FlexiChaP: A Reconfigurable ASIP for Convolutional, Turbo, and LDPC Code Decoding

Matthias Alles, Timo Vogt, Norbert Wehn
University of Kaiserslautern
Erwin-Schroedinger-Str.
67663 Kaiserslautern, Germany
Email: {alles, vogt, wehn}@eit.uni-kl.de

***Abstract*—Future mobile and wireless communication networks require flexible modem architectures to provide seamless services between different network standards. In this paper we focus on the outer modem which has to support various advanced channel coding techniques like convolutional codes, turbo codes, and low-density parity-check (LDPC) codes. We present an application-specific instruction-set processor (ASIP) which supports convolutional codes, binary/duo-binary turbo codes, and LDPC codes. Special emphasis is put on the support of LDPC codes. The ASIP consists of a special pipeline which is completely optimized for channel decoding. Logic synthesis yields an overall area of 0.62 mm^2 for this ASIP in a 65 nm low power technology. Payload throughputs of, e.g., up to 257 Mbps are possible at 400 MHz for the WiMAX and WiFi LDPC codes, outperforming existing ASIP solutions for LDPC decoding by an order of magnitude.**

I. Introduction

Next generation mobile communication networks have to support various standards to provide seamless services and heterogeneous interoperability. Thus, flexibility in modem architectures becomes a dominant aspect. A promising approach in this direction is *software defined radio* (SDR).

Recently, various platforms for SDR were published, e.g., [1][2][3]. Most of these platforms target the signal processing tasks in the inner modem, i.e., filtering, modulation, and channel estimation. These algorithms have to perform a huge amount of operations per sample with a high degree of data parallelism. Thus, these SDR platforms are multi-processor systems with SIMD (single-instruction multiple-data) vector processing engines. In this paper we focus on channel decoding which is the central processing part in the outer modem. Table I gives an overview of the channel codes used in current standards. As can be see, convolutional codes (CC), binary turbo codes (bTC), duo-binary turbo codes (dbTC), and low-density parity-check codes (LDPC) are established channel coding schemes. Especially the decoding algorithms of turbo codes and LDPC codes have a very high computational complexity. [1] reports that channel decoding contributes approximately 40 % to the total computational complexity of the physical layer of a UMTS or a WiFi 802.11a system, depending on the implementation platform. Similar results have been obtained in [4][5]. The aforementioned SDR platforms are not well suited for channel decoding algorithms since they substantially differ from the algorithms in the inner modem. They are non-standard signal processing algorithms with non-standard arithmetics and word widths. Hence, other architectural approaches become mandatory to provide high performance, low energy, and high flexibility at the same time.

We favour an approach which is based on application-specific instruction-set processors (ASIP). These processors are fully optimized for channel decoding by completely customizing instruction set, pipeline, and memory structure to this task. Such an ASIP can be considered as a weakly programmable processor, since it is intended to perform only channel decoding algorithms. A "just enough" policy dominates the processor design. Flexibility is only added where it is required by the channel decoding tasks. In [6], we presented the **FlexiTreP** ASIP family which is based on this design methodology. This processor family supports trellis based channel codes, i.e., convolutional, binary and duo-binary turbo codes for various standards. However, recent standards like IEEE802.11n (WiFi, [7]) or IEEE802.16e (WiMAX, [8]) use also LDPC codes. LDPC decoding substantially differs from trellis-based decoding algorithms.

In this paper we extend the FlexiTreP ASIP family:

- A new ASIP for decoding of structured LDPC codes (**FlexiProESL**: Flexible Processor Especially for Structures LDPC codes) is presented. This processor targets standardized LDPC codes for, e.g., WiMAX and WiFi.
- The FlexiTreP and FlexiProESL cores are merged into a single ASIP, named **FlexiChaP**, a Flexible Channel coding Processor, which is capable to support convolutional codes, binary/duo-binary turbo codes, and structured LDPC codes. To the best of our knowledge this is the first ASIP supporting all important channel decoding techniques.

We can find some processor approaches for LDPC decoding in the literature. However, they lack of high throughputs. E.g., the ASIP presented in [9] yields a throughput of only 15 Mbps at 400 MHz for a single iteration. In [10] the SDR SODA architecture [1] was enhanced for LDPC decoding resulting in a throughput of 30.4 Mbps for WiMAX LDPC decoding at 400 MHz. An ASIP architecture proposal [11] uses a SIMD approach with 96 data processing units. But despite the high parallelism throughputs of only about 50 Mbps are reported for a WiFi code at 20 iterations and a clock frequency of

Standard	Codes	States	Rates	Infobits	Throughput
GSM	CC	16,64	1/4...1/2	...876	...12 kbit/s
EDGE	CC	64	1/4...1/2	...870	...384 kbit/s
UMTS	CC	256	1/4...1/2	...504	...32 kbit/s
	bTC	8	1/3	...5114	...2 Mbit/s
CDMA2000	CC	256	1/6...1/2	...744	...28 kbit/s
	bTC	8	1/5...1/2	...20730	...2 Mbit/s
HSDPA	bTC	8	1/2...3/4	...5114	...14.4 Mbit/s
LTE	bTC	8	1/3	...6144	...150 Mbit/s
DAB	CC	64	1/4	none	...1.1 Mbit/s
DVB-H	CC	64	1/2...7/8	1624	...32 Mbit/s
DVB-T	CC	64	1/2...7/8	1624	...32 Mbit/s
DVB-RCT	dbTC	8	1/2, 3/4	...648	...31 Mbit/s
IEEE802.11a/g	CC	64	1/2...3/4	...4095	...54 Mbit/s
IEEE802.11n	CC	64	1/2...3/4	...4095	...300 Mbit/s
	LDPC	–	1/2...5/6	...1620	...300 Mbit/s
IEEE802.16e	CC	64	1/2...5/6	...864	...75 Mbit/s
	dbTC	8	1/2...3/4	...4800	...75 Mbit/s
	LDPC	–	1/2...5/6	...1920	...75 Mbit/s

TABLE I
SELECTION OF STANDARDS AND CHANNEL CODES

about 400 MHz. Our ASIP fully exploits the inherent parallelism of the LDPC decoding algorithm, yielding a very high throughput. A custom SIMD LDPC processor pipeline with 27 data processing units is presented. The pipeline employs an elaborated memory configuration that is especially suited for LDPC decoding. The resulting ASIP offers all the flexibility that is required to support current and future standards. Low latency and high throughputs of up to 257 Mbps at 400 MHz are achievable.

The rest of the paper is structured as follows: Section II introduces LDPC codes and their decoding algorithm. In Section III the instruction set and pipeline of the new LDPC ASIP are presented. Section IV shows synthesis results of different instances of the ASIP. Finally, Section V will conclude the paper.

II. LDPC CODES

LDPC codes [12] were already introduced in 1962 by Gallager. Like turbo codes they allow for a near optimum error correction performance. LDPC codes are linear block codes defined by a sparse parity check matrix H of size $M \times N$, see Figure 1a). For binary LDPC codes a valid codeword $\vec{x}$ has to satisfy Equation 1 in a modulo-2 arithmetic:

$$H\vec{x}^T = \vec{0} \qquad \forall \vec{x} \in \mathcal{C}, \tag{1}$$

with $\mathcal{C}$ the set of valid codewords. A column in H is associated to a codeword bit, a row corresponds to a parity check. A non-zero element in a row means that the corresponding bit contributes to this parity check. Typically, 2% or less of the elements of H are non-zero.

The complete code can be best visualized by a Tanner graph, a graphical representation of the associations between code bits and parity checks. Each column of H corresponds to a variable node (VN) and represents one code bit, each row of H corresponds to a check node (CN) and represents one parity check, respectively. Edges connect variable and check nodes according to the non-zero elements of the parity check

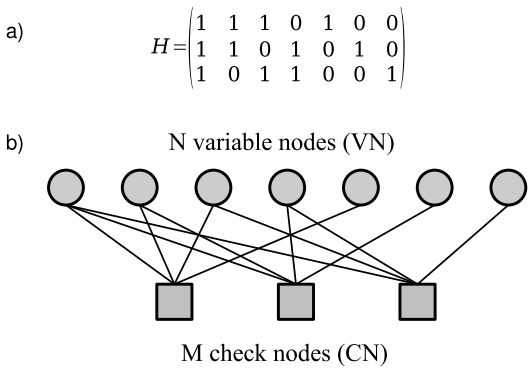

Fig. 1. Tanner graph of an irregular LDPC code

matrix. Figure 1b) shows the corresponding Tanner graph to the parity check matrix in Figure 1a) of an LDPC code with $N = 7$ variable nodes and $M = 3$ check nodes. The resulting code rate is $R = (N - M)/N = 4/7$.

A. LDPC Decoding

LDPC codes can be decoded using the Sum-Product algorithm (SPA) [12]. Soft information is exchanged iteratively between variable and check nodes. To reduce the implementation complexity the SPA was transformed into the logarithmic domain.

In detail the SPA in the logarithmic domain looks as follows: Two sets are defined that represent the connections in the Tanner graph between the two types of nodes. The set of check nodes connected to the variable node n, $n \in \{0, ..., N-1\}$ is given by:

$$\mathcal{M}(n) = \{m | m \in \{0, ..., M-1\} \wedge H_{mn} \neq 0\}. \tag{2}$$

Respectively, the set of variable nodes connected to the check node m, $m \in \{0, ..., M-1\}$ is given by:

$$\mathcal{N}(m) = \{n | n \in \{0, ..., N-1\} \wedge H_{mn} \neq 0\}. \tag{3}$$

With these sets we describe the iterative decoding algorithm: The variable node n is initialized with the corresponding log-likelihood ratio (LLR) of the received bit λ_n^{ch}. Next, messages are propagated from the variable nodes to the check nodes via the edges of the Tanner graph. For the first iteration the messages sent by the variable node n via its edges to the check node m with $m \in \mathcal{M}(n)$ is the corresponding LLR: $\lambda_{n \to m} = \lambda_n^{ch}$. The check node m computes new messages for the variable nodes. Due to the high implementation complexity of the optimal belief propagation algorithm, suboptimal algorithms are used for the check node implementation. We use the normalized Min-Sum algorithm [13] where only the two smallest magnitudes are used:

$$\lambda_{m \to n} = \alpha \times \left(\prod_{n' \in \mathcal{N}(m) \setminus n} sgn\left(\lambda_{n' \to m}\right) \right) \times \min_{n' \in \mathcal{N}(m) \setminus n} \left(|\lambda_{n' \to m}| \right). \tag{4}$$

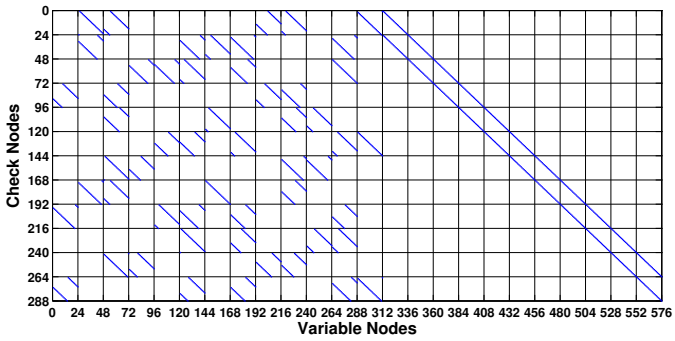

Fig. 2. A parity check matrix of the WiMAX code with $P = 24$.

When using the extrinsic scaling factor α, a communications performance close to the optimal algorithm is possible for high code rates $R \geq 2/3$. Typically, α is chosen as 0.75 or 0.875, since these values are easy to implement.

Then, the variable nodes compute an overall estimation of the decoded bit:

$$\Lambda_n = \lambda_n^{ch} + \sum_{m' \in \mathcal{M}(n)} \lambda_{m' \to n}. \tag{5}$$

The sign of Λ_n can be understood as the hard decision on the decoded bit. New messages $\lambda_{n \to m}$ are then computed following the extrinsic principle:

$$\lambda_{n \to m} = \lambda_n^{ch} + \sum_{m' \in \mathcal{M}(n) \setminus m} \lambda_{m' \to n} = \Lambda_n - \lambda_{m \to n}. \tag{6}$$

The decoding process is stopped after a maximum number of iterations or earlier if the parity check is satisfied.

B. Structured LDPC Codes

Fully parallel decoder implementations instantiate all nodes of the Tanner graph. This approach lacks flexibility and is furthermore infeasible for high block lengths. LDPC decoders are therefore implemented in a partly parallel fashion, where only a subset of nodes in the Tanner graph is implemented as hardware units. Low latency and high throughput are obtained by exchanging a huge amount of messages between variable and check nodes per clock cycle. A random connectivity between variable and check nodes poses big challenges for an efficient hardware implementation. Complex connectivity networks become mandatory to allow for a flexible and parallel message exchange, resolving occuring memory access conflicts. Thus, LDPC codes defined by standards are based on so called structured LDPC codes [14]. The matrices of these codes are composed of cyclically shifted identity matrices of size $P \times P$. Figure 2 shows the structured binary parity check matrix of the WiMAX code with $N = 576$ variable nodes, $M = 288$ check nodes, and $P = 24$. These codes allow for an efficient implementation of the connectivity between up to P variable and P check nodes. Memory access conflicts are avoided and a low complexity logarithmic barrel shifter is sufficient as connectivity network for a given P. Furthermore, storing the parity check matrix is simplified. Only the positions of the cyclically shifted identity matrices and the corresponding shift value have to be stored.

Parameter	WiMAX #	WiMAX min	WiMAX max	WiFi #	WiFi min	WiFi max
Block lengths	19	576	2304	3	648	1944
Submatrix sizes	19	24	96	3	27	81
Code rates	4	1/2	5/6	4	1/2	5/6
CN degrees	7	6	20	9	7	22
VN degrees	4	2	6	8	2	12
Edges	90	1824	8448	8	2376	7128
Codes	114			12		

TABLE II
PARAMETERS OF WIMAX AND WIFI LDPC CODES

C. Layered Decoding

Updating the variable and check nodes can be done with a two-phase scheduling: In the first phase all variable nodes are updated, in the second phase all check nodes, respectively. To improve the communications performance for a given number of iterations turbo decoding message passing (TDMP) [15], which is also known as layered decoding, shuffled decoding or Gauss-Seidel iterations, is used. The idea is to take intermediate results of the calculations into account still in the same iteration. This technique can be applied in partly parallel architectures in which not all nodes are processed at the same time. Consider we are processing one check node. After computing new LLRs, they are sent back to the connected variable nodes immediately. These variable nodes update their outgoing edges, such that the other check nodes connected to these variable nodes will receive already updated results. With this technique it is possible to reduce the number of iterations by up to 50 % at a fixed communications performance. The achieved gain in convergence speed can thus be used to reduce the latency or to increase the throughput of the decoding process.

III. LDPC ASIP

A. Decoder Requirements

As mentioned in the introduction, the FlexiTreP ASIP family [6] supports decoding of binary turbo codes, duo-binary turbo codes, and convolutional codes but lacks LDPC support. Thus, it is necessary to extend the FlexiTreP pipeline by LDPC decoding capabilities, in order to support all channel coding schemes defined by e.g. WiMAX and WiFi.

Standardized LDPC codes like WiMAX, WiFi, and DVB-S2 are structured ones. Table II gives an overview of the WiMAX and WiFi LDPC code parameters. Many block lengths, code rates, and sub-matrix sizes need to be supported. Thus, flexible architectures are mandatory.To obtain high throughput and low latency the inherent parallelism of structured LDPC codes has to be exploited and needs to be combined with layered decoding.Therefore, a SIMD processor pipeline with a special memory organization is the architecture of choice.

Due to the fundamental differences between turbo/convolutional and LDPC decoding, logic reuse is barely possible between the existing FlexiTreP functionality and the new LDPC functionality. RAM sharing is key for an efficient ASIP implementation. About 75% of the FlexiTreP area are utilized

by the RAMs in a 65 nm technology (0.11 mm^2 logic vs. 0.31 mm^2 memory). The goal is thus to share as much RAM as possible between the new LDPC functionality and the existing FlexiTreP functionality.

B. Instruction Set

As mentioned in the previous section, a highly parallel SIMD processor with an elaborated memory architecture is key for high throughput and low latency decoding of LDPC codes. Therefore, we process a whole sub-matrix of the parity check matrix with a single instruction. P messages of the Tanner graph are processed by one instruction, as shown in the following.

```
.text
l.subm 24                   ; reconfigure networks and
                            ; set submatrix size to 24
l.diag 0, s=1,  a=1         ; process submatrix
l.diag 0, s=6,  a=2         ; s determines shift offset
l.diag 0, s=11, a=8         ; a determines address
l.diag 0, s=4,  a=9
l.diag 0, s=23, a=12
l.diag 1, s=0,  a=13        ; last edge of check node

l.diag 0, s=18, a=1         ; new check node begins
l.diag 0, s=19, a=5
l.diag 0, s=5,  a=6
l.diag 0, s=22, a=7
l.diag 0, s=21, a=11
l.diag 0, s=0,  a=13
l.diag 1, s=0,  a=14        ; last edge of check node
...
l.diag 0, s=23, a=12
l.diag 1, s=0,  a=23        ; last edge of check node

l.pchk it=10                ; perform parity check
nop
PD                          ; power down
```

Listing 1. WiMAX assembler program

Listing 1 gives the assembler program for decoding the WiMAX LDPC code as given by the parity check matrix in Figure 2. The instruction `l.subm` is used to reconfigure the ASIP to the corresponding sub-matrix size P. According to this instruction the shifting networks are adjusted. With `l.diag` a complete sub-matrix is processed. The s (shift) and a (address) values determine the connection between variable nodes and check nodes: a=1 means that we process the variable nodes 24 to 47, s=1 means that these 24 values have to be cyclically shifted by one to establish the correct connection to the check nodes. Setting the first parameter of `l.diag` to 1 indicates that this is the last sub-matrix belonging to the check nodes that are being processed. After this instruction the check nodes start to output newly calculated messages. These messages are shifted into the original order and stored at the same address where they were read from.

After processing all sub-matrices of the parity check matrix, the instruction `l.pchk` tests, whether all parity checks were satisfied. In this case, i.e., decoding was successful, the ASIP powers down till a new block is available for decoding. If the parity check fails and less than the given number of iterations were performed, a new iteration is started. Therefore, the program counter is set to the second instruction.

FlexiTreP RAMs	#	depth	width	Usage in FlexiProESL
Program Memory	1	512	24	Program memory
Interleaver Table	1	6144	13	
Channel Values	2	4096	12	Message memory
Apriori Values	4	2048	8	
State Metrics (dp)	2	128	96	Channel values
Hard Decisions	1	196	32	A/S bypass

TABLE III
TYPICAL RAM CONFIGURATION USED IN FLEXITREP

Different block lengths, variable node and check node degrees can be easily realized with this instruction set. Furthermore, different sub-matrix sizes are supported due to a reconfigurable shifting network. So the ASIP provides a high flexibility and can decode the LDPC codes of WiMAX and WiFi standards.

C. LDPC ASIP Pipeline

As mentioned before, the primary goal is to maximize the memory reuse between FlexiTreP and the LDPC functionality. We therefore derive the constraints for the LDPC ASIP pipeline from the given memory organization in FlexiTreP. Table III gives an overview of the memories used in a typical FlexiTreP configuration. The state metric memories are the only dual-ported (dp) ones. Furthermore, they offer the highest memory bandwidth. For LDPC decoding they are suitable for the channel memories, storing the APP values of the variable nodes as given by Equation 5. When using both instances of the memory in parallel (192 bits bandwidth) and a 7 bit quantization, up to $P = 192/7 = 27$ APP values can be read per clock cycle. Because of the depth of the memories, block lengths of up to $N = 27 * 128 = 3456$ are supported, which is sufficient for the WiMAX and WiFi LDPC codes. Using a quantization of 5 bits for the messages that are exchanged between variable nodes and check nodes, the message memories need to offer a bandwidth of 5*27 = 135 bits. Since there is no other RAM in FlexiTreP that offers such a high bandwidth, memory partitioning becomes mandatory. We reorganized the channel RAMs such that the bandwidth requirements could be fulfilled.

According to the RAM organization, the ASIP can decode sub-matrix sizes of up to 27×27. For decoding LDPC codes with sub-matrix sizes higher than 27, the parity check matrix can be rearranged according to [16].

Figure 3 shows the proposed data path of FlexiProESL with twelve pipeline stages. Fetch (FE) and decode (DC) are the first two pipeline stages. In case an `l.diag` instruction is decoded, shift and address values are stored in pipeline registers. An `l.subm` instruction sets the parallelism register. Furthermore, control flags are set according to the instructions. In the address stage (AD) the APP values of the variable nodes in the channel memory (CV MEM) are addressed. At the same time the shift value for shifting back and the read address are stored in a RAM (A/S Bypass). The shifter stage (SH) is responsible for shifting the APP values according to the cyclically shift value defined by the `l.diag` operation. A reconfigurable barrel shifter with 27 inputs and outputs

performs this task. The extrinsic messages for the check nodes are calculated in the EXT stage. Therefore, the message of the previous iteration has to be subtracted from the APP value, see Equation 6. In the first iteration the extrinsic information equals the channel value.The extrinsic information is saturated to 5 bits and also stored in a bypass FIFO. In the check node input stage (CNI) the minimum search and a parity check are performed as required for the Min-Sum algorithm. The check node output stage (CNO) computes the outgoing messages of the check nodes. Calculation of new APP values is done in the following stage by taking into account the bypassed extrinsic information and the messages of the check node. These APP values are saturated to 7 bits (SAT) and shifted inverse (SHI) to the original order. Finally, in the write back stage (WB) the new APP values are written back to the channel memory.

Two cases exist, in which stalling becomes mandatory:

1) In case of varying check node degrees.
2) In some cases when accessing the same address in the CV MEM.

In case 1) it is possible that a check node with lower degree follows a check node with higher degree. Then, the stage CNO is still busy with generating new messages, but the stage CNI is already finished. In this case overwriting the determined values (MIN0_O, IND0_0, MIN1_O, and PAR_O) has to be prevented until the stage CNO has finished the processing of the previous check node. In case 2) it has to be assured that reading from an address in the CV MEM is only performed, when this address is not currently being processed, i.e., it was not read before without writing new results back. The number of stall cycles strongly depends on the code and can be reduced remarkably by rescheduling the check node processing and the processing sequence of edges inside one check node. We employed simulated annealing to reschedule the instructions.

IV. Results

The ASIP was modelled in the LISA language using CoWare's Processor Designer [17]. The design was verified with the generated VHDL and C++ simulation models. Synthesis was done with a 65 nm low power low leakage standard cell library and for Xilinx FPGAs. Table IV shows the results. Two ASIP instances with LDPC functionality were generated:

- The first one (FlexiProESL) provides only the LDPC functionality.
- The second one (FlexiChaP) provides full functionality, i.e., convolutional codes with the Viterbi algorithm (VA), binary/duo-binary turbo codes (bTC/dbTC), and LDPC codes.

For comparison reasons different instances of the FlexiTreP are listed as well.

The logic of FlexiProESL occupies an area of 0.11 mm^2, which is about the same as the FlexiTreP with full functionality. 425 MHz are possible for this processor. FlexiChaP requires an overall area of 0.23 mm^2. Due to the limited resource sharing, the logic area of FlexiChaP is approximately given by the sum of the area of FlexiProESL and FlexiTreP.

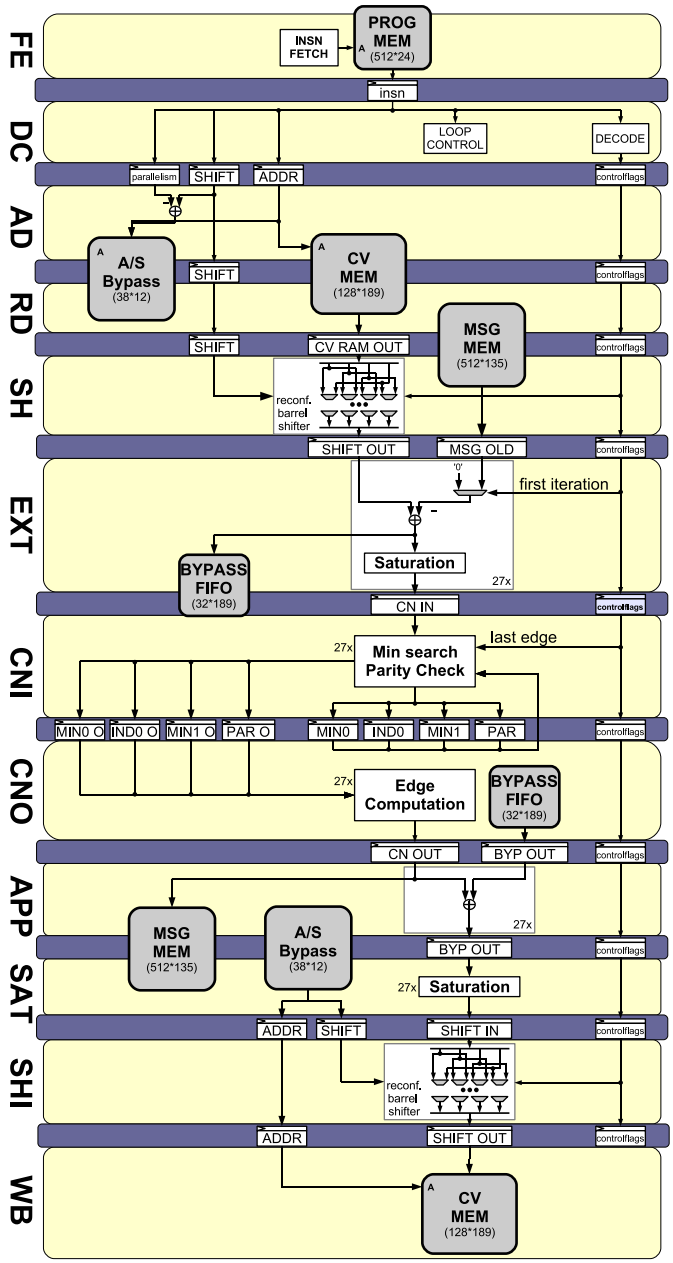

Fig. 3. Pipeline of the proposed LDPC ASIP (FlexiProESL)

The clock frequency is limited to 400 MHz because of the critical paths in turbo decoding. On an FPGA FlexiProESL allows for a clock frequency of 132 MHz, while FlexiChaP is limited to 109 MHz. In FlexiTreP 286 kbits or 0.31 mm^2 are typically needed for the memories. Because of the efficient RAM sharing as proposed in Section III-C the overall memory area increases only slightly. 0.39 mm^2 are required for the memories when adding LDPC support. The overall area thus increases from 0.42 mm^2 for FlexiTreP to 0.62 mm^2 for FlexiChaP.

With this RAM configuration it is possible to decode binary turbo codes with up to 6144 information bits, duo-binary turbo and convolutional codes with up to 8192 information bits ($R = 1/2$ for convolutional codes).

Name	Functionality				ASIC (65 nm, 1.10V, 120°C)		FPGA (Xilinx xc4vlx80-12)	
	bTC	dbTC	VA	**LDPC**	Size [μm^2]	Frequency [MHz]	Size [Slices]	Frequency [MHz]
FlexiTreP	X				74.391	450 (max)	4.207	135 (max)
FlexiTreP	X	X			88.966	415 (max)	5.494	117 (max)
FlexiTreP	X	X	X		109.320	400 (max)	7.012	109 (max)
FlexiProESL				**X**	113.099	425 (max)	8.076	132 (max)
FlexiChaP	X	X	X	**X**	232.346	400 (max)	14.495	109 (max)

TABLE IV
ASIC AND FPGA SYNTHESIS RESULTS FOR VARIOUS INSTANCES OF THE ASIP

Algorithm	Throughput/Mbps @ 400 MHz	Conditions
Viterbi	12 - 196	16 - 256 states
binary MAP	1.6 - 186	4 - 256 states
binary TC	18.6	4 - 16 states, 5 iterations
duo-binary TC	18.6 - 37.2	8 or 16 states, 5 iterations
LDPC WiMAX	27.7 - 237.8	10 - 20 iterations
LDPC WiFi	34-5 - 257.0	10 - 20 iterations

TABLE V
CORE PAYLOAD THROUGHPUT OF FLEXICHAP FOR CONVOLUTIONAL, TURBO AND LDPC CODES.

LDPC codes with a block length of up to $N = 3456$, a check node degree of up to 28 and up to 13824 edges are decodable. Thus, all LDPC codes in the WiMAX and WiFi standards are supported by the ASIP. Therefore, this RAM configuration of FlexiChaP allows for decoding of all codes in Table I except the long block lengths of the binary turbo code in CDMA2000. Note that the RAM configuration of FlexiChaP can be adapted to the current requirements.

Table V shows the obtained throughput results. For more information on the Viterbi, binary MAP, and binary/duo-binary turbo code throughputs please refer to [6]. As mentioned before, the number of stall cycles of FlexiChaP for LDPC decoding strongly depends on the order of the instructions. We could achieve payload throughputs of up to 237.8 Mbps for the WiMAX codes and up to 257.0 Mbps for the WiFi codes by rescheduling the instructions with simulated annealing. These numbers are obtained when using the largest codeword sizes, a code rate of $5/6$ and 10 iterations. Lowest throughputs are given for $N = 672$ for WiMAX and $N = 648$ for WiFi with $R = 1/2$ and 20 iterations. Note that more iterations are required for lower code rates to obtain reasonable communications performance results.

V. CONCLUSION

In this paper we presented two new ASIPs, namely FlexiProESL and FlexiChaP. FlexiProESL is an ASIP optimized for layered decoding of structured LDPC codes. FlexiChaP supports convolutional codes, binary/duo-binary turbo codes, and LDPC codes. Due to its reconfigurability and flexibility, it supports a vast number of communications standards. FlexiChaP requires an overall area of 0.62 mm^2 in a 65 nm low power technology. When running at 400 MHz we obtain throughputs for the LDPC part of up to 257 Mbps, outperforming other LDPC ASIPs by an order of magnitude.

ACKNOWLEDGMENT

This work has been supported by the Deutsche Forschungsgemeinschaft (DFG) within the Schwerpunktprogramm "Rekonfigurierbare Rechnersysteme".

REFERENCES

[1] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "SODA: A Low-power Architecture For Software Radio," in *Proc. 33rd International Symposium on Computer Architecture (ISCA'06)*, 2006, pp. 89–101.

[2] IMEC, "Scientific Report 2006: Software Defined Radio Flexible Air Interface," www.microelektronica.be/ wwwinter/mediacenter/en/SR2006/681340.html, 2006.

[3] J. Glossner, D. Iancu, M. Moudgill, G. Nacer, S. Jinturkar, and M. Schulte, "The Sandbridge SB3011 SDR Platform," in *Joint IST Workshop on Mobile Future and the Symposium on Trends in Communications (SympoTIC '06)*, 24-27 June 2006, pp. ii–v.

[4] C. Pan, N. Bagherzadeh, A. Kamalizad, and A. Koohi, "Design and Analysis of a Programmable Single-Chip Architecture for DVB-T Base-Band Receiver," in *Design, Automation and Test in Europe Conference and Exhibition, 2003*, 2003, pp. 468–473.

[5] M. Hosemann, R. Habendorf, and G. P. Fettweis, "Hardware-Software Codesign of a 14.4Mbit - 64 State - Viterbi Decoder for an Application-Specific Digital Signal Processor," in *Proc. IEEE Workshop on Signal Processing Systems 2003 (SIPS'03)*, Seoul, Korea, Aug. 2003.

[6] T. Vogt and N. Wehn, "A Reconfigurable Application Specific Instruction Set Processor for Convolutional and Turbo Decoding in a SDR Environment," in *Proc. Design, Automation and Test in Europe (DATE '08)*, Munich, Germany, Mar. 2008.

[7] IEEE 802.11n, "Wireless LAN Medium Access Control and Physical Layer specifications: Enhancements for Higher Throughput," IEEE P802.11n/D3.0, Sep. 2007.

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

[9] L. D. Noi, R. Martini, G. Masera, F. Quaglio, and F. Vacca, "ASIP design for partially structured LDPC codes," *Electronics Letters*, vol. 42, no. 18, pp. 1048–1049, 2006.

[10] S. Seo, T. Mudge, Y. Zhu, and C. Chakrabarti, "Design Analyis of LDPC Decoders for Software Define Radio," in *Proc. IEEE Workshop on Signal Processing (SIPS'07)*, Shanghai, China, October 2007, pp. 210–215.

[11] B. Bougard, R. Priewasser, L. V. der Perre, and M. Huemer, "Algorithm-Architecture Co-Design of a Multi-Standard FEC Decoder ASIP," in *ICT-MobileSummit 2008 Conference Proceedings*, Stockholm, Sweden, Jun. 2008.

[12] R. G. Gallager, "Low-Density Parity-Check Codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[13] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and X.-Y. Hu, "Reduced-Complexity Decoding of LDPC Codes," *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[14] E. Boutillon, J. Castura, and F. Kschischang, "Decoder-first code design," in *Proc. 2nd International Symposium on Turbo Codes & Related Topics*, Brest, France, Sep. 2000, pp. 459–462.

[15] M. M. Mansour and N. R. Shanbhag, "VLSI architectures for SISO-APP decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 4, pp. 627–650, Aug. 2003.

[16] J. Dielissen, A. Hekstra, and V. Berg, "Low cost LDPC decoder for DVB-S2," in *Proc. 2006 Design, Automation and Test in Europe (DATE '06)*, Munich, Germany, Mar. 2006.

[17] "CoWare," http://www.coware.com.