Parameterized Area-Efficient Multi-standard Turbo Decoder

Purushotham Murugappa, Amer Baghdadi, Michel Jézéquel
Institut Mines-Telecom; Telecom Bretagne; Lab-STICC CNRS UMR 6285
Electronics Department, Technopôle Brest Iroise 29238 Brest France
Email: {firstname.surname}@telecom-bretagne.eu

***Abstract*—Emerging wireless digital communication standards specify a large variety of channel coding options, each suitable for specific application needs. In this context, several recent efforts are being conducted to propose flexible channel decoder implementations. However, the need of optimal solutions in terms of performance, area, and power consumption is increasing and cannot be neglected against flexibility. In this paper we present a novel parameterized architecture for multi-standard Turbo decoding which illustrates how flexibility, architecture efficiency, and rapid design time can be combined. The proposed architecture supports both single-binary Turbo codes (SBTC) of 3GPP-LTE and double-binary Turbo codes (DBTC) of WiMAX and DVB-RCS standards. It achieves, in both modes, a high architecture efficiency of 4.37 bits/cycle/iteration/mm^2. A major contribution of this work concerns the rapid design time allowed by the well established design concept and tools of application-specific instruction-set processors (ASIPs). Using such a tool, the paper illustrates the possibility to design application-specific parameterized cores, removing the need of the program memory and the related instruction decoder.**

I. Introduction

Flexibility has become one of the major design considerations over the last years in many application domains. Digital communication domain is very representative of this trend where many flexible designs have been recently proposed for the challenging Turbo decoding application. For this application, there is a large variety of coding options specified in existing and future digital communication standards, besides the increasing throughput requirement (Table I).

Standard	Codes	Rates	States	Block size (bits)	Throughput (Mbps)
WiMAX	DBTC	1/2, 2/3, 3/4, 5/6	8	.. 4800	.. 75
DVB-RCS	DBTC	1/3 - 6/7	8	.. 1728	.. 2
3GPP-LTE	SBTC	1/3 - 0.95	8	.. 6144	.. 150

TABLE I. Selection of wireless communication standards specifying Turbo codes

In this context, many recent works have been proposed targeting flexible, yet high throughput, implementations of Turbo decoders [1][2][3][4][5]. The flexibility varies from supporting different modes of a single communication standard to the support of multi-standards multi-mode applications. Other implementations have even increased the target flexibility to the support of different channel coding techniques.

However, the real contribution while targeting the flexibility requirement concerns the achieved architecture efficiency in terms of performance/area. In fact, this efficiency is a key differentiator, as if it is not considered, flexibility and throughput can be simply reached by instantiating and duplicating dedicated cores for each standard. To that end, many algorithm, architecture, and technology optimization techniques are investigated in order to combine flexibility and high architecture efficiency. Parallelism techniques, which are necessary to reach the increasing throughput requirements, are analyzed and classified with respect to their area efficiency and parallelism degrees in [1][6]. Regarding the architecture model, the conventional parameterized architecture model can enable to increase the flexibility support of the hardware implementation through a careful control unit design which takes into consideration several input parameters. However, such an architecture model implies long design time as no well structured design methodology, with respect to the flexibility requirement, is available. On the other hand, recent efforts have targeted the use of Application-Specific Instruction-set Processor models (ASIP). Such an architecture model enables the designer to freely tune the flexibility/performance trade-off as required by the considered application. Furthermore, the well established design methodology and the mature available tools enable short design time.

In this paper, we use the ASIP design methodology and tools to implement a novel parameterized core for multi-standard Turbo decoding which illustrates how flexibility, architecture efficiency, and rapid design time can be combined. The proposed architecture (namely TDecASIP) demonstrates the possibility to achieve a high architecture efficiency (4.37 bits/cycle/iteration/mm^2) while using such an approach by selecting the appropriate parallelism and optimization techniques and by removing the need of the program memory and the related instruction decoder. The target flexibility is set to cover both single-binary Turbo codes (SBTC) of 3GPP-LTE and double-binary Turbo codes (DBTC) of WiMAX and DVB-RCS standards. The rest of the paper is organized as follows. Section II gives a brief introduction on the decoding algorithm of Turbo codes. Section III describes the architecture of the proposed parameterized core. The achieved results are summarized and compared along with few recent related works in Section IV. Finally the paper concludes with Section V.

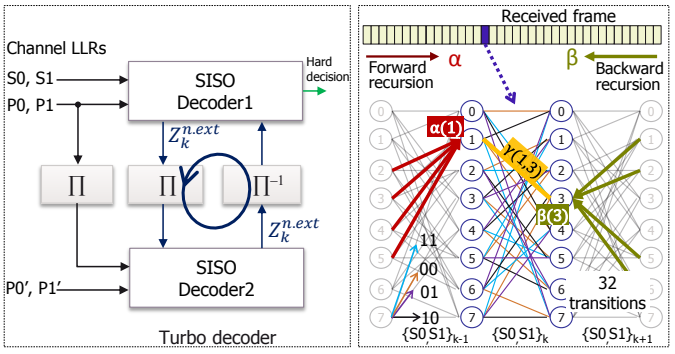

Fig. 1. Turbo codes decoder structure and an 8-states DBTC trellis

II. Turbo Decoding

The typical Turbo decoder structure consists of two Soft Input Soft Output (SISO) component decoders exchanging extrinsic information via an interleaving (Π) and deinterleaving (Π^{-1}) functions as illustrated in Fig. 1. The SISO decoders often implement the hardware-efficient Max-Log-MAP algorithm [7]. In order to explain briefly the underlined computations, let us consider the 8-state DBTC code of WiMAX standard, represented by its trellis in Fig. 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

978-3-9815370-0-0/DATE13/©2013 EDAA

a transition to occur between two trellis states (s': starting state, s: ending state). These branch metrics can be decomposed, as defined by the following expressions, in an intrinsic part ($\gamma_k^{intr_x}(s',s)$) due to the systematic information ($\gamma_k^{sys_x}(s',s)$) and the a priori information ($\gamma_k^{n.apr_x}(s',s)$), besides a redundancy part due to the parity information ($\gamma_k^{par_y}(s',s)$).

$$\gamma_k(s',s) = \gamma_k^{intr_x}(s',s) + \gamma_k^{par_y}(s',s) \quad \forall(x,y = 00,01,10,11) \tag{1}$$

$$\gamma_k^{intr_x}(s',s) = \gamma_k^{sys_x}(s',s) + \gamma_k^{n.apr_x}(s',s) \quad \forall(x = 00,01,10,11) \tag{2}$$

where $\gamma_k^{n.apr_x}(s',s)$ is the normalized a priori information of the k^{th} symbol, or the normalized extrinsic information ($Z_k^{n.ext}$), sent by the other decoder component (expression given below). Furthermore, the systematic and the parity information in these expressions represent the *symbol* Log-Likelihood-Ratios (LLRs) which can be obtained by direct addition and subtraction operations between the received channel *bit* LLRs (S0, S1, P0, P1, P0', P1').

Then the SISO decoder runs the forward and backward recursion over the trellis (Fig. 1).The forward state metrics $\alpha_k(s)$ of the k^{th} 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).

$$\alpha_k(s) = \max_{s'}(\alpha_{k-1}(s) + \gamma_k(s',s)) \quad \forall(s',s = 0,1,..7) \tag{3}$$

$$\beta_k(s) = \max_{s'}(\beta_{k+1}(s) + \gamma_k(s',s)) \quad \forall(s',s = 0,1,..7) \tag{4}$$

Finally, the extrinsic information of the k^{th} 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:

$$Z_k^{apos}(d(s',s)=x) = \max_{(s',s)/d(s',s)=x}(\alpha_{k-1}(s) + \gamma_k(s',s) + \beta_k(s)) \quad \forall(x = 00,01,10,11) \tag{5}$$

$$Z_k^{ext}(d(s',s)=x) = Z_k^{apos}(d(s',s)=x) - \gamma_k^{int_x}(s',s) \quad \forall(x = 00,01,10,11) \tag{6}$$

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.

$$Z_k^{n.ext}(d(s',s)=x) = Z_k^{ext}(d(s',s)=x) - min(Z_k^{ext}(d(s',s)=x)) \quad \forall(x = 00,01,10,11) \tag{7}$$

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.

III. Proposed Parameterized Architecture

In this section we summarize first the design motivations behind the architectural choices we made during the design stage, followed by the description of the proposed parameterized core and memory organization.

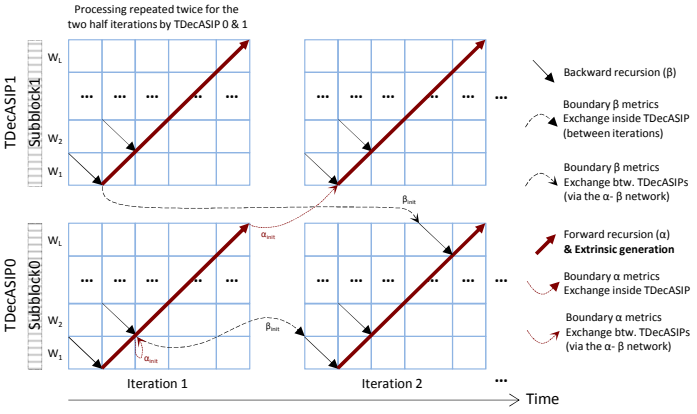

Fig. 2. Windowing and backward-forward schedule

A. Design motivations and design choices

The motivations behind the design choices that were made are as follows. In order to achieve the target throughput in the range of hundreds Mbps, a sub-block parallelism degree of 2 is adopted. In fact, such parallelism degree (which can be also increased to 4) allows for conflict-free memory accesses in both WiMAX and LTE standards for all specified frame length. Each sub-block is further divided into L windows of length W. This reduces the depth of the storage memory required for storage of previous state metrics to W (as required by the equation (4)).

Each TDecASIP uses two recursion units and employs Backward-Forward schedule for window processing. The first recursion unit (processing in the backward direction of the trellis) works on window j while the second recursion unit (processing in the forward direction of the trellis) works on window $j-1$ at the same time (as shown in Fig. 2). This enables to achieve the throughput equivalent to butterfly schedule (as in TurbASIP design) but by using backward-forward schedule which further enables use of hardware interleave address generators for extrinsic memory addressing. In the backward recursion, at the end of processing of the j^{th} window, the boundary state metrics are stored in an external (*BoundaryState*) memory. These state metrics are later used as initial states for the window $(j-1)$ in the subsequent iteration. In TDecASIP, the maximum window size is considered to be W=64 symbols.

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 are 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 [5] (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.

Based on the above design motivations and choices, we propose the two core Turbo decoder architecture shown in Fig. 3. Each core (TDecASIP) processes a sub-block of the input frame and interconnected by two 80-bit ring buses to enable state metric exchanges across sub-blocks. As β state metrics are quantized to 10 bits, 80-bit (for 8 states) wide bus is needed to exchange the boundary state metrics between processors. Each core 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 as illustrated in Fig. 3.

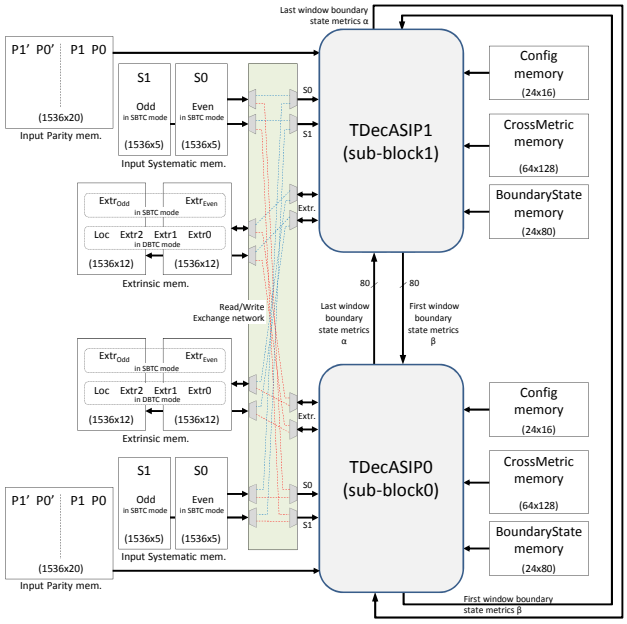

Fig. 3. Overview and memory organization of the proposed 2-TDecASIP Turbo decoder architecture

B. TDecASIP decoder architecture

Fig. 4 details the proposed TDecASIP architecture. The design consists of 8 pipeline stages, of which the first 3 stages are dedicated for the data fetch from the memories and for the control of the pipeline. Since the number of flexible parameters is small, their values are fetched from the configuration memory by the *ConfigFetch* stage. These parameters consist of the following:

- Mode: LTE or DVB/WiMAX.
- The number of iterations to be executed.
- Normal window size (W), size of the last window (W_L) and the number of windows (L).
- Extrinsic address generation initialization values: these values are required to configure the address generation logic in WiMAX/DVB or LTE modes [5].

1) Finite state machine for pipeline control: One of the main motivations of this work concerns the investigation of the possibility to design parameterized cores using the available ASIP design approach. Such possibility can potentially lead to a higher architecture efficiency by simplifying the instruction decoding logic and removing the program memory. Furthermore, it should lead for an increased energy efficiency as there are no program memory accesses in this case. Finally, such an approach still keeps the benefit of the short design cycle enabled by the well established ASIP design tools.

In the proposed modified ASIP design flow for parameterized cores design, the instruction program memory is used as a configuration (config) memory, where the configuration parameters are stored. Rather than defining specialized instructions, the corresponding finite state machine (FSM) is directly described in LISA (Language for Instruction Set Architectures) [9]. 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 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 6 states (as shown in Fig.4) and few flexible parameters that do not change during the decoding process. 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.4).

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. 4 occur when the window boundary is reached. At the end of the window processing, if the number of windows $L = 1$ then the forward recursion is executed for the window currently processed else the control is passed to $S2$ state. In $S2$ state both forward recursion for W_{curr-1} and backward recursion for W_{curr} window are executed in parallel by two dedicated state metrics processing units as shown in Figure 4. Since CrossMetric memory is read and written simultaneously by two different execution units, *Crossmetric* memory bank is chosen to be dual port memory.

When the backward recursion unit completes the processing of all L windows, the forward recursion unit would still be executing W_{L-1}. In case $W_L < W_R$, i.e. the L^{th} window size is less than W_R, a wait state $S3$ is introduced (corresponding to W_R-W_L clock cycles) so that the forward recursion unit can complete the execution of the $(L-1)^{th}$ window before transition to state $S4$. The $S4$ state only generates signals to activate forward recursion of the last window. Once the forward recursion of the last window (W_L) is complete, 5 clock cycles wait state $S5$ is inserted to ensure all the control data are flushed out of the pipeline before starting the next half iteration. During the last half iteration, hard decisions are made on the aposteriori LLRs.

2) Pipeline architecture: The *LLRtoSymbol* pipeline stage converts the fetched systematic and parity bit LLRs to symbol LLRs. If the processor is executing the first half iteration, the least significant 10 bits (P1,P0) of the parity data fetched are used, else the most significant 10 bits (P1',P0') are used for processing. The *BackwardBM* pipeline stage computes the branch metrics $\gamma_k(s', s)$ for the backward

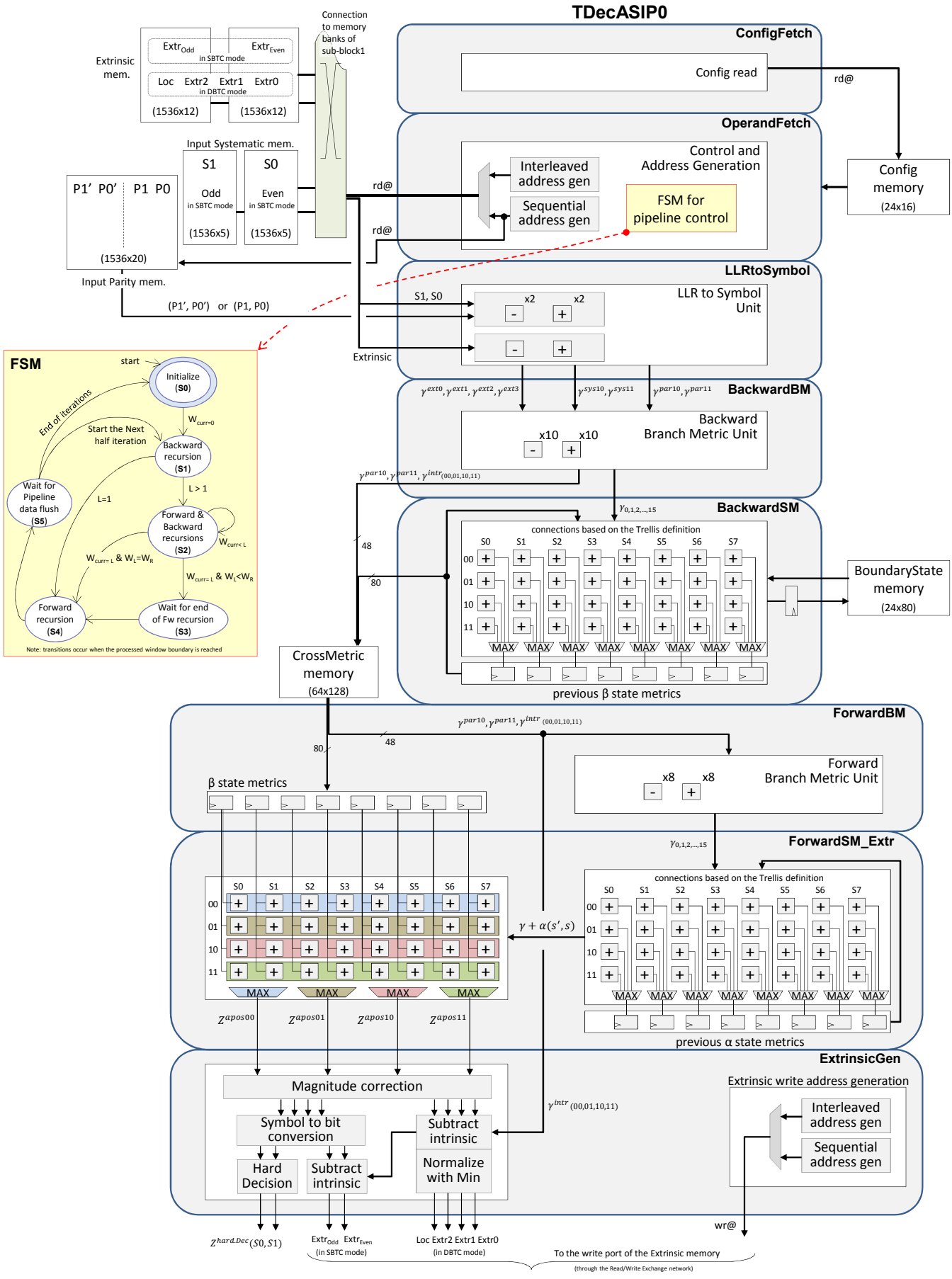

Fig. 4. Detailed pipeline architecture and FSM of the proposed TDecASIP parametrized core

recursion using extrinsic, systematic, and parity symbol LLRs as described by the equation (1).

Then the pipeline stage *BackwardSM* computes the 8 backward state metrics $\beta_k(s)$ corresponding to the received symbol k as defined in equation (4). To that end, the computations related to the 32 trellis transitions (refer Fig.1) are done in parallel using 32 adder nodes and 8 maximum operators as illustrated in Fig. 4. The computed 8 backward state metrics $\beta_k(s)$ are buffered in the CrossMetric memory as they are needed to compute the aposteriori information Z_k^{apos} in the *ForwardSM_Extr* stage. In addition, the CrossMetric memory buffers the $\gamma_k^{intr}(s', s)$ and $\gamma_k^{par}(s', s)$ of the processed window so they can be used directly in the *ForwardBM* pipeline stage for the computation of the branch metrics $\gamma_k(s', s)$ in the forward recursion (avoiding external memory accesses and double-ported input memories). State metrics are quantized to 10 bits while the intrinsic LLRs γ_i^{intr} need an 8 bits quantization.

The *ForwardSM_Extr* pipeline stage includes all the required hardware units (Fig. 4) to compute the forward state metrics $\alpha_k(s)$ as defined in equation (3) and to complete the computation of the 4 aposteriori LLRs Z_k^{apos} of the symbol k after fetching the $\beta_k(s)$ from the CrossMetric memory. Overflows are allowed in the state metric calculations $(\alpha(s), \beta(s))$ and the magnitude correction unit of the *ExtrinsicGen* pipeline stage implements the modulo normalization (as done in [10][1]). Finally, the magnitude corrected aposteriori Z_k^{apos} and the intrinsic $\gamma_k^{intr_x}$ LLRs are used to generate the normalized extrinsic (for DBTC or SBTC modes) and the hard decision.

C. Memory organisation

The memory organization of the proposed architecture is illustrated in Fig.3. 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 6144 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.3.

IV. Results and Discussions

The proposed parameterized core was modeled with Synopsys Processor Designer tool and the corresponding VHDL description was generated and synthesized targeting 65nm general purpose CMOS technology (worst case 0.9v and 125C). Table II summarizes the memory partitions and the post-synthesis logic and memory area results obtained for a single core. All the memories used are single port (sp) memories except for the CrossMetric and extrinsic memories which are double port (dp) memories. 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 design presented in this paper is 0.437 mm^2 for 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 when using BPSK modulation over an additive white gaussian noise (AWGN) channel (Fig. 5). If the frame length is N bits and the window size is W symbols, then the throughput of the proposed Turbo decoder is given by:

$$Throughput = \frac{Num_{procs} \times N \times f_{clk}}{((\frac{\lceil N_{sym}/W \rceil}{Num_{procs}} + 1) \times W + N_{pip}) \times (2 \times N_{iter})} \quad (8)$$

For the presented architecture: $Num_{procs} = 2$ processors, the maximum clock frequency is $f_{clk} = 510$MHz, considering the largest LTE frame size $N_{sym} = 3072$ symbols or $N = 6144$ bits and $N_{iter} = 6.5$ iterations, the throughput obtained is $Throughput = 150$ Mbps.

Design unit	Area (um^2)
ConfigFetch	191
OperandFetch	6586
LLRtoSymbol	957
BackwardBM	1905
BackwardSM	10038
ForwardBM	2480
ForwardSM_Extr	17847
ExtrinsicGen	5006
RegisterFile	13695
MemoryInterface	6683
Total logic area	65390
Total mem area	153478
Total area	218868

(a) Synthesis results

Memory	width (bits)	depth	# banks	type
Systematic	5	1536	2	sp
Parity	20	1536	1	sp
Extrinsic	12	1536	2	dp
CrossMetric	128	64	1	dp
BoundaryState	80	24	1	sp
Config	16	24	1	sp

(b) Memory partitions

TABLE II. Area utilization per TDecASIP in the 2-TDecASIP architecture, with CMOS 65nm target technology

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:

$$AE = \frac{Throughput \times N_{iter}}{Area_{Norm} \times f_{clk}} \quad (9)$$

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. 180nm, 130nm, 65nm, 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

	This work		[2]	[3]	[4]
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]
$Area_{Norm}$ @65nm (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&route

TABLE III. RESULTS AND COMPARISON WITH WITH FEW RECENT RELATED WORKS

to the usually (or even worst case) observed ratio. Considering this definition, 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 $0.097mm^2$ which results in a total area occupancy of $0.65mm^2$. 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 the proposed design are summarized and compared along with few recent related works in Table III. The cited three implementations [2][3][4] 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 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 [2] which supports both Turbo modes (DBTC and SBTC) and uses 8 parallel MAP decoders. The LTE-dedicated implementations presented in [3] and [4] exploit the available higher sub-blocking parallelism degrees in this standard (parallel interleaving with conflict-free memory accesses). Results comparison illustrates how the proposed architecture achieves a high architecture efficiency while using such an ASIP-based parameterized core approach by selecting the appropriate parallelism and optimization techniques.

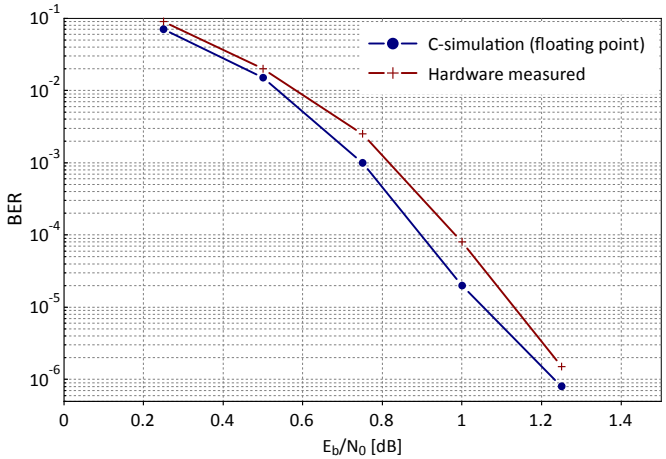

Fig. 5. Error rate performance comparison between the hardware implementation and the floating point simulation for WiMAX frame size 1920 bits

V. CONCLUSION

In this paper, we illustrated how flexibility, architecture efficiency, and rapid design time can be combined when using an ASIP design methodology and tools to implement a novel parameterized core for multi-standard Turbo decoding. The proposed architecture demonstrates the possibility to achieve a high architecture efficiency while using such an approach by selecting the appropriate parallelism and optimization techniques and by removing the need of the program memory and the related instruction decoder. The presented two core Turbo decoder achieves 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 (5.88 bits/cycle/iteration/mm^2 with a four core Turbo decoder architecture). The target flexibility was set to cover SBTC of 3GPP-LTE and DBTC of WiMAX/DVB-RCS standards.

REFERENCES

[1] O. Muller, A. Baghdadi, and M. Jezequel, "From Parallelism Levels to a Multi-ASIP Architecture for Turbo Decoding," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 17, no. 1, pp. 92 –102, 2009.

[2] J.-H. Kim and I.-C. Park, "A unified parallel radix-4 turbo decoder for mobile WiMAX and 3GPP-LTE," in *Proc. of the IEEE Custom Integrated Circuits Conference (CICC)*, 2009, pp. 487 –490.

[3] A. Ahmed, M. Awais, A. Rehman, M. Maurizio, and G. Masera, "A High Throughput Turbo Decoder VLSI Architecture for 3GPP LTE Standard," in *Proc. of the IEEE 14th International Multitopic Conference (INMIC)*, 2011, pp. 340–346.

[4] T. Ilnseher, F. Kienle, C. Weis, and N. Wehn, "A 2.15GBit/s Turbo Code Decoder for LTE Advanced Base Station Applications," in *Proc. of the 7th International Symposium on Turbo Codes (ISTC)*, 2012.

[5] Y. Sun, Y. Zhu, M. Goel, and J. Cavallaro, "Configurable and scalable high throughput turbo decoder architecture for multiple 4G wireless standards," in *Proc. of the Inter. Conf. on Application-Specific Systems, Architectures and Processors (ASAP)*, 2008, pp. 209 –214.

[6] O. Muller, A. Baghdadi, and M. Jezequel, "Parallelism Efficiency in Convolutional Turbo Decoding," *EURASIP Journal on Advances in Signal Processing*, 2010.

[7] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and Sub-Optimal Maximum A Posteriori Algorithms Suitable for Turbo Decoding," *European Transactions on Telecommunications (ETT)*, vol. 8, no. 2, pp. 119–125, 1997.

[8] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, "A 24Mb/s radix-4 logMAP turbo decoder for 3GPP-HSDPA mobile wireless," in *Proc. of the IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 1, 2003, pp. 150–484.

[9] A. Hoffmann, O. Schliebusch, A. Nohl, G. Braun, O. Wahlen, and H. Meyr, "A methodology for the design of application specific instruction set processors (ASIP) using the machine description language LISA," in *Proc. of ICCAD*, 2001, pp. 625–630.

[10] A. Hekstra, "An alternative to metric rescaling in Viterbi decoders," *IEEE Trans. on Comm.*, vol. 37, no. 11, pp. 1220 –1222, nov 1989.