Chapter 8
ASIP Design for Multi-Standard Channel Decoders

Purushotham Murugappa, Amer Baghdadi, and Michel Jezequel

8.1 Flexibility Requirement in Channel Decoder Design

Mobile wireless connectivity is a key feature of a growing number of devices, which will count soon in tens of billions, from laptops, tablets, cell phones, cameras, and other portable devices. The variety of applications and traffic types will be significantly larger than today and will result in more diverse requirements. These applications are driving the creation of new transmission techniques and design architectures that push the boundaries to achieve high throughput, low latency, area, and power efficient implementations.

Channel coding is one of the key techniques that enable reliable high throughput data transfer through unreliable wireless channels. However, as a large variety of channel coding options and flavors are specified in existing and emerging digital communication standards, there is an increasing need for flexible implementations. In fact, several powerful error correction techniques exist today, each suitable for specific application parameters (frame size, transmission channel, signal-to-noise ratio, bandwidth, etc.). Considering the emerging multi-mode and multi-standard applications, as well as the increasing interest for Software Defined Radio (SDR) and Cognitive Radio (CR) applications, combination of multiple error correction techniques becomes mandatory. Table 8.1 shows a representative set of mobile wireless standards to highlight their differences in data rates and channel encoding schemes. The most commonly used error correcting codes in these standards are convolutional codes (CC), turbo codes (SBTC: single-binary turbo codes and DBTC: double-binary turbo codes), and low-density parity-check (LDPC) codes.

P. Murugappa • A. Baghdadi (✉) • M. Jezequel
Institut Mines-Telecom, Telecom Bretagne, CNRS Lab-STICC,
Technopôle Brest-Iroise, 29238 Brest, France
e-mail: Purushotham.Murugappa@telecom-bretagne.eu; Amer.Baghdadi@telecom-bretagne.eu; Michel.Jezequel@telecom-bretagne.eu

C. Chavet, P. Coussy (eds.), *Advanced Hardware Design for Error Correcting Codes*, DOI 10.1007/978-3-319-10569-7_8

Table 8.1 Representative set of mobile wireless standards and related channel codes and parameters

Standard	Codes	Rates	States	Frame size (bits)	Throughput (Mbps)
UMTS	CC	1/4..1/2	256	.. 504	<1
	SBTC	1/3	8	.. 5,114	.. 2
HSDPA	SBTC	1/2–3/4	8	.. 5,114	.. 14.4
CDMA2000	CC	1/6 .. 1/2	256	.. 744	<1
	SBTC	1/5–1/2	8	.. 20,730	.. 2
IEEE-802.11n (WiFi)	CC	1/2..3/4	64	.. 4,095	.. 450
	LDPC	1/2–5/6	-	.. 1,620	.. 450
IEEE802.16e (WiMAX)	CC	1/2–5/6	64	.. 864	.. 75
	DBTC	1/2–3/4	8	.. 4,800	.. 75
	LDPC	1/2–5/6	-	.. 1,920	.. 75
DVB-S2	LDPC	1/4–9/10	-	.. 64,800	.. 90
DVB-RCS	DBTC	1/3–6/7	8	.. 1,728	.. 2
3GPP-LTE	SBTC	0.33–0.95	8	.. 6,144	.. 150

In this context, and at the receiver side, it is well known that channel decoding is one of the most computation, communication, and memory intensive, and thus, power-consuming component. Channel decoder design has been extensively investigated during the last few years and several implementations have been proposed. Some of these implementations succeeded in achieving high throughput for specific standards through the adoption of highly dedicated architectures that work as hardware accelerators. However, these implementations do not take into account flexibility and scalability issues. Particularly, this approach implies the allocation of multiple separated hardware accelerators to realize multi-standard systems, which often result in poor hardware efficiency. Furthermore, it implies long design-time which is no more compatible with the severe time-to-market constraints and the continuous development of new standards and applications.

More recently, several contributions have been proposed targeting flexible, yet high throughput, implementations of channel decoders. The flexibility varies from supporting different modes of a single communication standard to the support of multi-standards multi-modes applications. Other implementations have even proposed to increase the target flexibility to the support of different channel coding techniques. As a matter of fact, a knowledge gap is growing quickly in the last few years between the need for flexibility in the digital base-band processing segment of modern communication systems, and the actual availability of flexible while efficient hardware support to the quest for reconfigurability. The main reason that determines this growing gap is related to the poor area and energy efficiency of flexible solutions proposed till now and the huge increase of non-recurrent engineering (NRE) costs in the production of dedicated integrated circuits for specific applications (ASIC) with new semiconductor technologies.

Towards the target of filling the above mentioned gap, it becomes crucial to define and develop efficient and high performance flexible channel decoder architecture models for emerging and future digital communication systems. The need of optimal solutions in terms of performance, area, and power consumption is increasing and cannot be neglected against flexibility. In common understanding, a blind approach towards flexibility results in some loss in optimality. The objective of recent initiatives in this context is to unify flexibility-oriented and optimization-oriented approaches. The main goal is to deliver enablers and building block solutions in order to derive, for a specific application need, the best balance between a highly flexible solution and a specifically optimized one.

This chapter illustrates the use of Application-Specific Instruction-set Processor (ASIP) models and tools as one of the main recent design approaches towards this target, enabling the designer to scale and freely tune the flexibility/performance trade-off as required by the considered application requirements. Related contributions are emerging rapidly seeking to improve the resulting architecture efficiency in terms of performance/area and in addition to increase the flexibility support. The chapter starts by introducing the ASIP design approach and the recently available methodologies and tools. Then we illustrate the application of this design approach through two ASIP design examples for multi-standard turbo decoding with different architecture alternatives and design objectives. Considering the increased flexibility requirement for the support of multiple types of channel codes, the chapter presents then a short review of related state-of-the-art contributions to illustrate the trend towards the use of ASIP-based design approach in this context. The chapter ends with a summary to highlight the main conclusions and to introduce few related research perspectives.

8.2 ASIP Design Approach

In traditional design of flexible hardware architectures, the flexibility is incorporated by the expert designer through the use of initialization parameters loaded from a configuration memory or input ports of the design. The architecture description involves manual design of a finite state machine (FSM) that controls the different design units of the pipeline taking into account the various supported parameters. But when the number of flexibility parameters increases, the design and validation of such parametrized control logic become more and more complicated.

On the other hand, instruction-set based processors provide inherently high flexibility in terms of control logic design through software programmability. Their architectures have evolved dramatically in the last couple of decades [1] from microprogrammed FSMs to dynamically reordered parallel pipelines with multiple levels of parallelism and optimization techniques. More recently, the trend to design customized instruction-set processors has been made successful given the development of new design methodologies and tools. Such tools enable the designers to specify a customizable processor in weeks rather than months.

Two main approaches can be distinguished in this regard [1]. The first allows to configure and extend a predefined core or architectural skeleton with application-specific hardware resources [2–4]. Additional instructions can be defined with corresponding functional pipelines and application-specific register files or memory interfaces. The second approach allows to support architects in the effort to design from scratch a completely custom processor with its complete tool chain (compiler, simulator, etc.). Ideally, it uses an architectural description language (ADL) to describe the architecture of the design from which all tools and the synthesizable description of the core can be generated [5, 6]. All these approaches fall under the name of customizable processors and often are referred as ASIP for Application-Specific Instruction-set Processors (ASIPs).

Designing an efficient ASIP requires a deep analysis of the target flexibility parameters and algorithm variants in order to devise the adequate algorithm and architecture choices that enable efficient hardware resource allocation and sharing. The application-specific instruction-set can then be derived accordingly. The general ASIP design methodology comprises the following four steps: (1) analysis of target application and underlined algorithms with respect to flexibility requirements, (2) derivation of algorithm/architectural choices for the target flexibility and parallelism degree, (3) design of basic building blocks of the ASIP with efficient resource usage and sharing, and (4) design of the complete architecture of the ASIP including the dedicated instruction-set, datapath, pipeline stages, memory banks, and I/O interfaces.

One of the most mature ASIP development tools is Processor Designer which was first commercialized by the startup LISATek and then acquired by CoWare in 2003 and later by Synopsys in 2010. It was developed with a simulator-centric view [1] and uses a C-like language (LISA) for design description of programmable architectures and their peripherals and interfaces. The language syntax provides a high flexibility to describe the instruction set of various parallelism techniques with explicit modeling of both data-path and control. The usage of a centralized description of the processor architecture ensures the consistency of the Instruction-Set Simulator (ISS), software development tools (compiler, assembler, and linker, etc.), and RTL (Register Transfer Level) implementation, minimizing the verification and debug effort. The benefits of using this design approach are illustrated in the rest of this chapter through examples related to multi-standard channel decoders design.

8.3 ASIP-Based Decoders for Turbo Codes

This section illustrates the ASIP design flow for the implementation multi-standard turbo decoders. After a brief introduction on the hardware-efficient decoding algorithm for turbo codes, a short review of the related state-of-the-art implementations is given. Then, two ASIP design examples with different architecture alternatives and design objectives are presented together with performance analysis and comparisons.

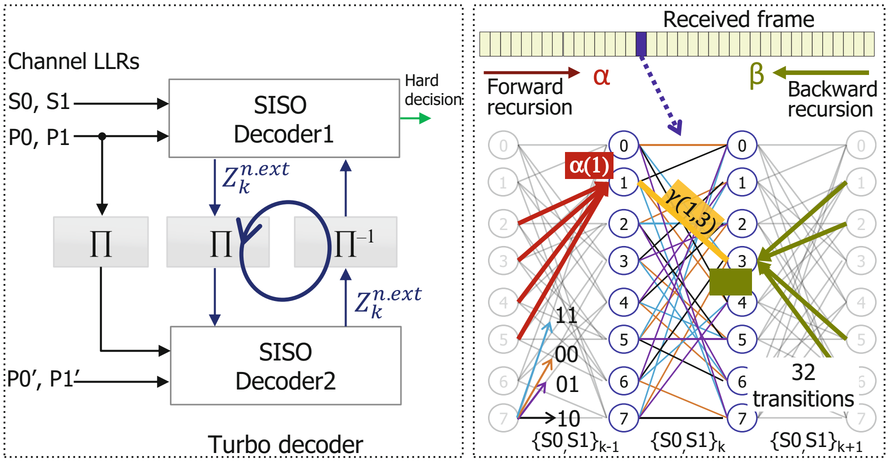

Fig. 8.1 Typical turbo codes decoder structure and an 8-states DBTC trellis

8.3.1 *Hardware-Efficient Decoding Algorithm*

Typical turbo decoder structure consists of two Soft Input Soft Output (SISO) component decoders iteratively exchanging extrinsic information via an interleaving (Π) and deinterleaving (Π^{-1}) functions as illustrated in Fig. 8.1. The SISO decoders often implement the max-log-MAP algorithm [7], which represents the hardware-efficient version of the maximum a posteriori (MAP) decoding using the BCJR algorithm known to achieve the minimal symbol error probability. In order to explain briefly the underlined max-log-MAP computations, let us consider the 8-state DBTC code of WiMAX standard, represented by its trellis in Fig. 8.1. For each received double binary symbol $\{S0,S1\}_k$, the SISO decoder computes first the branch metrics $\gamma_k(s',s)$ which represent the probability of a transition to occur between two trellis states (s': starting state, s: ending state). Then the SISO decoder runs the forward and backward recursions over the trellis (Fig. 8.1). The forward state metrics $\alpha_k(s)$ of the kth symbol are computed recursively using those of the $(k-1)$th symbol and the branch metrics of the corresponding trellis section. Similarly for the backward state metrics $\beta_k(s)$ which corresponds however to the backward recursion (traversing the trellis in the reverse direction).

$$\alpha_k(s) = \max_{s'}(\alpha_{k-1}(s) + \gamma_k(s^{'},s))$$
$$\forall(s',s=0,1,..7) \qquad (8.1)$$

$$\beta_k(s) = \max_{s'}(\beta_{k+1}(s) + \gamma_k(s^{'},s))$$
$$\forall(s',s=0,1,..7) \qquad (8.2)$$

Finally, the extrinsic information Z_k^{ext} of the kth symbol is computed for all possible decisions $(00, 01, 10, 11)$ using the forward state metrics, the backward state metrics, and the extrinsic part of the branch metrics as formulated in the following expressions:

$$Z_k^{apos}(d(s',s)=x) = \max_{(s',s)/d(s',s)=x} (\alpha_{k-1}(s) + \gamma_k(s',s) + \beta_k(s))$$
$$\forall(x = 00, 01, 10, 11) \tag{8.3}$$

$$Z_k^{ext}(d(s',s)=x) = Z_k^{apos}(d(s',s)=x) - \gamma_k^{int_x}(s',s)$$
$$\forall(x = 00, 01, 10, 11) \tag{8.4}$$

The extrinsic information can be normalized by subtracting the minimum value in order to reduce the related storage and communication requirements, thus only three extrinsic information values should be exchanged for each symbol.

$$Z_k^{n.ext}(d(s',s)=x) = Z_k^{ext}(d(s',s)=x) - min(Z_k^{ext}(d(s',s)=x))$$
$$\forall(x = 00, 01, 10, 11) \tag{8.5}$$

Executing one forward–backward recursion on all symbols of the received frame in the natural order completes one half iteration. A second half iteration should be executed in the interleaved order to complete one full turbo decoding iteration. Once all the iterations are completed (usually 6–7 iterations), the turbo decoder produces a hard decision for each symbol $Z_k^{hard\ dec.} \in (00, 01, 10, 11)$.

For SBTC, the use of the trellis compression (Radix-4) [8] represents an efficient parallelism technique and allows for efficient resource sharing with a DBTC SISO decoder, as two single binary trellis sections (two bits) can be merged into one double binary trellis section.

8.3.2 *State-of-the-Art on Turbo Codes Decoders*

Since the discovery of turbo codes in 1993 [9], considerable amount of research works has been targeting practical VLSI implementations of turbo decoders. Using mainly the low complexity max-log-MAP algorithm, many contributions have been proposed targeting diverse design objectives in terms of area efficiency, energy efficiency, flexibility, scalability, and high throughput. Among the initial efforts in this context we can cite the examples of [10–13]. Authors of [14] present an overview of the implementation aspects related to turbo decoding architectures discussing the issues of quantization and iteration stopping criteria. Several joint algorithmic/architecture optimization techniques have been investigated in [15]. One of the first ASIC implementations was presented in [16] achieving a throughput of 50 Mbps with ten decoding iterations and an operating frequency of 1 GHz.

Turbo codes have been since widely adopted in wireless communication standards like CDMA2000, 3GPP, LTE, WiMAX, DVB-RCS, etc. Many implementations have succeeded to meet the low throughput requirements of the early standards (e.g., CDMA2000 and 3GPP) using advanced DSP architectures [17–19] or customizable processors [20]. However, the scalability of such implementations is limited by the block interleavers specified in these standards which cause memory access contentions when targeting higher sub-block parallelism degree. The work presented in [21], targeting LTE, allows multiple SISO decoders (1, 2, 4, or 8) to concurrently process frame sub-blocks and integrates a three-stage network to connect the multiple memory and SISO decoder modules. Implemented in 90 nm CMOS technology, the design achieves a throughput of 129 Mbps with eight iterations and occupies an area of 2.1 mm^2 while exhibiting a power consumption of 219 mW and supporting the maximum specified frame size of 6,144 bits. Targeting Gbps throughputs, a recent work [22] has proposed an LTE compliant turbo decoder architecture with 32 parallel SISO decoders. A throughput of 2.15 Gbps is achieved with an on chip area of 7.1 mm^2 using 65 nm CMOS technology. Other works have proposed the additional support of DBTC specified in WiMAX standard. As an example, the work in [23] presents an architecture which supports all DBTC parameters specified in WiMAX and 18 frame sizes of the 188 specified in LTE. A high area efficiency is achieved by supporting only those frame sizes with interleaving properties that can be easily mapped to the extrinsic exchange paths of DBTC. Another example is the parameterized architecture of [24] which supports both turbo codes modes (DBTC and SBTC) and achieves a high throughput of 187 Mbps with eight parallel MAP decoders.

Recently, ASIP design approaches have been explored in this application context. Such an architecture model enables the designer to freely tune the flexibility/performance trade-off and thus to meet the increasing flexibility requirement efficiently. Furthermore, the well-established design methodology and the mature available tools enable short design-time. The rest of this section will illustrate this trend through the presentation of two ASIP design examples with different architecture alternatives and design objectives.

8.3.3 TurbASIP: Highly Flexible ASIP

One of the first proposed ASIPs for flexible turbo decoding has been presented in [25]. The main design objective for this ASIP, namely TurbASIP, was to explore the effectiveness of the newly proposed ASIP-design tools in terms of quality of the generated HDL code and flexibility limitations when targeting this class of applications. To that end, the target flexibility was set very high to investigate the support of any convolutional code trellis of DBTC and SBTC. The number of trellis states, however, is limited to 8 states in DBTC mode and 16 states in SBTC mode, as typically adopted in existing wireless communication standards. Another design objective for this ASIP was the investigation and the exploitation of the various parallelism techniques available for turbo decoding. Most of the available parallelism techniques [26] have been exploited. This includes:

1. Metric level parallelism, which concerns the processing of all metrics involved in the decoding of each received symbol inside a SISO decoder. It exploits the inherent parallelism of the trellis structure (as the same operations related to the computation of γ, α, β and the extrinsic information (γ^{ext}) should be repeated for all the trellis transitions) and the parallelism of the MAP computations (parallel computation of the metrics α, β, and extrinsic information γ^{ext}).
2. SISO decoder level parallelism, which exploits the use of multiple SISO decoders, each processing a sub-block of the same frame in natural or interleaved orders. At this level, parallelism can be applied either on sub-blocks and/or on component decoders (shuffled decoding).
3. Turbo decoder level parallelism, which simply duplicates whole turbo decoder to process iterations and/or frames in parallel. Nevertheless, this parallelism level is too area-expensive (all memories and computation resources are duplicated) and presents no gain in frame decoding latency, and thus it was not considered.

8.3.3.1 Overview of TurbASIP Architecture

Considering these design objectives, adequate algorithm/architectural choices have been derived to meet the target flexibility and parallelism degree. This step is followed by the design of basic building blocks of the ASIP with efficient resource usage and sharing. Figure 8.2 presents the overall architecture of TurbASIP.

The architecture exploits the metric level parallelism through the use of two hardware recursion units to process concurrently the forward and backward computations following a butterfly scheduling scheme. Each recursion unit includes a state metric calculation matrix with 32 *adder* nodes and 8 max operator nodes to compute all trellis transitions related metrics in parallel. It includes in addition the required hardware resources to compute branch metrics and the hard decision with multiple registers to store intermediate results. The target high flexibility degree is supported through the use of high number of multiplexers enabling to handle any convolutional code trellis structure. A main design choice in the ASIP architecture for turbo decoding concerns the memory organization. Besides the quantization aspects which impact the width of the memory words, the parallelism degree imposes the use of multiple memory banks and the flexibility requirement (supported frame lengths) impacts directly the size of these banks. Figure 8.2 illustrates how particular is the memory organization for such application and consequently how adequate is an ASIP design approach compared to general-purpose instruction-set processor

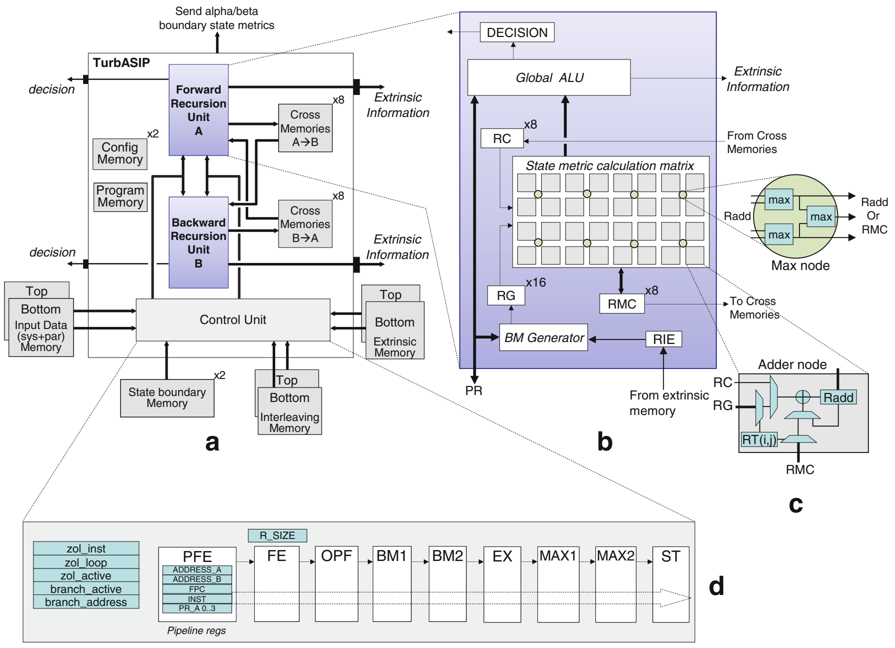

Fig. 8.2 TurbASIP architecture: (**a**) overview, (**b**) recursion unit, (**c**) adder node, (**d**) pipeline

architectures. Besides the *Input memories* (duplicated due to the parallelism implied by the choice of butterfly scheduling scheme) several memories are required: *Extrinsic memories* to store extrinsic messages exchanged between SISO decoders, *Interleaving memories* to store interleaving addresses, *Cross memories* to store intermediate metrics between the two executed forward/backward recursions, *State boundary memories* to store boundary state metrics for metric initialization between neighboring sub-blocks, *Program memory* to store program instructions, and *Config memory* to store parameters related to the supported codes (trellis definition, block size, number of iterations, etc.).

Exploiting the second level of parallelism, sub-block parallelism and shuffled decoding, is done through the instantiation of multiple ASIPs in each component decoder to process concurrently the received frame, partitioned in sub-blocks, in natural and interleaved order. The ASIP should feature the required input/output interfaces and the flexibility to support any interleaving rules and parallelism degree is met through the use of application-specific network-on-chip (NoC) architectures [27]. The efficiency of this parallelism level and related analysis of the architecture scalability are addressed in [26].

8.3.3.2 Pipeline Architecture and Instruction-Set

The devised pipeline structure for TurbASIP is illustrated in Fig. 8.2. The first three stages correspond to instruction address generation, instruction fetch, and instruction decode. The branch metrics are calculated in the two pipeline stages *BM1* and *BM2*. The *EX* stage executes the adder nodes to compute the 32 state metric LLRs as defined in (8.1) and (8.2). The *MAX1* stage executes the reconfigurable max operators and computes the α and β state metric LLRs by taking the maximum of the *RADD* registers column wise. During extrinsic generation phase, the max operators are reconfigured to calculate the maximum of the four *RADD* registers along the horizontal direction of the state metric calculation matrix. Thus, two partial a posteriori LLR values are generated per row. The final a posteriori LLRs are generated by the max operators in the *MAX2* pipeline stage that calculates the maximum of the eight partial a posteriori LLRs to produce the four a posteriori symbol LLRs γ^{apos}. The last stage of the pipeline (*ST*) generates the extrinsic information from a posteriori LLRs and the intrinsic LLRs γ^{int} as given by (8.4).

Appropriate instruction-set is designed according to the devised algorithm/architecture choices and target flexibility described above. An excerpt of assembly code for the first iteration of the turbo decoding execution is presented in Listing 8.1. It starts by initializing the ASIP configuration registers setting the mode by reading the trellis configuration registers (instruction *SET_CONF*). Next, the size in symbols of half of the window (SET_SIZE), the scaling factor (SET_SF), number of windows in the sub-block (SET_WINDOW_N), and the initial window counter (SET_WINDOW_ID) are set. The value 6 used by the *SET_SF* command allows to scale the input extrinsic LLRs by 0.875 before computing the branch metrics. The initial window boundary state registers of the first window (RMC) are initialized to be uniform, i.e., all states are initialized to same equiprobable value. Zero overhead loop (*ZOLB*) instruction is devised to execute lines (@26) and (@27) 32 times, i.e., half of window size (as set by the SET_SIZE instruction). The instruction at line (@26) implements the left side of the butterfly decoding scheme. During the first iteration of the shuffled decoding the extrinsic memories are uninitialized, hence the *WITHOUT_EXT* field of the instruction specifies not to use extrinsic information in the branch metric calculations. The *ADD M* field of the instruction forces the adder nodes to do 32 state metric calculations ($\alpha + \gamma$ or $\beta + \gamma$) in the *EX* stage of the pipeline. The *COLUMN* field configures the max nodes to calculate the maximum column-wise. An idle cycle is introduced via *NOP* instruction at line (@27) due to data dependency. Once the left side of butterfly decoding schedule is completed for a window, the right side of the butterfly schedule is processed by executing the instructions at lines (@30) and (@33) 32 times. The instruction *EXT ADD i LINE* computes the extrinsic information. The field *LINE* configures the max nodes in the *MAX1* pipeline stage to calculate the maximum row-wise. Additionally, this instruction activates *MAX2* stage to finalize the computation of the extrinsic information and *ST* stage to fetch the corresponding interleaved/de-interleaved addresses for NoC packetization.Two other short portions of assembly

Listing 8.1 Example of TurbASIP assembly code for the first turbo decoding iteration

```
.text
;set configuration wimax
        SET_CONF 0
        SET_CONF 1
        SET_CONF 2
        SET_CONF 3
;set half window size
        SET_SIZE 32
; set the scale factor 6=0.875
        SET_SF 6
;set number of windows and initial
;window id counter to zero
        SET_WINDOW_N 2
        SET_WINDOW_ID 0
;set boundary initialization of
;RMC registers as uniform
        SET_RMC UNIFORM, UNIFORM
        REPEAT UNTIL _loop0 1 times
        NOP
;repeat instructions between _RW1 to _CW1
;and between _CW1 to_LW1 SET_SIZE times
        ZOLB _RW1,_CW1,_LW1
        W_LD_BETA 0
;configure max units to take max column wise.
;store results in RMC
_RW1: DATA LEFT WITHOUT_EXT ADD M COLUMN
_CW1: NOP
;configure max units to take max column wise.
;store results in RMC
        DATA RIGHT WITHOUT_EXT ADD M COLUMN
;configure max units to take max rowwise.
;to calculate extrinsic
_LW1: EXT WITHOUT_EXT ADD i LINE
;increment the window number
        EXC_WINDOW
        NOP
        NOP
_loop0: NOP
```

code, presented in [28], are required to process the subsequent iterations of the turbo decoding which differ by using extrinsic LLRs during branch metric calculation and computing the hard decision in the last iteration.

8.3.3.3 Results and Architecture Efficiency Definition

The devised architecture for TurbASIP was described in LISA and validated using Synopsys Processor Designer tool and a bit-true software reference model of the algorithm. The generated VHDL code has been validated and synthesized targeting 65 nm general purpose CMOS technology. The high flexibility of TurbASIP allows to support any trellis description at run-time and quantization-related configurations at design-time. Considering a quantization of 4 bits for input LLRs and 8 bits for extrinsic information, one TurbASIP occupies a total area of ~0.19 mm^2 (logic and memories) for a maximum clock frequency of 500 MHz. A multi-ASIP configuration with 4 ASIPs for each component decoder (8 in total) occupies a total area of ~1.53 mm^2 (including a NoC based on Butterfly topology).

The following expression allows to compute the achieved turbo decoding throughput:

$$Throughput_{TurbASIP} = \frac{2 \times Bits_{sym} \times f_{clk} \times N_A/2}{N_{instr} \times N_{iter}} \tag{8.6}$$

where N_A = number of ASIPs, N_{instr} = number of instructions to process two symbols, $Bits_{sym}$ = number of bits per symbol, f_{clk} = clock frequency, N_{iter} = number of iterations.

As TurbASIP needs on average 4 instructions per iteration (2 for left butterfly and 2 for right butterfly) in order to process two double binary symbols, thus we have $N_{instr} = 4$ and $Bits_{sym} = 2$ in DBTC mode. Considering a 4x4 TurbASIP turbo decoder, and using the above expression where $N_A = 8$, f_{clk} =500 MHz, and N_{iter} =6, the achieved throughput is around 333 Mbps.

In order to evaluate the effectiveness of the obtained results and to be able to compare with state-of-the-art implementations, we define the *Architecture efficiency* (*AE*) metric as follows:

$$AE = \frac{Throughput \times N_{iter}}{Area_{Norm} \times f_{clk}} \tag{8.7}$$

Its unit of measure is bits/cycle/iteration/mm^2 and it represents the number of decoded bits per clock cycle per iteration per mm^2 that the proposed iterative channel decoder implementation is able to deliver. A high architecture efficiency indicates an optimized design which exploits efficiently its hardware resources during its execution time. An interesting point in the above expression of the *AE* concerns the normalization of the throughput achieved with respect to the considered clock frequency (f_{clk}) which increases the fairness when comparisons are done between different decoding architectures running at different clock frequencies. Published results in this context consider either the maximum achievable clock frequency by the proposed architecture or a lower operational clock frequency which is sufficient to achieve the target throughput. Thus, normalizing the presented throughput by the considered clock frequency enables to better exhibit the efficiency of the proposed architectural choices. Towards the same objective, the above expression of the *AE* normalizes the throughput by the considered number of decoding iterations (N_{iter}) as the published results can use slightly different values which impact the overall throughput. In most of these works, the same low complexity decoding algorithms, with identical convergence speed, are used. Similarly, the *AE* expression uses a normalized area measure ($Area_{Norm}$) as the published decoders are often based on different technology nodes (e.g., 180 nm, 130 nm, 65 nm, etc.). In addition, when the published design area is given post-place and route a downscaling factor of 2 is applied to obtain a reasonable estimate of the post-synthesis area. This factor is not very accurate as it depends to many parameters (technology node, CAD tools, operating conditions, etc.), but it gives a reasonable idea as it corresponds to the usually (or even worst case) observed ratio.

Considering the synthesis results presented above, a 4x4 TurbASIP turbo decoder presents an architecture efficiency of 2.6 bits/cycle/iteration/mm^2. This value is somehow low compared to recent related state-of-the-art implementations as it will be illustrated in the next sub-section and this is due to the target design objectives of high degrees of flexibility and scalability. In fact, the example of this highly flexible ASIP allows to illustrate the effectiveness of the ASIP design approach allowing to explore and implement rapidly any desired algorithm/architecture choices and to generate high quality synthesizable HDL code. It allows also the design of scalable multi-ASIP turbo decoders meeting the high-throughput requirement. The next sub-section further illustrates how it is possible to increase significantly the architecture efficiency of ASIP-based turbo decoders.

8.3.4 TDecASIP: Parameterized Area-Efficient ASIP

In this sub-section we present another example of ASIP for multi-standard turbo decoding and we analyze and discuss the architecture efficiency with respect to related state-of-the-art implementations. The design objective behind this ASIP design example, namely TDecASIP [29], is twofold: (1) investigate the maximum attainable architecture efficiency for ASIP-based turbo decoding, and related to this first objective (2) investigate the possibility to design application-specific parametrized cores using the available ASIP design flow. Such possibility can potentially lead to a higher architecture efficiency by simplifying the instruction decoding logic and removing the program memory. Furthermore, it is still keeping the benefit of the short design cycle enabled by the well-established ASIP design tools.

8.3.4.1 Overview of TDecASIP Architecture

A first key element to increase the architecture efficiency is to limit the flexibility degree to the exact target application at design-time. Rather than supporting any trellis code as in TurbASIP, in this design example of TDecASIP the target flexibility is set to cover only the turbo codes and related parameters specified in 3GPP-LTE, WiMAX, and DVB-RCS standards. In addition, this choice enables to compare the results with existing state-of-the-art implementations.

The second key element concerns the algorithm/architectures choices in terms of parallelism techniques and degrees. Most appropriate choices have been made targeting a throughput in the range of hundreds Mbps, as specified in the considered standards. In order to fully exploit the metric level parallelism, two recursion units are devised using backward–forward schedule for window processing. The first recursion unit (processing in the backward direction of the trellis) processes a window j while the second recursion unit (processing in the forward direction of the trellis) executes on the window $j-1$ in parallel as illustrated in Fig. 8.3. This enables

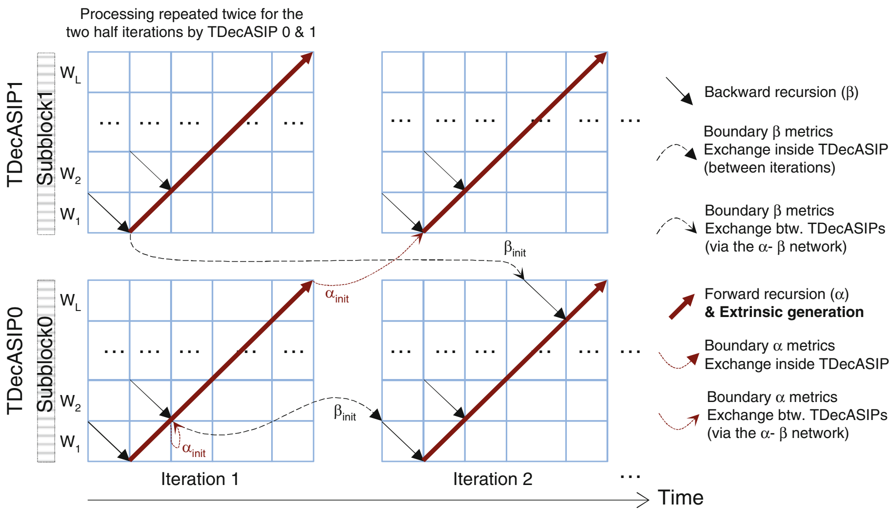
Processing repeated twice for the two half iterations by TDecASIP 0 & 1
TDecASIP1
Subblock1
TDecASIP0
Subblock0
W_L
W_2
W_1
β_{init}
α_{init}
Iteration 1
Iteration 2
Time
Backward recursion (β)
Boundary β metrics Exchange inside TDecASIP (between iterations)
Boundary β metrics Exchange btw. TDecASIPs (via the α- β network)
Forward recursion (α) & Extrinsic generation
Boundary α metrics Exchange inside TDecASIP
Boundary α metrics Exchange btw. TDecASIPs (via the α- β network)

Fig. 8.3 Windowing and backward–forward schedule in TDecASIP

to achieve the throughput equivalent to butterfly schedule (as in TurbASIP design), yet the using of the backward–forward schedule further enables efficient use of hardware interleave address generators for extrinsic memory addressing.

Regarding the exploitation of the second level of parallelism (SISO decoder level), only sub-blocking parallelism with a degree of two is devised as it allows to meet the target throughputs. Shuffled decoding efficiency is demonstrated only for very high parallelism degrees [26]. Thus, in TDecASIP, half iterations are performed in serial order, i.e., all processing cores perform first half iteration by reading the systematic and extrinsic information sequentially from memories, followed by the second half iteration where the systematic and extrinsic memories are read in interleaved order. The generated extrinsic data are written at the same location as it was read from. In both of these half iteration cycles the parity memory is always read sequentially. This type of scheduling presents the following advantages:

- Only one copy of systematic information bits is needed to be stored. This reduces the number of memory banks required and the configuration network complexity.
- Only sequential counter and interleaved address generator are needed for addressing the memories while the shuffled decoding needs in addition a deinterleaved address sequence. Given the adopted low sub-block parallelism degree, this serial decoding reduces the memory access complexity as only low number of multiplexers would be sufficient (read/write exchange network). WiMAX interleavers support sub-blocking of 2 and 4 while LTE interleavers support sub-blocking of at least 2 and 4 [30] (with a maximum of 64).
- Small number of memory banks also results in less address decoding logic and hence reduced total memory area, resulting in area-efficient decoding core.

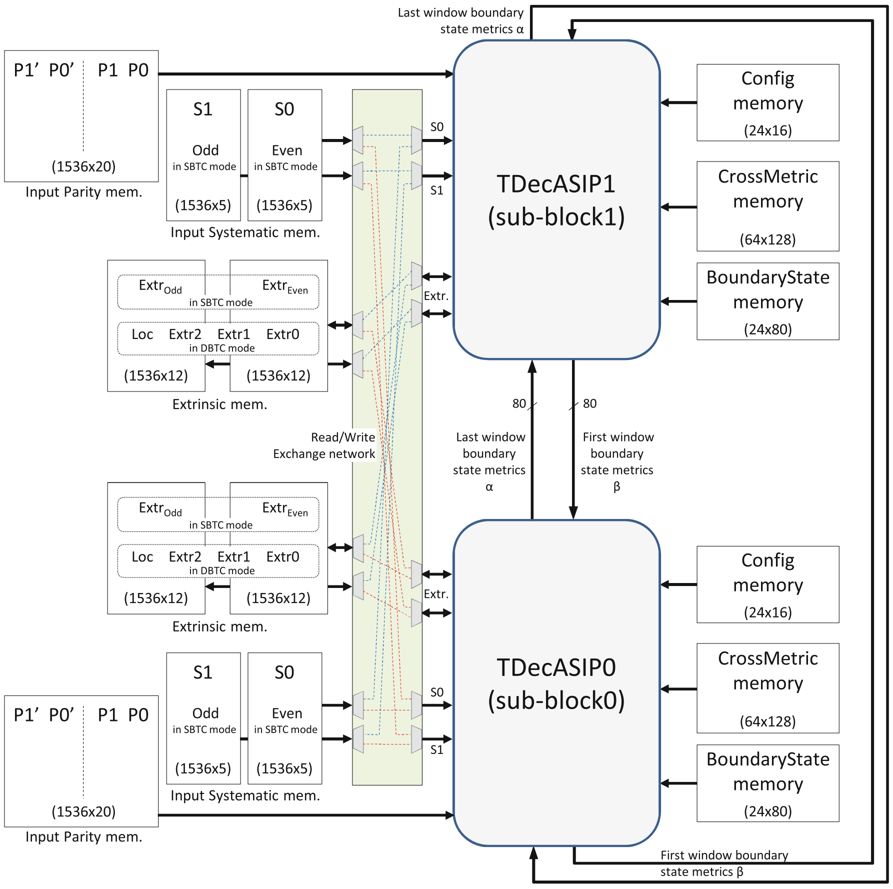

Fig. 8.4 Overview and memory organization of the 2-TDecASIP turbo decoder architecture [29]

Based on the above design motivations and choices, Fig. 8.4 illustrates the overall architecture of the two core turbo decoder. Each core (TDecASIP) processes a sub-block of the input frame and has direct access to configuration, CrossMetric, BoundaryState, and input Parity memories. The input Systematic and Extrinsic memory banks are connected to the cores through a simple read/write exchange network.

Figure 8.5 presents a detailed architecture of TDecASIP core. As for TurbASIP, the pipeline is structured in eight stages, of which the first three stages are dedicated for the data fetch from the memories and for the control of the pipeline. A modified ASIP design flow is proposed to enable parameterized core design. Rather than defining specialized instructions, the corresponding FSM is directly described in LISA. The current state of the FSM is treated as an instruction. This approach can be effective when the application exhibits a reduced number of flexible parameters

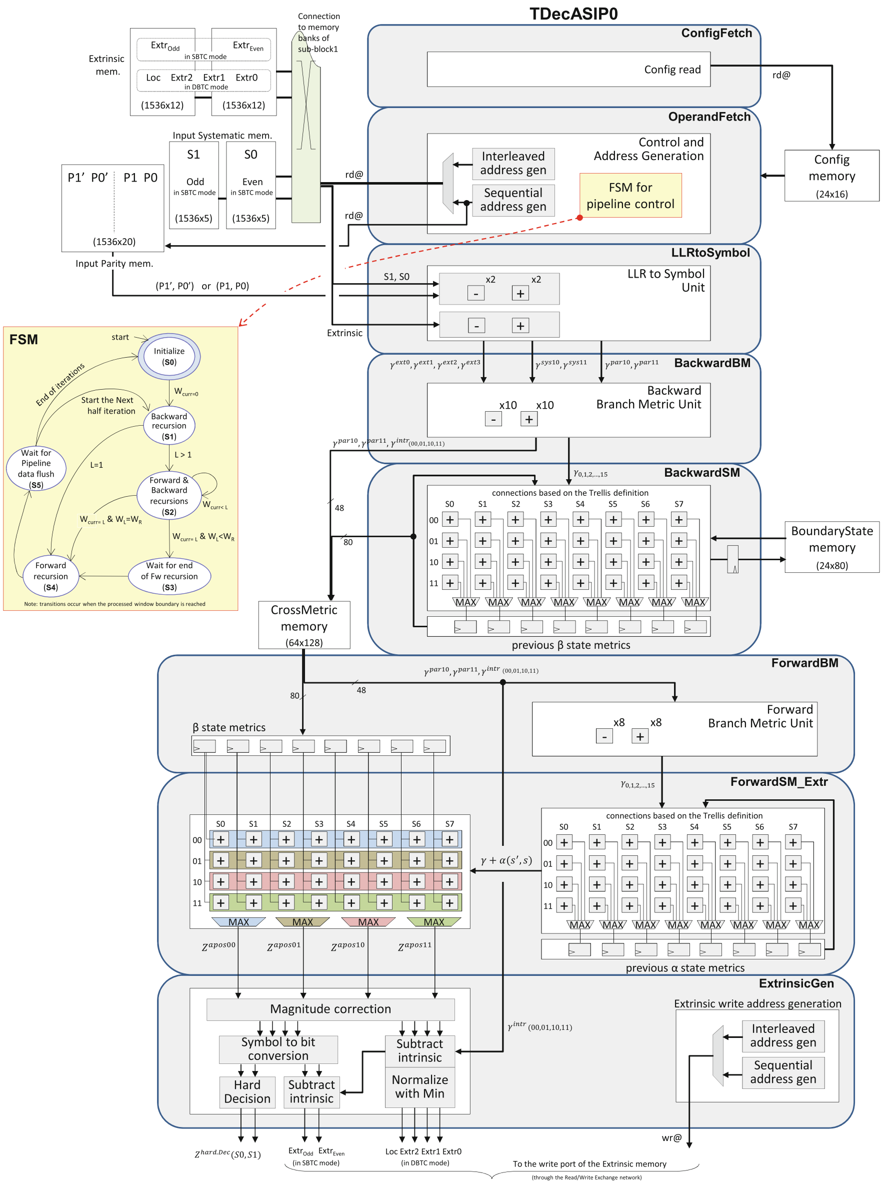

Fig. 8.5 Detailed pipeline architecture and FSM of the TDecASIP parametrized core [29]

and the corresponding processing presents a reduced number of control states. The target application in this study (flexible turbo decoding) is a good example with six states (as shown in Fig. 8.5) and few flexible parameters that do not change during the decoding process (code type, number of iterations, window size, number of windows, and extrinsic address generation initialization values). This FSM is implemented in the *OperandFetch* pipeline stage to generate appropriate control signals to activate or deactivate the appropriate stages of the pipeline (Fig. 8.5). As soon as the start signal is asserted, the processor starts with the *Initialize* state, initializing the registers to the default values and reading the configuration parameters mentioned above. At the end of the initialization, the FSM reaches *S1* state generating appropriate signals for the backward recursion execution. If the processor is executing the first half iteration, the generated addresses for systematic and extrinsic memories are sequential otherwise interleaved addresses are generated. The addresses for parity memories are always sequential. All FSM transitions in Fig. 8.5 occur when the window boundary is reached. In $S2$ state both forward and backward recursions are executed in parallel (on two different windows). The complete presentation of the FSM control states and the pipeline execution can be found in [29].

The memory organization of the proposed architecture is illustrated in Fig. 8.4. With negligible performance loss, the channel LLRs can be quantized to 5 bits and the normalized extrinsic information to 7 bits. As radix-4 is adopted in SBTC, systematic LLRs are stored in two memory banks, and similarly for extrinsic LLRs. This memory organization and the corresponding efficient address generation are allowed by the QPP (quadratic permutation polynomial) interleaver adopted in LTE standard which maps even addresses to even addresses and odd to odd. The total depth of these memories allow to store up to 6,144 LLRs, which corresponds to the maximum specified LTE frame length. As the parity LLRs are always read in sequence, the consecutive parity LLRs information bits are combined and stored in one memory bank as shown in Fig. 8.4.

8.3.4.2 Results and Discussions

TDecASIP was modeled using Synopsys Processor Designer tool and the corresponding VHDL description was generated and synthesized targeting 65 nm general purpose CMOS technology (worst case 0.9v and 125C). The total logic area, including the interleaver, is 0.065 mm^2 while the memory area for one processor is 0.15 mm^2. The total area (post-synthesis) for the two core turbo decoder is 0.438 mm^2 with a clock frequency of 510 MHz. The error rate performance of the hardware implementation has negligible degradation (less than 0.1 dB) when compared to the floating point C-simulations using BPSK modulation over an additive white gaussian noise (AWGN) channel (Fig. 8.6). The throughput can be expressed as follows:

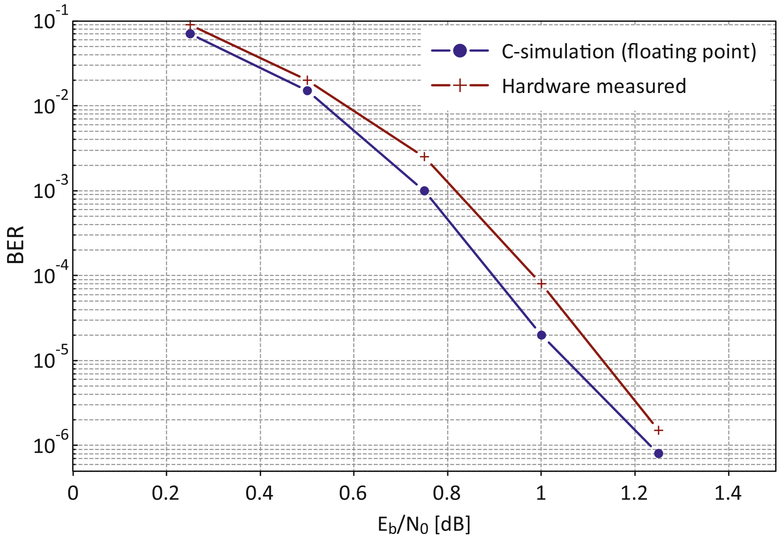

Fig. 8.6 Error rate performance comparison between the hardware implementation and the floating point simulation for WiMAX frame size 1,920 bits

$$Throughput_{TDecASIP} = \frac{N \times f_{clk}}{((\frac{\lceil N_{sym}/W \rceil}{Num_{procs}} + 1) \times W + N_{pip}) \times (2 \times N_{iter})} \tag{8.8}$$

For the presented architecture: $Num_{procs} = 2$ processors, number of pipeline stages $N_{pip} = 8$, window size $W = 64$ symbols, maximum clock frequency is $f_{clk} = 510\,\text{MHz}$, considering the largest LTE frame size $N_{sym} = 3{,}072$ symbols or $N = 6{,}144$ bits and $N_{iter} = 6.5$ iterations, the throughput obtained is 150 Mbps.

Considering the architecture efficiency definition given in the previous subsection, the proposed 2 processor turbo decoder achieves an architecture efficiency of 4.37 bits/cycle/iteration/mm^2. Furthermore, the proposed architecture is scalable and can be extended to 4 processing cores, since both LTE and WiMAX interleavers support sub-blocking level of 4 with conflict-free memory accesses. In this case, the memory area of one processing core decoder becomes 0.097 mm^2 which results in a total area occupancy of 0.65 mm^2. The architecture efficiency in this case is 5.88 bits/cycle/iteration/mm^2. This further illustrates the area efficiency of the sub-block parallelism, where the throughput is doubled while the occupied area is increased only by 1.47 times (rather than doubled). This is due to the fact that Systematic, Parity, Extrinsic, and BoundaryState memory requirements remain unchanged. The achieved results of this design are summarized and compared along with few recent related works in Table 8.2. The cited three implementations [24, 31, 32] use a conventional parametrized design approach with almost similar internal computation, interleaving, and storage optimization techniques. However, each of them has selected a different sub-blocking parallelism level (8, 16, and 32). The increased

Table 8.2 Results and comparison with few recent related works

	TDecASIP		[24]	[31]	[32]
Standard supported	LTE, WiMAX		LTE, WiMAX	LTE	LTE
LTE modes supported #	188		188	188	188
WiMAX modes supported #	17		17	–	–
Technology (nm)	65		130	90	65
Core area (mm^2)	0.438	0.65	10.7[a]	2.1	7.7[a]
$Area_{Norm}$ @65 nm (mm^2)	0.438	0.65	1.335	1.1	3.85
Throughput (Mbps)	150 @6.5iter	300 @6.5iter	187 @8iter	284 @5iter	2150 @6iter
Parallel MAPs #	**2**	**4**	**8**	**16**	**32**
f_{clk} (MHz)	510		250	200	450
AE (bits/cycle/iter/mm^2)	4.37	5.88	4.48	6.49	7.45

[a] Post place and route

architecture efficiency with the sub-blocking parallelism degree is coherent with the above discussed results of the proposed 2- and 4-TDecASIP architectures. The 4-TDecASIP architecture achieves even a slightly better architecture efficiency than the one presented in [24] which supports both turbo modes (DBTC and SBTC) and uses 8 parallel MAP decoders. The LTE-dedicated implementations presented in [31] and [32] exploit the available higher sub-blocking parallelism degrees in this standard (parallel interleaving with conflict-free memory accesses). Results comparison illustrates how the TDecASIP architecture achieves a high architecture efficiency while using such an ASIP-based parameterized core approach by selecting the appropriate parallelism and optimization techniques.

8.4 Flexibility Increase to Support Multiple Channel Code Classes

Flexibility requirement of channel decoding architectures becomes more and more crucial when considering the emerging multi-mode and multi-standard applications, as well as the increasing interest for SDR and Cognitive Radio concepts. Even for a single standard like WiMAX, several error correction codes (convolutional, turbo, LDPC, and block turbo) are specified as mandatory or optional. Hence, in the last few years, several multi-code architectures have been explored and proposed to support the decoding of two or more different classes of error correction codes. The aim is to propose novel optimization and resource sharing techniques of the memory, logic, and/or communication interconnects in order to achieve better efficiency in terms of area when compared to the direct assembly of dedicated individual decoders. This section presents a short review of the related state-of-the-art contributions to introduce the trend towards the use of ASIP-based design approach in this context. Due to the limited space, presenting few detailed examples and an exhaustive analysis is not affordable, yet a set of recent references is provided.

In this context, few initial initiatives have investigated the support of both convolutional and turbo codes. Authors of [33] and [34] have proposed a unified architecture designed for UMTS base-stations. A dual mode Viterbi/turbo decoder, sharing path metric calculation and extrinsic information memories, is proposed. A trellis processor used to update path metrics in both supported decoding algorithms. A 2 Mbps throughput at 88 MHz clock frequency is demonstrated when performing 10 turbo decoding iterations. In [35], another combined architecture is proposed. In this architecture the datapath and the memories are shared. A max-log-MAP algorithm is used for decoding both convolutional and turbo codes. However, this is only possible when the throughput requirement for convolutional codes (e.g., 12.2 kbps) is much lower than that of turbo codes (e.g., 384 kbps). In another effort to combine the two types of decoders, soft Viterbi decoding is used for turbo decoding and hard output Viterbi decoding is used for convolutional codes [36].

Similarly, unified decoder architectures for LDPC and turbo codes have been presented in [37–40]. Multi-code decoding is achieved in [37] by employing flexible add-compare-select (FACS) units. By representing LDPC codes as parallel concatenated Single Parity Check (SPC) codes, the authors have efficiently reused the turbo decoding hardware resources for LDPC decoding functions. The architecture supports decoding of SBTC codes of LTE and LDPC codes of WiFi and WiMAX. When implemented in 90 nm CMOS technology, the work reports a maximum throughput of 450 Mbps for SBTC decoding and 600 Mbps for LDPC decoding while occupying a total area of 3.2 mm^2. Similar architecture is presented in [40] to share logic and memory resources with additional decoding support of turbo codes specified in 3GPP, DVB-SH, and WiMAX standards. The entire design is implemented in 45 nm CMOS technology occupying an area of 0.9 mm^2 and clocked at 150 MHz to achieve low power and yet meeting the target throughput. However, studies presented in [41] conclude that such datapath sharing for LDPC and turbo decoding has little benefits and only for special configurations which have similar memory requirements between the decoding modes (LDPC/turbo). It further mentions that even in such cases, sharing memory is much more attractive than sharing computational hardware. In fact, the best match for a combined LDPC/turbo datapath can be achieved when both have the same granularity, e.g., at the check-node and log-butterfly operator level [41].

Besides the above-mentioned multi-code decoder architectures which can be considered as parameterized cores, several recent initiatives have explored ASIP-based design in this application context. As an example, the FlexiTreP ASIP presented in [42] supports trellis-based channel codes (i.e., convolutional, SBTC, and DBTC) for few target standards. Decoding of LDPC codes was later added to this architecture and presented in [43] as FlexiChap where memory sharing across turbo and LDPC modes was explored. Targeting the support of WiFi and WiMAX, the total area of the channel decoder has increased from 0.42 mm^2 for FlexiTreP to 0.62 mm^2 for FlexiChap in 65 nm CMOS technology. Moreover, in [39], the authors propose an ASIP architecture addressing in a unified way the turbo and LDPC coding requirements of LTE, WiFi, WiMAX, and DVB-S2/T2 with datapath and memory reuse across the different FEC families. Results illustrate how the obtained area was lower than the cumulated area of dedicated individual turbo and LDPC decoder solutions. Furthermore, the authors of this chapter have presented in [28] several contributions belonging to these last efforts targeting ASIP-based multi-code channel decoding architectures. Promising results are presented while investigating the maximum achievable architecture efficiency when adopting the rapid design methodology and well-established tools related to ASIP design concept.

Finally, several scalable multiprocessor architectures based on the use of dedicated NoC for combined LDPC/turbo decoding were investigated. High decoder parallelism degrees are necessary to achieve the increasing throughput requirement imposed by the emerging applications. However, this incurs memory access conflicts due to the interleaving rules specified in the turbo and LDPC codes. Efficient algorithms which can compute collision-free memory mapping of interleaving laws with no constraint imposed on the code itself and the target parallelism degree are proposed in [44] and [45]. The latter further proposes novel technique that allows for low-latency dynamic reconfiguration. Another, or complementary, alternative proposes the use of adequate NoC topologies with optimized message transfer techniques. Several flexible on-chip interconnection networks have been proposed with the aim of fully exploiting the parallelism of the LDPC/turbo decoder architecture by reducing the message latency, alleviating the memory conflicts and efficiently routing any permutation law. Among the recent related contributions we can cite the work presented in [46] which proposes a NoC architecture based on binary de Bruijn topology and the work presented in [47] which proposes a NoC-based multiprocessor architecture, based on generalized Kautz topology.

8.5 Summary

This chapter has illustrated the use of ASIP models and tools as one of the main recent design approaches towards the target of unify flexibility and efficiency in the design of multi-standard channel decoders. Many contributions are emerging rapidly in this domain, seeking to increase the flexibility support and to improve the resulting architecture efficiency. The presented design examples of ASIP for turbo decoding illustrate how this approach enables to implement any set of architecture choices targeting different design objectives. The flexibility degree can be tuned to be very high or very limited; even parameterized architecture models can be developed using ASIP design tools. The main benefits correspond to the structured and rapid design approach which accelerates the design and validation flow and enables high design-time flexibility with respect to traditional digital design practices.

The chapter illustrated how flexibility, architecture efficiency, and rapid design-time can be combined when using an ASIP design methodology and tools to implement novel cores for multi-standard turbo decoding. The highly optimized parameterized core presented supports both SBTC of 3GPP-LTE and DBTC of WiMAX and DVB-RCS standards. It achieves, in both modes, a high architecture efficiency of 4.37 bits/cycle/iteration/mm^2 and meets the 150 Mbps maximum targeted throughput of the LTE standard. The proposed architecture is scalable and the architecture efficiency increases with the sub-block parallelism degree. Detailed analysis and comparisons with relevant state-of-the-art solutions have been discussed.

Same approach of exploring the maximum achievable architecture efficiency using ASIP design concept in LDPC decoding can be found in [48]. This design trend has been further shown in the proposal of flexible channel decoder supporting multiple code types; in particular the support of LDPC and turbo codes as specified in recent wireless communication standards. The chapter has presented in this context a short review of the related state-of-the-art contributions. Finally, although the chapter has mainly considered the evaluation of the architecture efficiency, similar conclusions should be driven evaluating the energy consumption and efficiency. Furthermore, several recent initiatives have proposed to tackle the aspect of reconfiguration optimization and efficient management for multi-standard ASIP-based channel decoders [49]. Finally, the promising results demonstrated in recent state-of-the-art in channel decoding have paved the way to extend this design approach to other key components of advanced communication systems. Several recent contributions start appearing in this context, e.g., for MIMO detection, and even beyond digital communication applications domain.

References

1. Ienne P, Leupers R (2006) Customizable embedded processors–design technologies and applications. Morgan Kaufmann, San Mateo
2. Cadence Xtensa Customizable Processors (formerly product of Tensilica) (2014). http://ip.cadence.com/ipportfolio/tensilica-ip/xtensa-customizable
3. Stretch Software-Configurable Processors (2014). http://www.stretchinc.com/technology/
4. Mei B, Lambrechts A, Mignolet J-Y, Verkest D, Lauwereins R (2005) Architecture exploration for a reconfigurable architecture template. IEEE Trans Des Test Comput 22(2):90–101
5. Synopsys Processor Designer (formerly product of CoWare) (2014). http://www.synopsys.com/Systems/BlockDesign/ProcessorDev
6. Synopsys IP Designer (formerly product of Target Compiler Technologies) (2014). http://www.synopsys.com/IP/ProcessorIP/asip/ip-mp-designer
7. Robertson P, Hoeher P, Villebrun E (1997) Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding. Eur Trans Telecommun 8(2):119–125

8. Bickerstaff M, Davis L, Thomas C, Garrett D, Nicol C (2003) A 24Mb/s radix-4 logMAP turbo decoder for 3GPP-HSDPA mobile wireless. In: Proceedings of the IEEE international solid-state circuits conference (ISSCC), vol 1, pp 150–484
9. Berrou C (1991) Procédé de décodage itératif, module de décodage et décodeur correspondants/Iterative decoding method, corresponding decoding module and decoder, Patent EP0735696 B1, France Telecom & TDF.
10. Hsu J-M, Wang C-L (1998) A parallel decoding scheme for turbo codes. In: Proceedings of the IEEE international symposium on circuits and systems (ISCAS)
11. Wang Z, Suzuki H, Parhi K (1999) VLSI implementation issues of TURBO decoder design for wireless applications. In: Proceedings of the IEEE workshop on signal processing systems (SiPS'99), pp 503–512
12. Kwon TW, Kim DW, Kim WT, Joo EK, Choi JR, Choi P, Kong JJ, Choi SH, Chung WH, Lee KW (1999) A modified two-step sova-based turbo decoder for low power and high performance. In: Proceedings of the IEEE region 10 conference (TENCON'99), vol 1, pp 297–300
13. Chaikalis C, Noras J (2002) Implementation of an improved reconfigurable sova/log-map turbo decoder in 3gpp. In: Proceedings of the third international conference on 3G mobile communication technologies, May, pp 146–150
14. Boutillon E, Douillard C, Montorsi G (2007) Iterative decoding of concatenated convolutional codes: implementation issues. Proc IEEE 95(6):1201–1227
15. Gnaedig D (2005) Optimisation des architectures de décodage des turbo-codes. Ph.D. dissertation, Telecom Bretagne – UBS
16. Viglione F, Masera G, Piccinini G, Ruo Roch R, Zamboni M (2000) A 50 mbit/s iterative turbo-decoder. In: Proceedings of the ACM/IEEE design, automation and test in Europe conference and exhibition (DATE'00), pp 176–180
17. TMS320C64x DSP turbo-decoder coprocessor (2004). http://www.ti.com/lit/ug/spru534b/spru534b.pdf
18. Loo K, Alukaidey T, Jimaa S (2003) High performance parallelised 3GPP turbo decoder. In: Proceedings of the 5th European personal mobile communications conference
19. Zhong Z, Peng T, Zhong Z, Wang W, Liu Z (2010) Hardware implementation of Turbo coder in LTE system based on PICOCHIP PC203. In: Proceedings of the 12th IEEE international conference on communication technology (ICCT)
20. Gilbert F, Thul M, Wehn N (2003) Communication centric architectures for turbo-decoding on embedded multiprocessors. In: Proceedings of the design, automation and test in Europe conference & exhibition (DATE)
21. Wong C-C, Lee Y-Y, Chang H-C (2009) A 188-size 2.1mm2 reconfigurable turbo decoder chip with parallel architecture for 3GPP LTE system. In: Proceedings of the symposium on VLSI circuits, pp 288–289
22. Ilnseher T, Kienle F, Weis C, Wehn N (2012) A 2.15GBit/s turbo code decoder for LTE advanced base station applications. In: Proceedings of the international symposium on turbo codes and iterative information processing (ISTC)
23. Lin C-H, Chen C-Y, Chang E-J, Wu A-Y (2011) A 0.16nJ/bit/iteration 3.38mm2 turbo decoder chip for WiMAX/LTE standards. In: Proceedings of the international symposium on integrated circuits (ISIC)
24. Kim J-H, Park I-C (2009) A unified parallel radix-4 turbo decoder for mobile WiMAX and 3GPP-LTE. In: Proceedings of the IEEE custom integrated circuits conference (CICC)
25. Muller O, Baghdadi A, Jezequel M (2006) ASIP-based multiprocessor SoC design for simple and double binary turbo decoding. In: Proceedings of the design, automation test in Europe conference & exhibition (DATE)
26. Muller O, Baghdadi A, Jezequel M (2010) Parallelism efficiency in convolutional turbo decoding. EURASIP J Adv Signal Process 2010:927920. doi:10.1155/2010/927920
27. Moussa H, Muller O, Baghdadi A, Jezequel M (2007) Butterfly and Benes-based on-chip communication networks for multiprocessor turbo decoding. In: Proceedings of the design, automation test in Europe conference & exhibition (DATE)

28. Murugappa P (2012) Towards optimized flexible multi-ASIP architectures for LDPC/turbo decoding. Ph.D. dissertation, Telecom Bretagne – UBS
29. Murugappa P, Baghdadi A, Jezequel M (2013) Parameterized area-efficient multi-standard turbo decoder. In: Proceedings of the design, automation test in Europe conference & exhibition (DATE)
30. Sun Y, Zhu Y, Goel M, Cavallaro J (2008) Configurable and scalable high throughput turbo decoder architecture for multiple 4G wireless standards. In: Proceedings of the international conference on application-specific systems, architectures and processors (ASAP)
31. Ahmed A, Awais M, Rehman A, Maurizio M, Masera G (2011) A high throughput Turbo decoder VLSI architecture for 3GPP LTE standard. In: Proceedings of the IEEE 14th international multitopic conference (INMIC), pp 340–346
32. Ilnseher T, Kienle F, Weis C, Wehn N (2012) A 2.15GBit/s turbo code decoder for LTE advanced base station applications. In: Proceedings of the 7th international symposium on turbo codes (ISTC)
33. Bickerstaff M, Garrett D, Prokop T, Thomas C, Widdup B, Zhou G, Davis L, Woodward G, Nicol C, Yan R-H (2002) A unified turbo/viterbi channel decoder for 3gpp mobile wireless in 0.18-mm cmos. IEEE J Solid-State Circuits 37(11):1555–1564
34. Thomas C, Bickerstaff MA, Davis LM, Prokop T, Widdup B, Zhou G, Garrett D, Nichol C (2003, April) Integrated circuits for channel coding in 3g cellular mobile wireless systems. IEEE Commun Mag 150–159
35. Kreiselmaier G, Vogt T, Wehn N (2004) Combined turbo and convolutional decoder architecture for UMTS wireless applications. In: Proceedings of the design, automation test in Europe conference & exhibition (DATE)
36. Cavallaro JR, Vaya M (2003) VITURBO: a reconfigurable architecture for Viterbi and turbo decoding. In: Proceedings of the international conference on acoustics, speech, and signal processing (ICASSP)
37. Sun Y, Cavallaro JR (2008) Unified decoder architecture for LDPC/Turbo codes. In: Proceedings of the IEEE workshop on signal processing systems (SiPS)
38. Sun Y, Cavallaro J (2011) A flexible LDPC/turbo decoder architecture. J Signal Process Syst 64(1):1–16
39. Naessens F, Bougard B, Bressinck S, Hollevoet L, Raghavan P, Van der Perre L, Catthoor F (2008) A unified instruction set programmable architecture for multi-standard advanced forward error correction. In: Proceedings of the IEEE workshop on signal processing systems (SiPS)
40. Giuseppe Gentile MR, Fanucci L (2010) A multi-standard flexible turbo/LDPC decoder via ASIC design. In: Proceedings of the 6th international symposium on turbo codes and iterative information processing
41. Dielissen J, Engin N, Sawitzki S, van Berkel K (2008) Multistandard fec decoders for wireless devices. IEEE Trans Circuits Syst II Express Briefs 55(3):284–288
42. Vogt T, Wehn N (2008) A reconfigurable application specific instruction set processor for convolutional and turbo decoding in a sdr environment. In: Proceedings of the design, automation test in Europe conference & exhibition (DATE)
43. Alles M, Vogt T, Wehn N (2008) FlexiChaP: a reconfigurable ASIP for convolutional, turbo, and LDPC code decoding. In: Proceedings of the 5th international symposium on turbo codes and related topics
44. Tarable A, Benedetto S, Montorsi G (2004) Mapping interleaver laws to parallel turbo and ldpc decoders architectures. IEEE Trans Inf Theory 50(9):2002–2009
45. Sani A, Coussy P, Chavet C (2013) A first step toward on-chip memory mapping for parallel turbo and LDPC decoders: a polynomial time mapping algorithm. IEEE Trans Signal Process 61(16):4127–4140
46. Moussa H, Baghdadi A, Jezequel M (2008) Binary de Bruijn on-chip network for a flexible multiprocessor LDPC decoder. In: Proceedings of the 45th design automation conference (DAC)

47. Condo C, Martina M, Masera G (2012) A network-on-chip-based turbo/LDPC decoder architecture. In: Proceedings of the design, automation test in Europe conference & exhibition (DATE)
48. Murugappa P, Lapotre V, Baghdadi A, Jezequel M (2013) Rapid design and prototyping of a reconfigurable decoder architecture for QC-LDPC codes. In: Proceedings of the IEEE international symposium on rapid system prototyping (RSP)
49. Lapotre V, Murugappa P, Gogniat G, Baghdadi A, Diguet J-P, Bazin J-N, Hubner (2013) Optimizations for an efficient reconfiguration of an ASIP-based turbo decoder. In: Proceedings of the IEEE international symposium on circuits and systems (ISCAS)