Optimizations for an Efficient Reconfiguration of an ASIP-Based Turbo Decoder

Vianney Lapotre*, Purushotham Murugappa†, Guy Gogniat*, Amer Baghdadi†, Jean-Philippe Diguet*, Jean-Noël Bazin † and Michael Hübner ‡
*Université de Bretagne Sud, Lab-STICC UMR 6285, Lorient, France. Email: firstname.lastname@univ-ubs.fr
†Institut Telecom, Telecom Bretagne, CNRS Lab-STICC UMR 6285, Brest, France. Email: firstname.lastname@telecom-bretagne.eu
‡Rurh-Universitt Bochum, ESIT, Bochum, Germany. Email: michael.huebner@rub.de

***Abstract*—The multiplication of wireless standards is introducing the need of flexible multi-standard baseband receivers. A multi-ASIP approach for turbo decoding is an answer to reach high throughput and high flexibility. The increasing demand of throughput for new greedy application on mobile devices and the reduction of latency between two frames create the need of an efficient reconfiguration management of such multi-ASIP platforms. In this paper, we propose to tackle reconfiguration optimization of a multi-standard ASIP for turbo decoding developed during previous work. Results show that for an area overhead of 0.012 mm^2 in 65 nm CMOS technology, a significant reconfiguration time optimization is achieved thanks to a reduction of the ASIP configuration load of 70%. Moreover, in a multi-ASIP context in which 8 ASIPs are implemented the configuration load is divided by ten thanks to the possibility to use a multicast mechanism for ASIP configuration loading.**

I. Introduction

The evolution of recent wireless communication standards aims at increasing the requirements in terms of throughput, robustness against destructive channel effects and convergence of services in a smart terminal. Turbo codes [1] are frequently adopted in these wireless standards to reach a low bit error rate. The increasing throughput requirement often imposes the efficient exploitation of different parallelism levels. In this context, multi-ASIP (Application-Specific Instruction-set Processor) architectures for turbo decoding [2], [3], [4] is a promising approach to reach high flexibility, high throughput and energy efficiency. In [2] and [3], authors propose to implement the ASIP described in [5] in order to build a flexible multi-ASIP based turbo decoder for LTE requirements. This ASIP is configured through an interleaver memory, a program memory and the dynamically Reconfigurable Channel Code Control (DRCCC). The DRCCC is a look-up table based unit which allows the configuration of the structure of the convolutional code, the internal data-path, and the configuration memory. Two configurations are stored in this unit, a *working* and a *shadow* configuration. The working configuration holds the parameters that are actually used while the shadow configuration is used to prepare the next configuration. One cycle switching can be performed between these two configurations thanks to a special instruction. However, using a specific instruction in the program to switch between two configurations limits the flexibility because the reconfiguration scenario is defined statically. In [4], authors present the *UDec* architecture. It consists of 8 ASIPs interconnected via a Network on Chip (NoC). Within each component decoder ASIPs are also connected by a ring network for metric exchanges. Each ASIP is configured through a program and a configuration memory. The configuration memory contains several communication parameters which are loaded into internal registers of the ASIP during an initialization step, while the program describes the control flow for the initializing loop and the decoding loops.

Unfortunately, the dynamic reconfiguration aspect of these platforms is superficially addressed. All these platforms can be reconfigured through program and configuration memories but the configuration mechanisms are not optimized for an efficient implementation in a multi-core system. In this paper, we propose original optimizations for an efficient reconfiguration of multi-ASIP architectures. The proposed approach is illustrated through the flexible DecASIP core described in [4].

The rest of this paper is organized as follows. Section II introduces the turbo decoding parameters and describes the DecASIP. Section III presents the proposed optimizations to reach an efficient configuration of the multi-ASIP platform. Section IV describes the implementation of the optimizations into the DecASIP processor and the evaluation of the (re)configuration performance. Finally, section V concludes the paper.

II. DecASIP

The DecASIP [4] implements the Max-Log MAP algorithm as described in [6]. It supports convolutional turbo codes up to eight-state double binary codes or sixteen-state single binary codes. Large frames are processed by dividing the frame into N windows each with a maximum size of 64 symbols. Each ASIP can have a maximum of 12 windows. The DecASIP is configured through a program memory and a configuration memory that respectively contain the instructions to perform the decoding algorithm and all parameters required to perform the initialization of the ASIP. Since the DecASIP is designed to be integrated in a multi-ASIP architecture as described in [4], it requires several parameters to deal with a subblock of the frame and several parameters to configure the ASIP mode. Concerning the subblock partitioning, each ASIP is configured with the size and the number of windows it has to decode. Furthermore, the last window size can be different so it corresponds to an additional parameter. In a single binary turbo code mode, the address of the tail bits in memory, the size and the number of windows for the tail bits have to be configured. Parameters for the ASIP mode correspond to the location of the ASIP in the architecture, the number of ASIPs required, the parameter which defines if the current ASIP is in charge of tail bits or not, the target standard (3GPP-LTE, WIMAX, or DVB-RCS) and the scaling factor for extrinsic information. Finally, some seed values are necessary for address generation in order to exchange information over the NoC that connects the ASIPs of each decoder component.

The work presented in [4] on the proposal of the DecASIP architecture focuses simply on providing the target flexibility and performance requirements in terms of throughput and area efficiency. Indeed, the topic of dynamic reconfiguration is not addressed. Despite of the proposed high flexibility, it presents some lacks to offer an efficient reconfiguration. The next section points out theses lacks and proposes solutions to implement an efficient reconfigurable ASIP in a multi-ASIP architecture context.

978-1-4673-5762-3/13/$31.00 ©2013 IEEE

bit	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
@0	-																					Tail	ASIPId			
@1	Turbo Seed 0													Turbo Seed 1												
@2	-						TurboInitIteration				Maxiteration								State				NumSteps			
@3	Turbo Step 0													Turbo Step 1												
@4	Turbo Step 2													Turbo Step 3												
@5	Turbo Step 4													Turbo Step 5												
@6	Turbo Step 6													Turbo Step 7												
@7	-		@ Tail bits													Scaling Factor							Mode			
@8	Turbo PrevStep													Blocklength in bits												
@9	-			NumASIPs				StepIndex			WindowSize								LastWindowSize							
@10	-		CurrentWindowN_norm								CurrentWindowID_tail								WindowN_tail							

Table I
CONFIGURATION MEMORY ORGANIZATION

III. PROPOSED OPTIMIZATIONS

We propose to reach an efficient dynamic reconfiguration of the DecASIP through several optimization techniques. The first technique concerns the storage optimization of configuration parameters. The second technique proposes a new configuration memory organization. The third technique introduces a generic program independent of the configuration to be performed and tackles the management of interframe delay since the ASIP is currently reseted after each processed frame. The final technique deals with the multi-configuration storage.

A. Configuration parameters storage

To reach (re)configuration efficiency, we propose to move all parameters defined in the instruction code words to the configuration memory. This solution allows to (re)configure only one memory to change all configuration parameters. Furthermore, when the ASIP is configured, the configuration memory can be accessed without any conflict since the configuration parameters are transferred into internal registers of the ASIP during the initialization step. Thus, the configuration transfer can be partially or completely masked by loading the next configuration in configuration memory during the processing of the current frame. If the configuration loading is completely masked, the configuration overhead consists in the initialization step only. The next sub-section presents a way to integrate the new parameters in a smart memory organization.

B. Configuration memory organization

In order to improve the (re)configuration of the ASIP, it is essential to analyze the organization of the configuration memory. The configuration parameters stored in the configuration memory can be divided in four categories: 1) domain dependent, 2) identical for all ASIPs, 3) different for all ASIPs and 4) different for the last ASIPs which decode the tail bits in single binary turbo code mode. A smart memory organization should allow an efficient broadcasting of the configuration parameters to the required ASIPs. According to the previously described categories, we create four groups which occupy four different parts of the configuration memory.

Table I shows configuration memory organization. It is built as follows: (1) from address @0 to @1, parameters can be different for each ASIP. Furthermore, to optimize the initialization step of ASIP, the parameter *Tail* which indicates if the ASIP has to perform or not the tail bits is included in this group. Only the last two ASIPs are concerned by the tail bits in single binary turbo code mode; (2) from address @2 to @6, parameters stored are domain dependent; (3) from address @7 to @10, parameters stored are the same for all ASIPs. This organization allows a good way for a fast reconfiguration at the platform level. Indeed, multicast mechanisms can be used to load the configuration in a multi-ASIP architecture in order to minimize data transfers load. In this context, two multicast transfers are necessary to send domain dependent parameters to all ASIPs and one multicast transfer for parameters that are the same for all ASIPs. Finally, unicast transfers are used to load the ASIP dependent parameters.

C. Generic program

Three different programs are currently used in the DecASIP: two programs for single binary turbo code and one program for duo binary turbo codes. In single binary mode, after the initialization step, the last two ASIPs have to perform the tail bits while other ASIPs execute NOP operations. So, a particular program is loaded in these last two ASIPs. In duo binary mode, the frames are decoded after the initialization step. We propose to merge these three programs to one unique program presented in Table II. This solution allows to consider a single program for different configurations and thus reduces the number of reconfigurations. For this purpose, the program which integrates the tail bits computation is used as a reference program. From instruction 1 to 4 in Table II, the initialization phase of the ASIP is performed. When the *ASIP_INIT* is executed, one configuration memory line is read and corresponding internal registers are initialized. The instruction *Repeat until* owns 2 parameters. The first one marks the boundary of the loop while the second one indicates that the number of iterations of the loop depends of the configuration size. The new memory organization previously described requires 11 read accesses. Hence the initialization step requires 13 clock cycles. (i.e. one clock cycle to read the memory, one more clock cycle to initialize the loop, and another clock cycle to complete the initialization of the ASIP). Instructions from line 5 to 16 concern the tail bits decoding in single binary mode. In duo binary mode, no tail bits have to be decoded. In order to minimize the impact on the complexity of the ASIP by adding new control instructions, we have chosen to modify the *Fetch* pipe stage in order to detect and replace the instructions for tail bits with *NOP* instructions if the ASIP is not concerned. Hence, using a unique program in this mode adds 12 extra *NOP* instructions before the decoding which corresponds to tail bits computation in single binary mode. However, the extra introduced clock cycles are negligible regarding the number of cycles required to perform the decoding on one entire frame. Instructions from line 17 to 40 concern the decoding of the frame. This part of the program is common for single and duo binary turbo code modes. In order to create a generic program, the instructions *Repeat until*,

k	Label	Instruction
1		Repeat until _init 0
2		NOP
3		ASIP_INIT
4	_init:	NOP
5		SET_WindowsN 0
6		SET_WindowsInit 0
7		TailBits
8		ZOLB _RW0,_RW0,_LW0
9		NOP
10		DATA LEFT ADD metric column2
11	_RW0:	NOP
12	_LW0:	NOP
13		EXCH_BETA_ALPHA
14		NOP
15		NOP
16		NOP
17		SET_WindowsN 1
18		SET_WindowsInit 1
19		Repeat until _LOOP 1
20		PUSH
21		ZOLB _RW1,_CW1,_LW1
22		NOP
23		DATA LEFT ADD metric column2
24	_RW1:	NOP
25		EXCH_BETA_ALPHA
26	_CW1:	NOP
27		DATA RIGHT ADD metric column2
28	_LW1:	EXTCALC ADD info line2
29		WINDOW_INIT ALPHABETA_0_3
30		NOP
31		WINDOW_INIT ALPHABETA_4_7
32		HARD_DECISION
33		NOP
to		NOP
39		NOP
40	_LOOP:	NOP
41		ASIP_STOP

Table II
GENERIC PROGRAM ASSEMBLY CODE

SET_WindowsN and *SET_WindowsInit* have to be independent of the configuration parameters. For this purpose, these instructions own an additional parameter that changes depending of the context. This parameter indicates which configuration parameter stored in internal register has to be used with this instruction. For example, the instruction *Repeat until _LOOP* line 1 is associated with the additional parameter *0*. When this instruction is read, the number of iterations of the loop is read in the internal register that contains the configuration size while it is read in the register that contains the maximum number of decoding iterations when this additional parameter is *1* (line 19 in Table II).

In order to allow the decoding of consecutive frames without reconfiguration, the instruction *ASIP_STOP* in line 41 tackles the interframe management. It is used to trigger the reset of all internal registers which are read during the decoding of the first symbol of a new frame. This reset avoids interferences caused by previous data generated during decoding of the previous frame. Afterwards, ASIP is stopped, and waits for a new trigger on the *Restart* pin to perform a new frame with the same configuration without a new initialization phase of the ASIP. The *Reset* pin is used when a new configuration has been loaded in the configuration memory.

D. Multi-configuration

Recent wireless mobile terminals deal with several communication standards and are able to concurrently execute applications that simultaneously access to the network. In this context, it is interesting to have several configurations available in the memory and to switch between one to another quickly. Furthermore, when configurations are often executed, storing these configurations in the configuration memory saves data transfers to load the memory and consequently reduces the penalty due to configuration transfers. For this purpose, we need to design the ASIP to manage these different configurations. A simple way to address this point, is to add in the configuration memory the address of the configuration that has to be loaded into the ASIP. This address is stored in the first address of the configuration memory. So, during the initialization step, the ASIP reads the first data from the configuration memory which corresponds to the address of the next configuration in memory to be loaded. Then the configuration parameters are read in the configuration memory starting from this address and the ASIP internal registers are configured.

	Pipeline	Register file	Total
DecASIP	0.078	0.076	0.175
New ASIP	0.080	0.087	0.187
Diff.	0.002 +2.6%	0.011 + 14.5%	0.012 +6.8%

Table III
ASIP AREA COMPARISON IN mm^2

Optimizations described in this section allow to reduce the (re)configuration impact: 1) locally through the optimization of the storage of configuration parameters to efficiently use the memory capacity and through the possibility to decode several frames without a new initialization step of the ASIP and 2) globally thanks to the new memory organization and the generic program which reduce the total configuration load to transfer when a new configuration has to be performed. Moreover, the multi-configuration management is becoming a key feature to optimize the management of a mobile device which deals with several communication standards and applications. The next section presents the implementation and the impact of proposed optimizations in the DecASIP.

IV. IMPLEMENTATION AND RESULTS

A. ASIP implementation

Optimizations described in Section III have been implemented on the DecASIP presented in [4]. Interframe management has been implemented by adding extra input pins to the ASIP. These pins can be driven to indicate the address of the next configuration in the configuration memory. This choice has been done to reduce the development time of the new ASIP. Moreover, to provide more flexibility for future configuration organization, input pins have been added to inform the ASIP about the size of the configuration. The ASIP was modeled in LISA language using Synopsys Processor Designer [7]. Synthesis of the previous and the new cores was done with 65nm CMOS technology with a clock frequency objective equals to 500MHz. Synthesis results have been extracted to determine the impact of the optimizations on the pipeline and the register file of the ASIP.

Regarding results from Table III, the global logic overhead on the ASIP is 6.8% (0.012 mm^2). The new pipeline of the ASIP introduces an area overhead of 2.6% (0.002 mm^2). This overhead is spread along the low complex pipeline stages (i.e. pre-fetch, fetch, decode, and operand fetch) that are in charge of one or two optimizations. This implementation limits the impact on the area on the complex pipeline stages in charge of the decoding treatments. On the register file side, interframe management and the larger number of parameters to be stored in internal registers

	Config. parameters	Program memory	1 ASIP	n ASIPs
New ASIP	286	-	286	n.52+260+104
DecASIP	336	640	976	n.976
Gain	14%	100%	70%	90% ($n = 8$)
[5]	383 + Interleaver	$\sim$1080	1463 + Interleaver	n.1463 + n.Interleaver
Gain	25%	100%	80%	93% ($n = 8$)

Table IV
CONFIGURATION AND PROGRAM BIT LOAD COMPARISON IN BITS

cause a significant impact on the register file area. This increasing is around 14.5% (0.011 mm^2). The area overhead is mainly due to the additional connections and registers used to re-initialize the metrics register between two frames.

B. Dynamic reconfiguration performance

In this section, we evaluate the gain of proposed optimizations on the reconfiguration timing performance. For this purpose, we consider the following reconfiguration steps: 1) The transfer of configuration parameters in the configuration memory of one or several ASIPs in a multi-ASIP context. 2) The ASIP starts the initialization process. During this step, the ASIP reads the configuration stored in configuration memory and initializes internal registers. Then, the ASIP is ready to execute the computation on the input frame. 3) Once the computation on a frame is done, two possible scenarii can appear: the same configuration is used for the next frame or a new configuration is required. If the same configuration is used the *Restart* input pin is triggered when the input data is available, else, the steps 1) and 2) are performed again.

Table IV compares the configuration and program load (in bits) for the proposed ASIP, the original DecASIP presented in [4] and the ASIP presented in [5]. For one ASIP, we observe that the proposed ASIP can be configured with 286 bits instead of 976 bits thanks to the generic program described in Section III-C while 1463 bits and the complete interleaver table are required for the ASIP in [5]. Moreover, the new memory organization proposed in Section III-B allows the optimization of the configuration memory loading. Indeed, parameters can be sent to several ASIPs through a multicast mechanism. Thus, in a multi-ASIP context, each original ASIP has to be configured with its own configuration and program memory while configuration memory of the proposed new ASIP can be loaded using a multicast mechanism as follows: 52 bits are independently loaded in each ASIP. ASIPs that compose the same decoder component are loaded with 130 common bits. Finally, 104 configuration bits are broadcasted to all ASIPs. Thus, the impact of the number of ASIPs on the configuration load is significantly reduced: n.52 bits instead of n.976 bits, where n is the number of ASIPs implemented. For example, if 8 ASIPs are implemented in a multi-ASIP platform, the configuration load is 7808 bits with the original DecASIP, 11704 bits for [5] plus the interleaver tables and 780 bits with the proposed ASIP. Thus, in this case the configuration load is divided by 10 and 15 compared the DecASIP and the ASIP from [5] respectively.

The new configuration memory organization also impacts the initialization time of the ASIP by reducing the number of read accesses to the memory. Only 11 read accesses are necessary instead of 15 in the original ASIP. Moreover, thanks to the new interframe management presented in Section III-C, the proposed ASIP can execute the same configuration without a new initialization step. When the configuration corresponding to the next frame is already stored in the configuration memory, the reconfiguration process consists in: 1) drives the input pins of the ASIP to indicate the location of the next configuration and 2) reset the ASIP to launch the initialization step. In this case the ASIP can be reconfigured in 15 clock cycles (with 13 clock cycles to initialize the ASIP) if we assume that two cycles are necessary to drive the input pins of the ASIP which define the configuration location in the memory and launch the ASIP. Furthermore, when the initialization step is performed, the ASIP does not read the configuration memory until the next initialization. Hence, during the computation on a frame, configuration parameters can be loaded in the configuration memory. Hence, the memory loading process can be totally masked. If it is masked, the complete (re)configuration of a multi-ASIP platform represents an overhead of around 15 clock cycles. This low (re)configuration time overhead allows the implementation of such an optimized ASIP in multi-ASIP architecture for future high throughput and low latency requirements.

V. CONCLUSION

Multi-ASIP architectures for turbo decoding is a promising approach to reach high flexibility and high throughput requirements. However, configuration of such multi-core platforms is becoming a critical point. The configuration optimizations presented in this paper allow the reduction of the configuration load of 70% compared to the original ASIP. Furthermore, in a multi-ASIP context implementing 8 ASIPs, the proposed smart configuration memory organization allows dividing the configuration load by 10 using a multicast mechanism. Future work targets the implementation of a configuration interconnect structure implementing multicast mechanism and a configuration manager managing the configuration decision and generation in a flexible multi-ASIP architecture.

REFERENCES

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

[2] C. Brehm, T. Ilnseher, and N. Wehn, "A scalable multi-ASIP architecture for standard compliant trellis decoding," in *International SoC Design Conference (ISOCC)*, 2011, pp. 349 –352.

[3] T. Vogt, C. Neeb, and N. Wehn, "A reconfigurable multi-processor platform for convolutional and turbo decoding," in *Proc. of International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2006, pp. 16–23.

[4] P. Murugappa, A.-K. R., A. Baghdadi, and M. Jézéquel, "A Flexible High Throughput Multi-ASIP Architecture for LDPC and Turbo Decoding," in *Proc. of Design, Automation and Test in Europe Conference & Exhibition (DATE)*, 2011.

[5] T. Vogt and N. Wehn, "A reconfigurable asip for convolutional and turbo decoding in an sdr environment," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 10, pp. 1309 –1320, oct. 2008.

[6] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding," *European Transactions on Telecommunications*, vol. 8, no. 2, pp. 119–125, 1997. [Online]. Available: http://dx.doi.org/10.1002/ett.4460080202

[7] Synpsys homepage, [Online]. Available:http://www.synopsys.com.