease the decoder power are approaches which can be applied to all decoder architectures, the effects on throughput and energy are eliminated.

Tab. 6-2 High-throughput LDPC decoder scaling rules

	Bit-serial	Bit-parallel
Logic dominated	Code range $n \cdot d_v < (1700 + 150 \cdot w) \cdot M_{ROUTING}{}^2$	Code range $n \cdot d_v < 500 \cdot \frac{M_{ROUTING}^2}{w}$
	Area $A_{DEC} = l_{DEC}^2 \sim n \cdot d_V \cdot (40 + 3.5 \cdot w + 3 \cdot \lceil ld(d_V) \rceil) \cdot \lambda^2$	Area $A_{DEC} = l_{DEC}^2 \sim n \cdot d_v \cdot (11.5 \cdot w + 2 \cdot ld(d_v)) \cdot \lambda^2$
	Iteration Period $T_{IT} \sim (w+1) \cdot \max(T_{CN}, T_{BN}, T_{INT})$ using (3-18), (3-40), (3-41)	Iteration Period $T_{IT} \sim T_{BN} + T_{CN} + 2 \cdot T_{INT}$ using (3-18), (3-20), (3-22)
	Energy $E_{IT} \sim \left[49 \cdot w + 7 \cdot ld(d_V) - 38 + 0.015 \cdot w \cdot \frac{l_{DEC}}{\lambda} + 90 \cdot w^2 \right] \cdot n \cdot d_v \cdot V_{DD}{}^2 \cdot \lambda \cdot \frac{nJ}{mV^2}$	Energy $E_{IT} \sim \{11 \cdot n \cdot (3.5 \cdot w - 1.5 + 2 \cdot \lceil ld(d_V) \rceil) + 34 \cdot m \cdot (2 \cdot \lceil ld(d_C) \rceil - 3) + 0.015 \cdot n \cdot d_V \cdot w \cdot \frac{l_{DEC}}{\lambda}\} \cdot V_{DD}{}^2 \cdot \lambda \cdot \frac{nJ}{mV^2}$
Routing dominated	Code range $n \cdot d_v > (1700 + 150 \cdot w) \cdot M_{ROUTING}{}^2$	Code range $n \cdot d_v > 500 \cdot \frac{M_{ROUTING}^2}{w}$
	Area $A_{DEC} = l_{DEC}^2 \sim \left(2.4 \cdot \frac{n \cdot d_V}{M} \cdot \lambda + \sqrt{\left(2.4 \cdot \frac{n \cdot d_V}{M} \cdot \lambda \right)^2 - l_L^2 \cdot \frac{M_{ROUTING} - M}{M}} \right)^2$	Area $A_{DEC} = l_{DEC}^2 \sim \left(2.4 \cdot \frac{n \cdot d_V \cdot w}{M} \cdot \lambda + \sqrt{\left(2.4 \cdot \frac{n \cdot d_V \cdot w}{M} \cdot \lambda \right)^2 - l_L^2 \cdot \frac{M_{ROUTING} - M}{M}} \right)^2$
	Iteration Period $T_{IT} \sim (w+1) \cdot \max(T_{CN}, T_{BN}, T_{INT})$ using (3-18), (3-40), (3-41)	Iteration Period $T_{IT} \sim T_{BN} + T_{CN} + 2 \cdot T_{INT}$ using (3-18), (3-20), (3-22)
	Energy $E_{IT} \sim \left[49 \cdot w + 7 \cdot ld(d_V) - 38 + 0.015 \cdot w \cdot \frac{l_{DEC}}{\lambda} + 90 \cdot w^2 \right] \cdot n \cdot d_v \cdot V_{DD}{}^2 \cdot \lambda \cdot \frac{nJ}{mV^2}$	Energy $E_{IT} \sim \{11 \cdot n \cdot (3.5 \cdot w - 1.5 + 2 \cdot \lceil ld(d_V) \rceil) + 34 \cdot m \cdot (2 \cdot \lceil ld(d_C) \rceil - 3) + 0.015 \cdot n \cdot d_V \cdot w \cdot \frac{l_{DEC}}{\lambda}\} \cdot V_{DD}{}^2 \cdot \lambda \cdot \frac{nJ}{mV^2}$

Tab. 6-3 Decoder features of published LDPC decoder

Nr.	Author	Ref.	n	m	d_V	d_C	w	λ / nm	M	$M_{ROUTING}$	V_{DD} / V	A_{DEC} / mm^2	T_{IT} / ns	E_{IT} / nJ	Scaling rules	$A_{DEC,NORM}$ / mm^2	$T_{IT,NORM}$ / ns	$E_{IT,NORM}$ / nJ	ATE / mm^2·ns·nJ
1)	Blanksby	[5]	1024	512	3.3	6.5	4	160	5	2	1.5	52.5	15.6	10.8	D)	32.5	9.9	40.0	12856.8
2)	Zhengja	[61]	2048	384	6	32	4	65	7	4	1.2	5.35	16.1	45.1	D)	4.1	15.3	56.6	3536.9
3)	Zhengja	[61]	2048	384	6	32	4	65	7	4	0.7	5.35	115.1	16.6	D)	4.1	109.4	20.8	9302.8
4)	Mohsenin no split	[16]	2048	384	6	32	5	65	7	4	1.3	18.2	80.3	19.3	D)	9.5	66.1	16.4	10282.4
5)	Mohsenin Split-2	[16]	2048	384	6	32	5	65	7	4	1.3	8.8	22.4	10.7	D)	4.6	18.4	9.1	772.3
6)	Mohsenin Split-4	[16]	2048	384	6	32	5	65	7	4	1.3	5	7.5	6.2	D)	2.6	6.2	5.3	85.2
7)	Darabiha HD	[21]	2048	384	6	32	1	180	7	4	n.a.	17.64	20.0	0.0	D)	13.3	11.2		
8)	Darabiha	[15]	660	176	4	15	4	130	7	4	1.2	7.3	26.7	18.4	A)	3.7	14.8	64.6	3545.6
9)	Darabiha	[15]	660	176	4	15	4	130	7	4	0.6	7.3	135.8	4.8	A)	3.7	75.2	16.9	4720.7
10)	Lin	[17]	1200	480	3	7.5	6	130	8	5	1.2	10.24	50.7	6.8	D)	9.5	31.6	18.9	5662.6
11)	Ishikawa	[24]	2304	1152	3	6	6	180	6	3	n.a.	36	434.7	1565.0	D)	3.7	185.4		
12)	Mansour	[25]	2048	1024	3	6	4	180	6	3	1.8	14.3	320.0	251.8	D)	3.5	176.8	344.6	215761.6
13)	Swamy	[29]	128	64	3	6	6	180	6	3	n.a.	9.9	51.2	0.0	C)	16.1	17.3		
14)	Brandon	[19]	256	128	3	4	4	180	6	3	1.8	6.96	32.0	97.3	A)	6.4	17.9	909.8	103566.7
15)	Chen	[32]	2048	128	2.9	46	6	90	9	6	0.8	3.84	88.7	16.9	D)	4.1	58.8	32.3	7825.7
16)	Proposed		2048	384	6	32	6	40	7	4	0.9	2.3	7.5	18.5		2.3	7.5	18.5	319.6

Furthermore, those implementations performing a trade-off between decoder complexity and decoding performance using approximations and, therefore, are not bit-true equivalent are underlined in grey. These include e.g. the split-row decoder in [16] or the decoder in [15] using an approximation for the second minimum.

The published decoder features are scaled to the LDPC code of the IEEE 802.3an standard, a message word length of $w = 6$, and a 40-nm CMOS technology with seven metal layers. Thereby, for each implementation one of the four scaling rule types is chosen based on the decoder architecture and the code and technology parameters. As can be seen in Tab. 6-3, the code and technology parameters used in literature mainly lead to either routing-dominated bit-parallel or logic-dominated bit-serial decoders.

Typically, only the total number of metal layers is stated in the publications while a fair scaling would require information about the usage of these layers. Here, the number of metal layers utilized for the design of the nodes is estimated to three. Thus, the number of routing layers is assumed to be

$$M_{ROUTING} = M - 3 \tag{6-4}$$

in the following.

The normalized ATE-complexities of the published decoder implementations and of the proposed decoder are depicted in Fig. 6-16. Implementations performing a trade-off between decoder complexity and decoding performance using approximations are highlighted by open markers.

In contrast to the first published LDPC decoder 1) various decoder implementations have been proposed which achieve a lower ATE-complexity. The most ATE-efficient decoder 2) known from literature is the one presented in [61]. In comparison to [5] this decoder allows for a very low AT-complexity with a moderate increase in energy per iteration. Thereby, the reduction in ATE-complexity is achieved by multiplexing in time on message-level leading to a significant reduction in silicon area of nearly a factor of eight. In contrast, the ET-complexity is only increased by a factor of two. Thereby, the increase in iteration period is limited by an extensive pipelining scheme.

All other published decoder implementations perform a similar trade-off between decoder area and ET-complexity. This becomes obvious when looking at the ET-complexity illustrated in Fig. 6-17 in which the decoder 1) allows for the lowest complexity. All other bit-true equivalent decoders with a reduced ATE-complexity show a higher ET-complexity. Thereby, some decoders even allow for a reduced energy per iteration in comparison to 1) but suffer from a significantly higher iteration period.

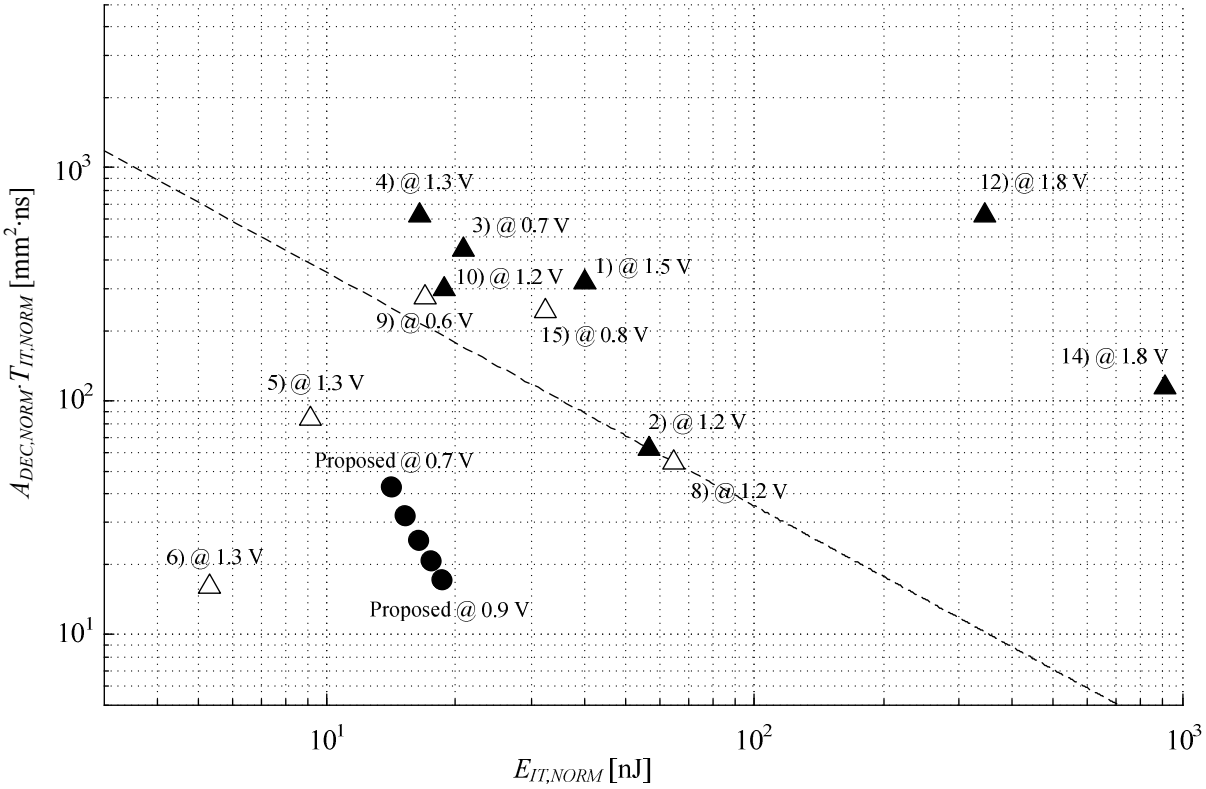


Fig. 6-16 ATE-complexity

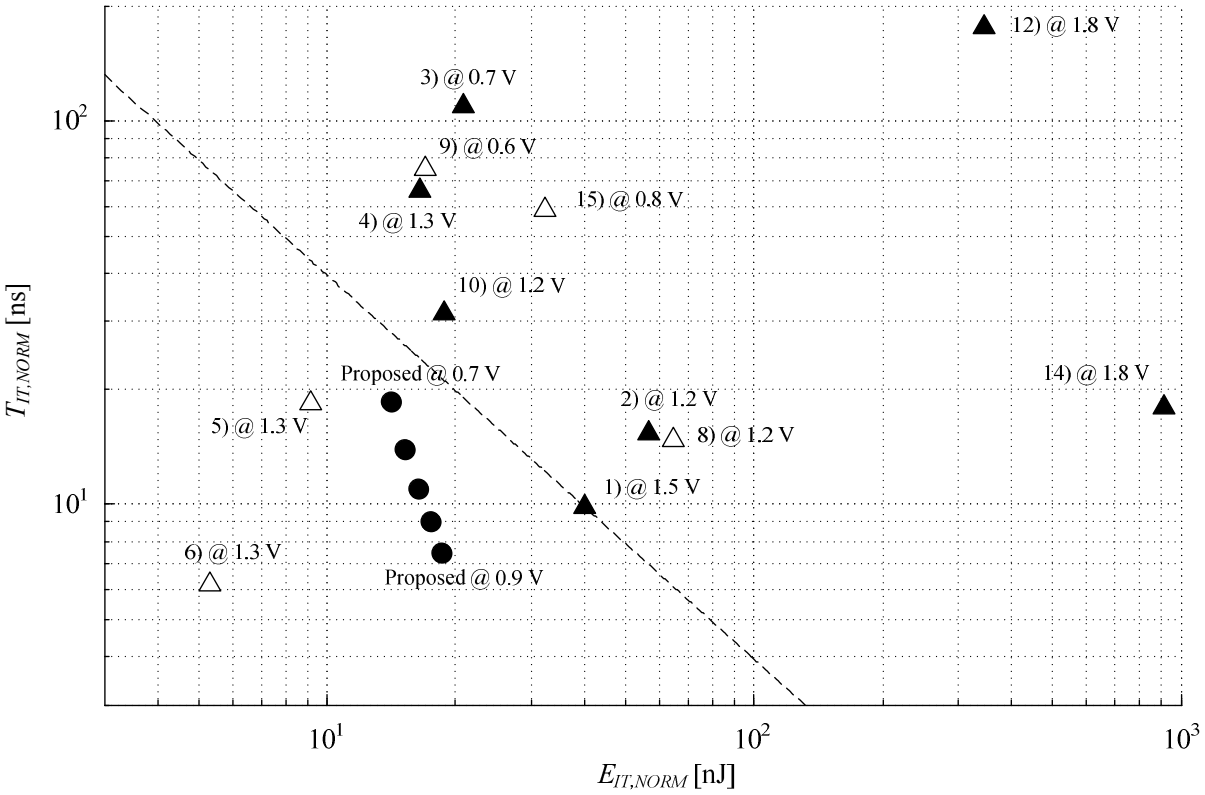


Fig. 6-17 ET-complexity

In comparison to the most ATE-efficient other decoder implementations known from literature [61] the proposed decoder reduces the ATE-complexity by more than one order of magnitude as can be seen in Fig. 6-16. This reduction splits up into a factor 1.8 in silicon area, two in iteration period and a factor of three in energy per iteration. In comparison to other decoders which allow for an energy per iteration lower than 20 nJ, the proposed decoder allows for a reduction in AT-complexity of more than a factor of 10.

In contrast to all other decoders the ATE-complexity is not achieved by trading-off silicon area with ET-complexity. As Fig. 6-17 reveals, the decoder developed in this work shows a smaller iteration period and energy per decoded bit than the decoder 1).

By varying the supply voltage energy can be traded-off with throughput. The resulting trade-off is also depicted in Fig. 6-16. By reducing the supply voltage the ATE-efficiency decreases, as the increase in iteration period is higher than the reduction in energy per iteration.

The advantage of the proposed decoder in power reduction using early termination becomes obvious when comparing the results e.g. with the decoder in [62] as is done in Fig. 6-18. This decoder bases on an approximate algorithm and achieves nearly the same energy per iteration (0.6V) but with a significantly increased iteration period by nearly a factor of 10.

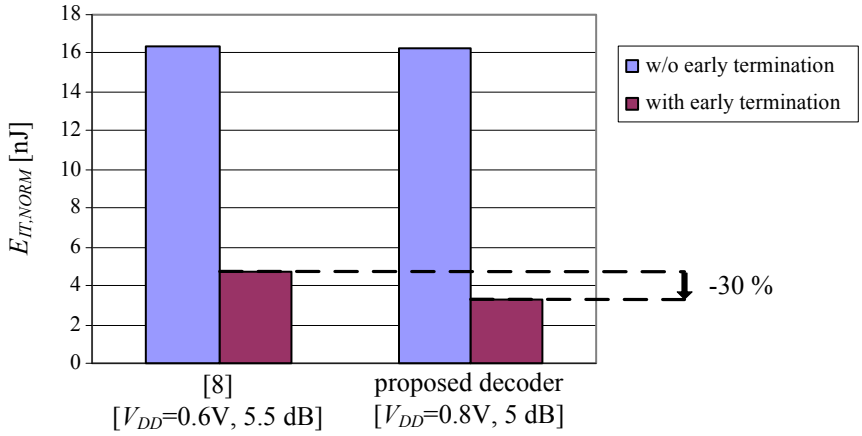


Fig. 6-18 Early termination

Although without early termination the energy per iteration is almost equal for the two decoders the proposed decoder shows a reduction of 30 % when comparing the energy with early termination enabled. This reduction is due to the global clock gating in which the clock of the whole decoder can be disabled.

7 Conclusion

The implementation of highly optimized high-throughput LDPC decoders requires an optimization of at best all design levels. In this work optimization strategies for various design levels have been discussed starting on algorithmic level with an analysis of fixed-point implementations and ending with a full-custom design approach on physical-implementation level. To allow for a quantitative analysis on the different design levels accurate area, timing, and energy cost models of high-throughput LDPC decoders have been derived.

On algorithmic level possible fixed-point Sum-Product and Min-Sum decoding algorithms have been analyzed. Thereby, it has been shown that a linear approximation of the transcendent Φ function in the Sum-Product algorithm allows for a very attractive decoding performance. On the other hand for Min-Sum-based decoder implementations it has been demonstrated that it is possible to choose a post-processing function which simultaneously allows for an optimal decoding performance and further reduces the decoder complexity.

Subsequently, two new decoder architectures have been introduced. Targeting very high-throughput applications the new hybrid-cell architecture allows for decoder throughputs close to the bit-parallel bit- and check-node architecture while reducing the silicon area and energy per iteration. Additionally, a systematic architectural analysis of bit- and check-node decoders revealed a new partially bit-serial decoder architecture. This architecture increases the decoder throughput in comparison to known bit-serial architectures significantly while maintaining the low area requirements. By adapting the cost models the efficiency of this new architecture has been shown for almost the complete range of code complexities.

For a further optimization on circuit and physical-implementation level a decoder has been realized in a 40-nm CMOS technology for an exemplary code. Therefore, the hardware-efficient post-processing function has been introduced into the partially bit-serial architecture and the decoder has been designed using a full-custom design approach. The resulting decoder has been compared to other implementations known from literature in a benchmarking. As the typically applied scaling rules underestimate the influence of the interconnect on the decoder throughput and energy per iteration fair scaling rules are derived based on the accurate area, timing, and energy cost models.

The benchmarking indicates that the performed optimization strategies proposed in the last seven years at best result in a trade-off between silicon area and ET-complexity. In contrast the proposed decoder reduces all three decoder metrics. Thereby, the silicon area and the iteration period is decreased by a factor of about two and the energy per iteration can even be

reduced by a factor larger than three. In total the proposed decoder allows for an ATE-complexity which is one order of magnitude smaller than for other high-throughput LDPC decoder implementations known from literature.

8 Abbreviations

HDL
Hardware description language 49

I_{MAX}
Maximum number of decoding iterations 19

$L(c_i)$
Received log-likelihood information of symbol i 10

$L(Q_i)$
Updated log-likelihood information of symbol i 10

$L(q_{ij})$
Extrinsic log-likelihood information of symbol i for parity check j 12

$L(r_i)$
A-posteriori information for symbol i 10

$L(r_{ij})$
A-posteriori information of symbol i from SISO decoder j, for LDPC decoder j indicates the parity check 10

l_{AVAIL}
Available Manhattan length in CMOS metal stack 29

l_{AVG}
Average Manhattan length of one interconnect line between one bit and one check node ... 25

l_{CNA}
Silicon area of check-node array logic 28

l_{DEC}
Macro side length of LDPC decoder 25

l_{DEC_P}
Macro side length of bit-parallel LDPC decoder 28

l_{DEC_S}
Macro side length of bit-serial LDPC decoder 37

l_{L_P}
Macro side length of bit-parallel LDPC decoder logic 30

l_{L_S}
Macro side length of bit-serial LDPC decoder logic 37

l_{MAX}
Maximum Manhattan length of one interconnect line between one bit and one check node ... 27

l_{REQ}
Required Manhattan length of all bit- and check-node connections in a bit-parallel LDPC decoder 25

LSB
Least-significant bit 23

$L_{TEMP\_i-1}(R_j)$
Intermediate result of $L(R_j)$ considering all symbols $i'<i$ 70

$L_{TEMP\_j-1}(Q_i)$
Intermediate result of $L(Q_i)$ considering all parity checks $j'<j$ 70

m
Blocklength of information vector x 9

M
Number of metal layers in CMOS metal stack 30

$M_{ROUTING}$
Number of metal layers in CMOS metal stack used for routing 29

MSB
Most-significant bit 16

n
Blocklength of code vector y 9

n_1, n_2
Normally distributed random variables 50

$N_{BN\_P\_FA}$
Number of full adders in bit-parallel bit node . 22

N_{CC_AVG}
Average number of active clock cycles in an LDPC decoder 104

$N_{CN\_P\_CS}$
Number of compare and swap elements in a bit-parallel check node 23

$N_{DEC\_PS\_REG}$
Number of registers in new partially bit-serial LDPC decoder 89

$N_{DEC\_S\_REG}$

9 Bibliography

[1] R. Gallager: "Low-density parity-check codes," *Information Theory, IRE Transactions on,* vol. 8, pp. 21-28, 1962.

[2] C. Sae-Young, G. D. Forney, Jr., et al.: "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *Communications Letters, IEEE,* vol. 5, pp. 58-60, 2001.

[3] A. Viterbi: "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *Information Theory, IEEE Transactions on,* vol. 13, pp. 260-269, 1967.

[4] C. Berrou, A. Glavieux, et al.: "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on*, 1993, pp. 1064-1070 vol.2.

[5] A. J. Blanksby, C. J. Howland: "A 690-mW 1-Gb/s 1024-b, Rate-1/2 Low-Density Parity-Check Code Decoder," *Solid-State Circuits, IEEE Journal of,* vol. 37, pp. 404-412, 2002.

[6] IEEE: "IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications," *IEEE Std 802.3an-2006 (Amendment to IEEE Std 802.3-2005),* pp. 0_1-167, 2006.

[7] V. Oksman, S. Galli: "G.hn: The new ITU-T home networking standard," *Communications Magazine, IEEE,* vol. 47, pp. 138-145, 2009.

[8] IEEE: "802.11n. Wireless LAN Medium Access Control and Physical Layer specifications: Enhancements for Higher Throughput. IEEE P802.16n/D1.0, Mar 2006," 2006.

[9] IEEE: "802.16e. Air Interface for Fixed and Mobile Broadband Wireless Access Systems. IEEE P802.16e/D12 Draft, oct 2005," 2005.

[10] "European Telecommunications Standards Institude (ETSI). Digital Video Broadcasting (DVB) Second generation framing structure for broadband satellite applications; EN 302 307 V1.1.1 www.dvb.org."

[11] W. Zhang, Y. Guan, et al.: "An Introduction of the Chinese DTTB Standard and Analysis of the PN595 Working Modes," *Broadcasting, IEEE Transactions on,* vol. 53, pp. 8-13, 2007.

[12] R. L. Galbraith, T. Oenning, et al.: "Architecture and Implementation of a First-Generation Iterative Detection Read Channel," *Magnetics, IEEE Transactions on,* vol. 46, pp. 837-843.

[13] M. P. C. Fossorier, M. Mihaljevic, et al.: "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *Communications, IEEE Transactions on,* vol. 47, pp. 673-680, 1999.

[14] C. Jinghu, M. P. C. Fossorier: "Near optimum universal belief propagation based decoding of low-density parity check codes," *Communications, IEEE Transactions on,* vol. 50, pp. 406-414, 2002.

[15] A. Darabiha, A. C. Carusone, et al.: "A 3.3-Gbps Bit-Serial Block-Interlaced Min-Sum LDPC Decoder in 0.13-µm CMOS," in *Custom Integrated Circuits Conference, 2007. CICC '07. IEEE*, 2007, pp. 459-462.

[16] T. Mohsenin, D. Truong, et al.: "Multi-Split-Row Threshold decoding implementations for LDPC codes," in *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, 2009, pp. 2449-2452.

[17] C.-C. Lin, K.-L. Lin, et al.: "A 3.33Gb/s (1200,720) low-density parity check code decoder," in *Solid-State Circuits Conference, 2005. ESSCIRC 2005. Proceedings of the 31st European*, 2005, pp. 211-214.

[18] M. Karkooti, P. Radosavljevic, et al.: "Configurable, High Throughput, Irregular LDPC Decoder Architecture: Tradeoff Analysis and Implementation," in *Application-specific Systems, Architectures and Processors, 2006. ASAP '06. International Conference on*, 2006, pp. 360-367.

[19] T. Brandon, R. Hang, et al.: "A scalable LDPC decoder ASIC architecture with bit-serial message exchange," *Integr. VLSI J.,* vol. 41, pp. 385-398, 2008.

[20] M. Cocco, J. Dielissen, et al.: "A scalable architecture for LDPC decoding," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, 2004, pp. 88-93 Vol.3.

[21] A. Darabiha, A. C. Carusone, et al.: "Multi-Gbit/sec low density parity check decoders with reduced interconnect complexity," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, 2005, pp. 5194-5197 Vol. 5.

[22] Y. Engling, N. Borivoje: "A 1.1-Gb/s 4092-bit Low-Density Parity-Check Decoder," in *Asian Solid-State Circuits Conference, 2005*, 2005, pp. 237-240.

[23] P. Urard, E. Yeo, et al.: "A 135Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes," in *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, 2005, pp. 446-609 Vol. 1.

[24] T. Ishikawa, K. Shimizu, et al.: "High-throughput decoder for low-density parity-check code," in *Design Automation, 2006. Asia and South Pacific Conference on*, 2006, p. 2 pp.

[25] M. M. Mansour, N. R. Shanbhag: "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *Solid-State Circuits, IEEE Journal of,* vol. 41, pp. 684-698, 2006.

[26] T. Brack, M. Alles, et al.: "Low Complexity LDPC Code Decoders for Next Generation Standards," in *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*, 2007, pp. 1-6.

[27] K. Se-Hyeon, P. In-Cheol: "Loosely coupled memory-based decoding architecture for low density parity check codes," *Circuits and Systems I: Regular Papers, IEEE Transactions on,* vol. 53, pp. 1045-1056, 2006.

[28] K. K. Gunnam, G. S. Choi, et al.: "VLSI Architectures for Layered Decoding for Irregular LDPC Codes of WiMax," in *Communications, 2007. ICC '07. IEEE International Conference on*, 2007, pp. 4542-4547.

[29] R. Swamy, S. Bates, et al.: "Design and Test of a 175-Mb/s, Rate-1/2 (128,3,6) Low-Density Parity-Check Convolutional Code Encoder and Decoder," *Solid-State Circuits, IEEE Journal of,* vol. 42, pp. 2245-2256, 2007.

[30] S. Yang, M. Karkooti, et al.: "VLSI Decoder Architecture for High Throughput, Variable Block-size and Multi-rate LDPC Codes," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, 2007, pp. 2104-2107.

[31] M. Bimberg, M. B. S. Tavares, et al.: "A High-Throughput Programmable Decoder for LDPC Convolutional Codes," in *Application -specific Systems, Architectures and Processors, 2007. ASAP. IEEE International Conf. on*, 2007, pp. 239-246.

[32] C. Chih-Lung, L. Kao-Shou, et al.: "A 11.5-Gbps LDPC decoder based on CP-PEG code construction," in *ESSCIRC, 2009. ESSCIRC '09. Proceedings of*, 2009, pp. 412-415.

[33] S. Jin, W. Zhongfeng, et al.: "Multi-Gb/s LDPC Code Design and Implementation," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 17, pp. 262-268, 2009.

[34] Z. Zhang, L. Dolecek, et al.: "Low error rate LDPC decoders," in *Signals, Systems and Computers, 2009 Conference Record of the Forty-Third Asilomar Conference on*, 2009, pp. 1278-1282.
[35] B. Xiang, R. Shen, et al.: "An Area-Efficient and Low-Power Multirate Decoder for Quasi-Cyclic Low-Density Parity-Check Codes," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. PP, pp. 1-1, 2009.
[36] H. Shuangqu, X. Bo, et al.: "A flexible architecture for multi-standard LDPC decoders," in *ASIC, 2009. ASICON '09. IEEE 8th International Conference on*, 2009, pp. 493-496.
[37] J. Nan, P. Kewu, et al.: "High-Throughput QC-LDPC Decoders," *Broadcasting, IEEE Transactions on,* vol. 55, pp. 251-259, 2009.
[38] X.-Y. Shih, C.-Z. Zhan, et al.: "A 52-mW 8.29mm^2 19-mode LDPC decoder chip for Mobile WiMAX applications," in *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*, 2009, pp. 121-122.
[39] Z. Zhengya, V. Anantharam, et al.: "An Efficient 10GBASE-T Ethernet LDPC Decoder Design With Low Error Floors," *Solid-State Circuits, IEEE Journal of,* vol. 45, pp. 843-855.
[40] M. Korb, T. G. Noll: "Towards a Scaling Model for Fair Benchmarking of LDPC Decoder Architectures," *www.eecs.rwth-aachen.de*.
[41] M. Korb, T. G. Noll: "Area and Latency Optimized High-Throughput Min-Sum Based LDPC Decoder Architectures," in *ESSCIRC*, 2009, pp. 408-411.
[42] D. E. Knuth: *The Art of Computer Programming Volume 3 Sorting and Searching*: Addison-Wesley, 1973.
[43] C. Henning, T. G. Noll: "Architecture and implementation of a bitserial sorter for weighted median filtering," in *Custom Integrated Circuits Conference, 1998. Proceedings of the IEEE 1998*, 1998, pp. 189-192.
[44] D. MacKay: "http://www.inference.phy.cam.ac.uk/mackay/codes/data.html." vol. 2009.
[45] M. Korb, T. G. Noll: "LDPC decoder area, timing, and energy models for early quantitative hardware cost estimates," in *System on Chip (SoC), 2010 International Symposium on*, 2010, pp. 169-172.
[46] R. Ho, K. W. Mai, et al.: "The future of wires," *Proceedings of the IEEE,* vol. 89, pp. 490-504, 2001.
[47] "http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/." vol. 2008.
[48] Z. Zhengya, L. Dolecek, et al.: "Investigation of Error Floors of Structured Low-Density Parity-Check Codes by Hardware Emulation," in *Global Telecommunications Conference, 2006, IEEE*, 2006, pp. 1-6.
[49] T. Richardson: "Error floors of LDPC codes," in *Allerton Conference on Communications, Control, and Computing*, 2003.
[50] D. U. Lee, J. D. Villasenor, et al.: "A hardware Gaussian noise generator using the Box-Muller method and its error analysis," *Computers, IEEE Transactions on,* vol. 55, pp. 659-671, 2006.
[51] G. Box, M. Muller: "A Note on the Generation of Random Normal Deviates," *Annals Math. Statistics,* vol. 29, pp. 610-611 1958
[52] P. L'Ecuyer: "Maximally Equidistributed Combined Tausworthe Generators," *Math. Computation,* vol. 65, pp. 203-213, 1996
[53] B. Sklar: *Digital Communication, Fundamentals and Applications*, 2 ed.: Prentice Hall, 2002.
[54] "www.eve-team.com/products/datasheets/ax.php," in *White Paper ZeBU-AX*. vol. 2009.

[55] S. L. Howard, C. Schlegel, et al.: "A degree-matched check node approximation for LDPC decoding," in *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*, 2005, pp. 1131-1135.
[56] G. Lechner, T. Pedersen, et al.: "EXIT Chart Analysis of Binary Message-Passing Decoders," in *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, 2007, pp. 871-875.
[57] M. Korb, T. G. Noll: "Area- and energy-efficient high-throughput LDPC decoders with low block latency," in *ESSCIRC (ESSCIRC), 2011 Proceedings of the*, 2011, pp. 75-78.
[58] Cadence: "ICCraftsman," 11.2.41 ed.
[59] T. Mohsenin, B. M. Baas: "High-Throughput LDPC Decoders Using A Multiple Split-Row Method," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, 2007, pp. II-13-16.
[60] O. Weiss, M. Gansen, et al.: "A flexible datapath generator for physical oriented design," in *ESSCIRC*, 2001, pp. 393-396.
[61] Z. Zhang, V. Anantharam, et al.: "A 47 Gb/s LDPC Decoder With Improved Low Error Rate Performance," in *VLSI Circuits, 2009 Symposium on*, 2009, pp. 286-287.
[62] A. Darabiha, A. Chan Carusone, et al.: "Power Reduction Techniques for LDPC Decoders," *Solid-State Circuits, IEEE Journal of,* vol. 43, pp. 1835-1845, 2008.
[63] T. Mohsenin, B. Baas: "A Split-Decoding Message Passing Algorithm for Low Density Parity Check Decoders," *Journal of Signal Processing Systems,* pp. 1-17, 2010.

Curriculum vitae

Name	Matthias Korb
Date of birth	12th of November 1979
City of birth	Arnsberg, Germany

Education

1986 – 1990	Heinrich-Knoche-Grundschule, Arnsberg, Germany
1990 – 1999	Graf-Gottfried-Gymnasium, Arnsberg, Germany
05/1999	General Qualification for University Entrance
2000 - 2006	RWTH Aachen University, Germany Electrical Engineering and Information Technology Specialization: Information and Communication Engineering
04/2006	Diploma Electrical Engineering and Information Technology
since 2003	Fern Universitaet Hagen, Germany (Open University) Business Administration and Economics
09/2006	Prediploma Business Administration and Economics

Professional Experience

1999 – 2000	Community Service Environmental Office Arnsberg, Germany
since 2006	Research and Teaching Assistant Institute for Electrical Engineering and Computer Systems (EECS), RWTH Aachen University, Germany