Stopping-free dynamic configuration of a multi-ASIP turbo decoder

Vianney Lapotre, Purushotham Murugappa Velayuthan, Guy Gogniat, Amer Baghdadi, Michael Hubner, Jean-Philippe Diguet

Stopping-free dynamic configuration of a multi-ASIP turbo decoder

Vianney Lapotre*, Purushotham Murugappa†, Guy Gogniat*, Amer Baghdadi†, Michael Hübner ‡ and Jean-Philippe Diguet*
*Univ. Bretagne Sud, UMR6285, Lab-STICC, F56100 Lorient, France. Email: firstname.lastname@univ-ubs.fr
†Telecom Bretagne, UMR6285, Lab-STICC, F29200 Brest, France. Email: firstname.lastname@telecom-bretagne.eu
‡Rurh-Universität Bochum, ESIT, Bochum, Germany. Email: michael.huebner@rub.de

***Abstract*—The multiplication of wireless standards is introducing the need of flexible and reconfigurable multistandard baseband receivers. At the physical layer, multiprocessor turbo decoders have been recently developed in order to provide an answer to the increasing throughput requirement of emerging standards. However these solutions do not sufficiently address reconfiguration performance issues which can be a limiting factor in the future. This work focuses on the design of a reconfigurable multiprocessor architecture for turbo decoding achieving very fast reconfiguration without compromising decoding performances. Dynamic reconfiguration can be performed within a single frame decoding duration opening new perspective for reconfigurable multistandard baseband receivers. For that purpose, optimizations at the processing element level and a novel bus-based configuration infrastructure are proposed. Results show that up to 64 processings elements can be dynamically configured in 5.352 μs. This low configuration latency corresponds to a single frame decoding duration when performing 6 decoding iterations for a throughput up to 666 Mbps.**

***Keywords*-ASIP; Dynamic configuration; Turbo decoder; Wireless communications; ASIP;**

I. Introduction

In the last years, multi-modes wireless communication standards have been developed in order to reach higher requirements in terms of throughput, robustness against destructive channel effects and convergence of services in a smart terminal. Forward error correction codes constitute a key feature of a wireless standard. Turbo codes are frequently adopted in recent wireless standards to reach a low bit error rate (BER). The increasing throughput requirement often imposes the efficient exploitation of the different levels of parallelism which have been introduced in multi-modes, multi-standards and multiprocessor decoders [1], [2], [3], [4] in order to offer high throughput and high flexibility. These architectures implement subblock parallelism [5] where each frame is divided into subblocks and then each subblock is processed in parallel using adequate initializations. In this context, flexible ASIP (Application Specific Instruction set Processor) based multiprocessor architectures [1], [2], [6] have been explored. Multi-ASIP architecture is a promising approach to reach high flexibility, high throughput and energy efficiency. An ASIP based decoder implementing several ASIPs described in [7] and a LTE accelerator in order to build a flexible turbo decoder for LTE requirements has been presented in [6]. In [1], the authors present a NoC based multi-ASIP architecture in which ASIPs can be grouped to perform independent decoding tasks in parallel. In [2], authors present the *UDec* architecture. It consists of ASIPs (named DecASIPs) supporting both Single Binary Turbo Codes (SBTC) and Duo Binary Turbo Codes (DBTC) interconnected via a Network on Chip (NoC). Within each component decoder ASIPs are also connected by a ring network for boundary state metric exchanges.

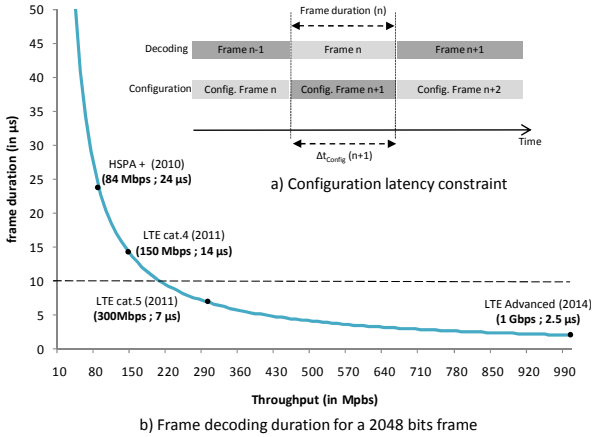

Figure 1. Configuration latency constraint and frame duration

Dynamic reconfiguration of such platforms is particularly challenging as in many standards and/or applications decoding parameters can be changed as early as one data frame ahead [8], thus it becomes mandatory to aggressively improve reconfiguration times in order to perform the reconfiguration of the whole multiprocessor platform within a single frame decoding duration. Fig. 1.a illustrates a scenario in which three frames are serially decoded with three different configurations. The configuration latency of a frame (noted Δt_{Config}) is constrained by the previous frame decoding duration. Fig. 1.b shows the frame duration for a frame size of 2048 bits depending on the decoding throughput. This example demonstrates that the available time to decode a frame critically decreases when the throughput requirement increases. Consequently, reaching a reconfiguration time below 10 μs will be a key concern to face expected throughput of future communication standards like LTE Advance which provides throughput up to 1Gbps. Management of dynamic reconfiguration of such multiprocessor channel decoder is not well addressed in the literature. In [1], [2], [4], [6], flexible multiprocessor decoders are presented but no configuration infrastructure is proposed to deal with low latency reconfiguration of future communication standards.

Among the few related works, we can cite the recent architecture presented in [9] where the authors propose solutions for the reconfiguration management of the multiprocessor Turbo/LDPC decoder architecture presented in [3]. Up to 35 processing elements (PEs) and up to 8 configuration buses have been implemented. Each PE is configured through a configuration memory. Groups of four PEs are connected to a dedicated configuration bus. The remaining PEs are shared among the buses when the number of PEs is not divisible by four. Dynamic reconfiguration during one frame duration is possible when the current configuration is small enough to load a new configuration in the memory. If not, authors provide management solutions to deal with this issue, such as erasing the current configuration during the last decoding iteration and continue the reconfiguration process during the first iteration of the new configuration, but it is not always sufficient. Then, stopping the current treatments to configure the new configuration is unavoidable and leads to a decoding quality loss in terms of BER. Moreover, the cost of the proposed multi-bus configuration infrastructure becomes too high with the increasing number of PEs and leads to a complex configuration transfer management. To leverage these issues, it becomes essential to propose original solutions for a low complexity and stopping-free configuration of multiprocessor turbo decoders.

Based on the work presented in [2], where a flexible multi-ASIP turbo decoder implementation is proposed, this paper aims to further investigate and optimize the reconfiguration process in order to support stopping-free dynamic reconfiguration for high throughput requirements and high level of parallelism. Configuration latency below 10 μs must be proposed to reach this objective for emerging and future communication standards as shown in Fig. 1. This work features the following contributions: i) optimization of the DecASIP [2] processing element for configuration efficiency in a multi-ASIP context, and ii) efficient bus-based configuration interconnection structure for a low configuration transfer latency and low area overhead.

The rest of this paper is organized as follows. Section II introduces the UDec architecture. Section III presents the proposed optimizations to reach an efficient configuration of the DecASIP in a multi-ASIP context. Section IV highlights the main issues that need to be addressed in order to build an efficient configuration infrastructure while section V describes the proposed bus-based solution. Section VI presents the implementation results. Finally section VII concludes the paper.

II. Multi-ASIP turbo decoder

The *UDec* turbo decoder architecture [2] is shown in Fig 2. It consists of two rows of DecASIPs interconnected via a butterfly Network on Chip [10]. Each row corresponds to a component decoder. In the example of Fig. 2, four ASIPs are organized in 2 component decoders respectively

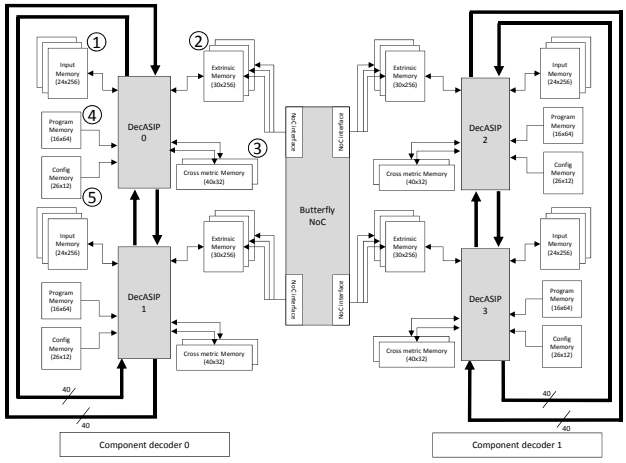

Figure 2. UDec system architecture example with 2x2 ASIPs

built with 2 ASIPs. Within each component decoder the ASIPs are connected by two 80-bit buses for boundary state metrics exchange. The DecASIP implements the Max-Log MAP algorithm as described in [11]. It supports both single and double binary convolutional turbo codes and implements radix-4 trellis compression technique for SBTC mode. Large frames are processed by dividing the frame into N windows each with a maximum size of 64 symbols. Each ASIP can manage a maximum of 12 windows. The DecASIP is associated with 3 memory banks of size 24x256 used to store the input channel LLR values ①. There are also another 3 banks of size 30x256 used for extrinsic information storing ②. Each ASIP is further equipped with two 40x32 memories which hold state values ③. Moreover, each ASIP is configured through a program ④ and a configuration memory ⑤. The configuration memory contains all parameters required to perform the initialization of the ASIP while the program memory contains the instructions in order to perform the decoding algorithm. Since the DecASIP is designed to work in a multi-ASIP architecture as described in [2], it requires several parameters to deal with a subblock of the data 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. All these parameters are required for a configuration of an ASIP within the platform.

In [2], authors show that the DecASIP architecture provides high performance and high flexibility. However the topic of dynamic reconfiguration is not addressed. Despite its high flexibility, it presents some lacks to offer an efficient dynamic reconfiguration. The next section points out theses lacks and proposes several solutions to implement an efficient reconfigurable DecASIP for the UDec turbo decoder architecture.

III. Reconfigurable DecASIP

Several optimizations are proposed to reach an efficient dynamic reconfiguration of the DecASIP architecture. The first optimization is related to the storage of configuration parameters. Currently, some parameters are stored in the configuration memory and others are provided in the program instructions directly [2]. The second optimization deals with the way used to load the configuration memory through the configuration memory organization. The third optimization corresponds to the development of a generic program independent of the configuration to be performed.

A. Configuration parameters storage

To reach configuration efficiency, we propose to move all parameters from the program memory to the configuration memory. This solution allows to configure a single memory to change all the configuration parameters (instead of loading both new program memory and configuration memory). Furthermore, once the ASIP is configured, the configuration memory can be accessed without any conflict since the configuration is loaded inside internal registers of the ASIP during the initialization step. This is a key point to prepare the next configuration if necessary. Indeed, the entire next configuration can be loaded in the configuration memory during the processing of the current data frame. Thus the configuration loading can be completely masked.

B. Configuration memory organization

In order to improve the configuration of the ASIP, it is essential to analyze the organization of the configuration memory. The parameters stored in the configuration memory are very specific. They can be divided in four categories: 1) interleaving 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 a single binary turbo code mode. All these characteristics need to be taken into account in order to build a low latency configuration process. A smart memory organization should allow an efficient broadcasting of the configuration parameters to the required ASIPs. Thus we propose to group the parameters depending on the previously described categories. Three groups which occupy different parts of the configuration memory are defined.

Table I shows the proposed configuration memory organization. The memory is organized as follows: (1) from address @0 to @1, parameters can be different for each ASIP. Furthermore, to optimize the initialization step of the ASIP, the parameter *Tail* which indicates if the ASIP has to perform or not the tail bits is also included in this group. Only the last two ASIPs are concerned by the tail bits in a single binary turbo code mode; (2) from address @2 to @6, the parameters are domain dependent; (3) from address @7 to @10, the parameters 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 order to minimize the data transfers load. In this context, two multicast transfers are necessary to send domain dependent parameters to dedicated 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

We propose to simplify the configuration mechanism by using a unique generic program. Since all parameters contained in the program memory have been moved to the configuration memory, three possibilities exist for the program: 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, data frames are decoded after the initialization step. In order to merge these three possible programs, the new unique program has to be able to tackle these three cases. For this purpose, the program which integrates the tail bits computation is used as a reference. We have chosen to modify the *Fetch* pipe stage of the ASIP in order to detect and replace the instructions for tail bits with *NOP* instructions if the ASIP is not concerned. The value of the bit *Tail* stored in the configuration memory (Table I) determines if the ASIP is concerned or not by tail bits decoding. In duo binary mode, no tail bits have to be decoded. So, using a unique program in this mode adds 12 extra *NOP* instructions before the decoding step which corresponds to tail bits computation in single binary mode. However, these extra clock cycles are negligible regarding the number of cycles required to perform the decoding on one entire data frame.

Optimizations described in this section allow to reduce the (re)configuration impact thanks to the new memory organization and the generic program which reduce the total configuration load to be transferred when a new configuration has to be performed.

IV. Main challenges for an efficient configuration infrastructure

In the UDec architecture, the 80-bit buses and the Butterfly NoC are optimized and dedicated to exchange data between DecASIPs. So, these interconnection structures can not be used to transfer the configuration data without performance lost. To build an efficient solution, the configuration

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
NEW CONFIGURATION MEMORY

interconnection structure has to be specific and to take into account the following requirements: 1) Low complexity, 2) Multicasting mechanism and 3) Burst transfer.

A. Low complexity

The configuration infrastructure only manages configuration memories updates. Thus, this extra hardware must have a minimal impact on the global design complexity in terms of area overhead. When designing a communication architecture in a multiprocessor platform, two main technologies are available: Network on Chip or On-Chip Bus. Last decade has seen the huge adoption of Networks on Chip in complex System on Chip to mainly enhance the throughput and the scalability compared to a bus-based communication infrastructure. However, the design of a communication interconnect dedicated to configuration data does not require such a complex approach. Indeed, configuration broadcasting can be defined as a unidirectional communication between a configuration manager that generates and transfers configuration data to one or a group of processing elements that have to be configured. Hence, there is no transfer concurrency issue, and a unique component, called *Master*, is able to initiate a transfer. These features lead to a bus-based structure that provides a simple communication interconnect for this particular context.

B. Multicasting and selection

The UDec platform is configured through DecASIP configuration memories. As shown in section III, the DecASIP configuration memory is organized in order to allow multicast mechanisms for an efficient and fast configuration of the multi-ASIP platform. Moreover, depending on the application requirements, the number of activated DecASIPs to perform a given configuration can be tuned at runtime. Hence, a mechanism of processor selection has to be introduced in order to send configuration data to activated DecASIPs only.

C. Incremental data burst transfer

The last point to build an efficient configuration infrastructure for the UDec platform is related to the transfer mode. Since some of the configuration data has to be loaded in adjacent parts in the configuration memory, all related transfers can be defined as a burst starting from a *base address* in the configuration memory. For example, based on the DecASIP memory organization of Table I, configuration data identical for all DecASIPs can be incrementally transferred starting from the base address @7. Four incremental transfers are then performed.

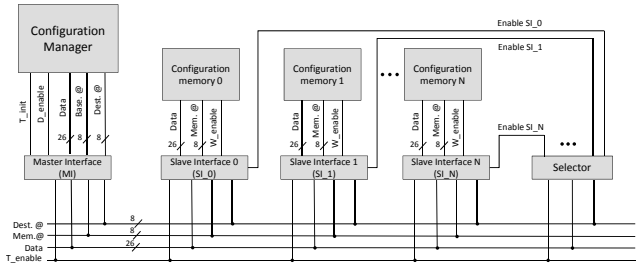

Figure 3. Architecture of the proposed bus interconnect

Many On-Chip Buses have been developed these last years that propose different topologies and different communication protocols. Representative On-chip Buses are the AMBA [12], the CoreConnect [13] or the Avalon bus [14]. Unfortunately, these solutions do not support multicast. [15] supports multicast but this solution implements complex arbitration mechanisms and communication protocols that are not necessary in our context. The Fast Simplex Link (FSL) [16] proposes a low complexity unidirectional bus for data transfer. Unfortunately multicast is not supported. It is thus required to propose an optimized bus dedicated to configuration data for the UDec platform.

V. CONFIGURATION INFRASTRUCTURE

To address the main issues highlighted in section IV, we propose a new bus-based communication infrastructure as well as the associated communication protocol. Our goal is to optimize configuration data transfers into DecASIPs configuration memories for the UDec platform. In this section, we detail the architecture, the dynamic selection of activated DecASIPs and the protocol.

A. Architecture overview

The proposed bus architecture is presented in Figure 3. This architecture can be split in four functional blocks: *Master Interface* (MI), *Slave Interface* (SI), *Selector* and *interconnect*. Each configuration memory is connected to the bus through a SI. The configuration manager deals with the configuration generation which is based on internal decisions

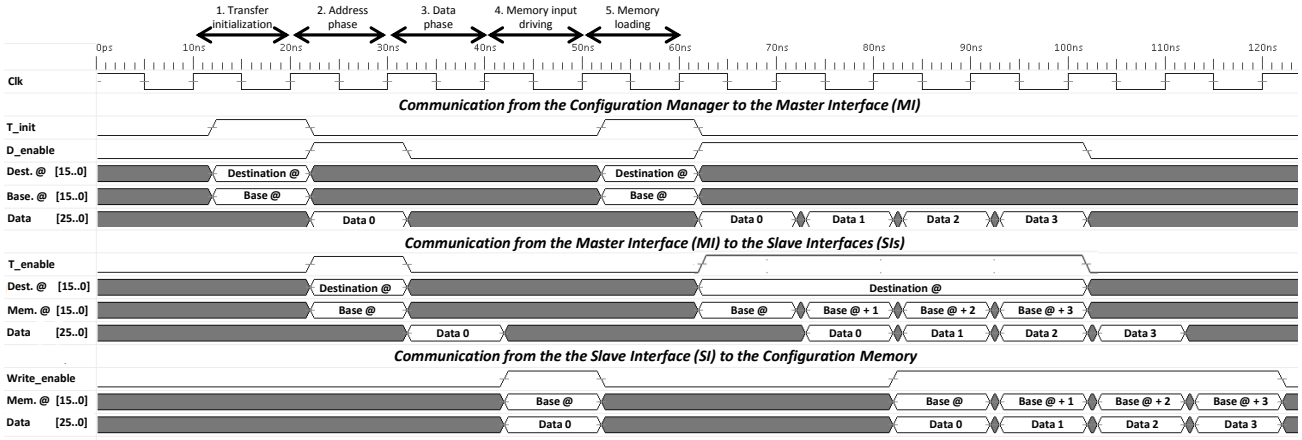

Figure 4. Communication from the configuration manager to the configuration memory through the communication infrastructure

and external information and commands (this point is not addressed in this paper).

The MI provides an interface allowing the connection of the configuration manager to the bus. To initiate a transfer, the MI receives, from the configuration manager, the address of a SI or a group of SIs (called *Destination address*) and the memory base address where the transfer starts. During a transfer, the MI also manages the increment of the memory address based on the base address.

The SI provides an interface between the bus and the configuration memory. Its role is, when a transfer is enabled, to check if the destination address corresponds to its own address or one of its associated multicast addresses. Then, the SI retrieves the data (and the associated memory address) from the bus and writes it into the configuration memory.

The Selector provides a simple and efficient solution to select, at run-time, DecASIPs that are targeted by the next configuration data. For this purpose, each SI has a 1-bit input that is driven by the Selector. When this input is enable, the associated SI is activated and reacts to the events on the bus while it ignores all transfers in the other case.

The interconnect part of the proposed architecture consists of three buses and a transfer enable control signal. Two address buses are required. The first one (*Dest.@* in Figure 3) is used to select the destination (i.e. one SI or a group of SIs) and the second (*Mem.@*) is used to indicate the target memory address. The third bus is used to send the configuration data. Finally, a control signal (*T_enable*) is used to inform SIs that a transfer has been enabled.

B. Transfer protocol

The transfer of configuration data can be divided in three steps: 1) initialization and data transfer from the configuration manager to the MI, 2) data transfer from the MI to one or several SIs and 3) configuration memory loading from the SI.

1) ***From the configuration manager to the MI (upper part of Figure 4)****:* During the initialization step, the configuration manager sends the destination address and the base memory address to the MI. The *T_init* control signal (Figure 3) is driven to indicate to the MI that a transfer initialization is required. On the MI side, when these two addresses are read, the first one is stored and the second one is used to initialize the memory address increment process. These addresses are used until a new transfer initialization step is performed. After the initialization step, the configuration manager can send one data per cycle on the *Data* bus. The *D_enable* control signal is also driven at the same time to inform the MI that a data is available. Obviously, the data transfer can be suspended if no data is available. Figure 4 shows an example of transfer initialization and data transfer between the configuration manager and the MI.

2) ***From the MI to the SI(s) (middle part of Figure 4)****:* Figure 4 presents two examples of data transfer on the bus. The first one shows the transfer of a single data, and the second shows a data burst. The transfer on the bus consists of two phases: address phase and data phase. The address phase lasts for a single clock cycle. During this cycle, the destination and the base memory addresses are sent on the corresponding bus. The *T_enable* control signal is also driven to indicate that a transfer occurs. During the data phase, the data is sent on the *Data* interconnect. When a data burst is performed, a data is available at each clock cycle. The destination address is maintained on the bus during the transfer procedure while, for each data, the memory address is incremented by the MI.

3) ***From the SI to the configuration memory (lower part of Figure 4)****:* When a transfer occurs, the SIs involved in the transfer store the memory address (read during the address phase) and get the data on the next clock cycle. To write into the configuration memory, the memory address is stored during one clock cycle. When the data is available, the control signal *write_enable* of the memory is driven and the memory address and the data are sent on the interconnect between the SI and the configuration memory.

These three steps allow the transfer of a data into the

configuration memory in 5 clock cycles. Moreover, thanks to the pipeline nature of the transfers, the configuration infrastructure is able to provide one data per clock cycle to the destination. As will be demonstrated in section VI such a solution outperforms existing solutions and allows reaching a very low latency reconfiguration time.

C. Selection

The UDec platform can dynamically select the number of DecASIPs involved in the decoding process depending on the requirements of an application (e.g. throughput, error rate, etc.). When a configuration command occurs, a selection mechanism is launched to select the SIs associated to the configuration memories connected to the DecASIPs involved in the next configuration. When an SI is not selected, it ignores all transfers on the bus. The Selector is configured through the bus infrastructure by the configuration manager which sends a configuration vector on the bus which is forwarded to the SIs.

This section has detailed the configuration infrastructure highlighting main features and providing an in-depth analysis of the latency. Next section focus on the implementation of the proposed solutions.

VI. Implementation Results

A. RDecASIP implementation

The proposed optimizations described in section III have been implemented on the DecASIP presented in [2]. The ASIP was modeled in LISA language using Synopsys (ex. Coware) Processor Designer tool. 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 area of the ASIP.

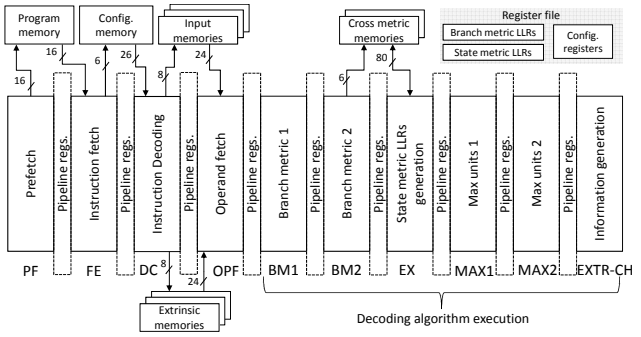

Figure 5. DecASIP Pipeline

To evaluate the impact of the new features on the ASIP area, we extracted the area synthesis results for each pipeline stage of the RDecASIP. This ASIP consists of 10 pipeline stages as shown in Fig. 5. The reconfiguration optimizations presented in this paper do not affect all the stages. Only three stages are impacted. Indeed, from *BM1* to *EXTR-CH*, stages are dedicated to data computation and this part of the ASIP pipeline is not directly concerned by the configuration optimizations proposed in this work. Moreover, The prefetch stage is identical in the two implementation of the DecASIP. On the other hand, fetch (*FE*), decode (*DC*), and operand fetch (*OPF*) stages have seen their area increased compared to the previous ASIP. The proposed optimizations were implemented along the pipelines stages as follows:

- FE: The FE stage insures the automatic replacement of instructions for tail bits computation by *NOP* when the ASIP is not concerned by tail bits decoding.
- DC: This stage is mainly impacted by the transfer of all flexible parameters in a unique configuration memory. Instead of a direct access to some parameters in instruction code words, parameters are now read from registers. Thus, the number of connections with the register file has been increased.
- OPF: This stage is impacted by the new configuration memory organization since it is in charge of the parameter registers initialization. The area overhead comes from the increasing number of parameters in the configuration memory and by added control structures that manage the configuration size flexibility. Since more configuration parameters are read from the configuration memory, the number of connections with the register file has been increased to configure additional registers.

	ASIP area (in μm^2
DecASIP	183,010
RDecASIP	184,638
Diff.	1,628 (+0.9%)

Table II
ASIP AREA COMPARISON IN μm^2

	Config param.	Prog. mem.	1 ASIP	n ASIPs
RDecASIP	286	-	286	n.52+260+104
DecASIP	336	640	976	n.976
Gain	14%	100%	70%	90% ($n = 8$)

Table III
CONFIGURATION AND PROGRAM BIT LOAD COMPARISON IN BITS

Table II shows the global area comparison between the DecASIP and the new version optimized for dynamic reconfiguration called RDecASIP. We observe that the global logic overhead on the ASIP is 0.9% (1,628 μm^2). This overhead is mainly due to the additional internal registers used to store the configuration parameters read from the configuration memory.

The RDecASIP is configured through the optimized configuration memory presented in section III. Table III compares the configuration and program load (in bits) for the proposed RDecASIP and the original DecASIP presented in [2]. For one ASIP, we observe that the proposed RDecASIP can be configured with 286 bits instead of 976 bits thanks to the generic program described in section III-C. Moreover, the new memory organization proposed in section III allows the optimization of the configuration memory loading.

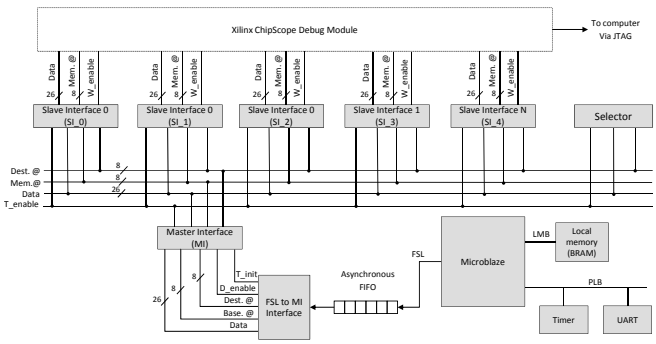

Figure 6. Architecture of the prototype

Indeed, parameters are sent to several ASIPs through a multicast mechanism. Thus, in a multi-ASIP context, each DecASIP has to be configured with its own configuration and program memory while configuration memory of the proposed RDecASIP can be loaded using a multicast mechanism as follows: 52 bits are independently loaded in each ASIP. ASIPs that work in the same interleaving domain are loaded with 130 common bits. Finally, 104 configuration bits are broadcasted to all ASIPs. Thanks to this new configuration memory organization, 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 to configure the 8 ASIPs is 7808 bits with the DecASIP and 780 bits with the proposed RDecASIP.

The timing performances for turbo decoding of the new and the original version of the DecASIP are identical. Indeed, the pipeline architecture is still the same and both implementation are able to reach the maximum frequency of 500MHz. However, in double binary mode, the RDecASIP requires 8 extra clock cycles after the initialization phase to start decoding. Nevertheless, we can disregard these impacts in front of the decoding time of a data frame that is identical for both versions of the ASIP.

B. Configuration infrastructure implementation

To validate the proposed bus architecture and communication protocol presented in section V an hardware prototype on a Xilinx XUPV5 platform based on a Virtex 5 LX110T FPGA was developed. The prototype architecture is shown in Figure 6. It consists of a Xilinx Microblaze soft core that generates the configuration at run-time. The configuration is then sent through an FSL bus to a FSL to MI interface. The Fast Simplex Link (FSL) [16] connection has been considered as this interconnect structure proposes a fast, simple and unidirectional connection. An asynchronous FIFO is associated to the FSL connection in order to provide frequency domain flexibility on both Microblaze and configuration infrastructure sides. The FSL to MI interface realizes the protocol adaptation between the FSL communication protocol and our bus protocol. Finally, the outputs of each SI are connected to a Xilinx ChipScope module that allows the run-time monitoring of these signals. This module replaces the configuration memories associated with the SIs.

Nb. ASIPs	Transfer latency (in ns)			Speedup	
	This work	[13]	[12]	vs. [13]	vs. [12]
4	1 032	3 872	2 212	3.75	2.14
6	1 176	5 808	3 168	4.94	2.69
8	1 320	7 744	4 224	5.87	3.2
16	1 896	15 488	8 448	8.17	4.45
32	3 048	30 976	16 896	10.16	5.54
64	5 352	61 952	33 792	11.57	6.31

Table IV
CONFIGURATION TRANSFER TIME IN ns

Thanks to this hardware implementation, configuration transfer time were evaluated for several number of RDecASIP. For this purpose, the Microblaze and proposed bus frequency is set to 125 MHz. The Chipscope module is configured to monitor the output signals of the SIs. Table IV shows the configuration transfer times of the proposed bus compared with designs implementing CoreConnect PLB4 [13] and AMBA AXI4 [12] buses connected to a Microblaze with the clock frequency set up to 125MHz. Thanks to the multicast mechanisms, a low overhead of 72 ns is necessary to configure each additional couple of RDecASIPs (one ASIP in both natural and interleaved domains) while 968 ns and 528 ns are necessary for [13] and [12] respectively. Results of Table IV show that the proposed implementation significantly reduces the configuration time overhead when the number of active RDecASIPs increases compared to classical bus approaches.

Infrastructure Component	Area (in μm^2)
MI	1 790
SI	1 150
Selector	784
Infrastructure for 8 RdecASIPs	15 199
8 RDecASIP	1 477 104

Table V
AREA OF THE PROPOSED CONFIGURATION ARCHITECTURE

A logical synthesis of the proposed bus components was also done with 65nm CMOS technology with a clock frequency objective equals to 500MHz. Table V shows the area evaluation for the three components of the proposed configuration infrastructure. The logic overhead caused by the configuration infrastructure is 0.015 mm^2 which leads to a low penalty of 1% regarding the logic area of the 8 RDecASIPs (1.477 mm^2). Furthermore, regarding frequency objective of 500 MHz, a speedup of 4 on the configuration transfer latencies shown in Table IV can be expected compared to the 125 MHz FPGA prototype.

C. Stopping-free reconfiguration analysis

The minimum frame decoding duration providing a stopping-free configuration is determined by the longest configuration time that is 5.352 μs in Table IV. Thus, the maximum achievable throughput is theoretically limited for a given frame size and is given by the equation (1) where $Frame\ duration_{min}$ is equal to 5.352 μs.

Frame size (bits)	T_{max} (Mbps)	N_{ASIP}	Frame size (bits)	T_{max} (Mbps)	N_{ASIP}
96	17	2	1920	358	36
480	89	10	4800	666	64
880	164	16	6144	666	64

Table VI
ESTIMATED MAXIMUM THROUGHPUT FOR A STOPPING-FREE DYNAMIC CONFIGURATION OF THE UDEC PLATFORM

$$T_{max}\ (in\ bps) \quad = \quad \frac{Frame\ size\ (in\ bits)}{Frame\ duration_{min}\ (in\ s)} \qquad (1)$$

Considering the UDec platform implementing RDecASIPs, the maximum achievable throughput is limited by the number and the performance of the RDecASIP. Equation (2) shows the estimated throughput of the UDec platform.

$$T \quad = \quad \frac{F_{clk}.(N_{ASIP}/2)}{N_{instr}.N_{iter}} \qquad (2)$$

Where N_{ASIP} is the number of RDecASIPs, N_{instr} is the average number of instructions to decode one symbol (N_{instr} = 4 for the RDecASIP), N_{iter} is the number of iterations performed and F_{clk} is the frequency. It is worth to note that increasing the subblock parallelism degree can be done with careful consideration of boundary state metrics initialization as it can impact the convergence speed and thus reduce the parallelism efficiency [17]. Several parameters should be considered in terms of frame size, code rate, interleaving rules, and state metrics initialization methods [17]. Considering equation (1) and equation (2) where the number of RDecASIP is limited to 64, Table VI, shows the maximum estimated throughput achievable in SBTC and DBTC allowing a stopping-free dynamic configuration and the corresponding number of RDecASIP that have to be used for different frame sizes and a number of decoding iterations equals to 6.

Results of Table VI show that stopping-free dynamic configuration can be achieved for a throughput up to 666 Mbps by coupling the RDecASIP presented in section III and the configuration infrastructure proposed in section V. For a UDec architecture implementing 64 RDecASIPs, Table VI shows that for long frames the throughput is limited by the number of implemented RDecASIPs while the throughput for smaller frames is bound by the configuration latency. However, considering an ASIC implementation of the configuration infrastructure, this bound can be divided by 4.

Compared to the recent related work proposed in [9],the configuration infrastructure consists of several buses, each connected to a group of 4 PEs. Up to 8 buses have been implemented to configure 35 PEs to provide throughput up to 292 Mbps in DBTC mode and 150 Mbps in SBTC mode. However, the way the buses are driven is not described in details in the paper but the management of the 8 buses in parallel should increase the complexity of the *configuration manager* used to load new configurations. The approach proposed in this paper provides optimizations at PE level in order to build an efficient and low complexity configuration infrastructure. The proposed solution is a single bus implementing mechanisms optimized for dynamic configuration management that leads to stopping-free decoding when the constraints previously analyzed are respected. Such an approach allows an efficient dynamic configuration of high speed decoders without loss in error correction quality.

VII. CONCLUSION

This work describes an ASIP-based reconfigurable architecture for turbo decoding. This architecture is based on the UDec architecture implementing the RDecASIP optimized for an efficient dynamic configuration in a multiprocessor context. The architecture is reconfigured through a novel bus-based configuration infrastructure implementing incremental burst, unicasting, multicasting and broadcasting mechanisms providing a high speed configuration data transfer. Dynamic configuration in one frame decoding duration is provided for high throughput up to 666 Mbps when the frequency of the configuration infrastructure is 125 MHz.

REFERENCES

[1] T. Vogt, C. Neeb, and N. Wehn, "A Reconfigurable Multi-Processor Platform for Convolutional and Turbo Decoding," in *Proc. of the International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2006, pp. 16–23.

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

[3] C. Condo, M. Martina, and G. Masera, "A Network-on-Chip-based turbo/LDPC decoder architecture," in *Proc. of the Design, Automation and Test in Europe Conference & Exhibition (DATE)*, 2012.

[4] M. Martina, M. Nicola, and G. Masera, "A Flexible UMTS-WiMax Turbo Decoder Architecture," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 55, no. 4, pp. 369–373, April 2008.

[5] J.-M. Hsu and C.-L. Wang, "A parallel decoding scheme for turbo codes," in *Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS)*, 1998, pp. 445–448.

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

[7] T. Vogt and N. Wehn, "A Reconfigurable ASIP for Convolutional and Turbo Decoding in an SDR Environment," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 10, pp. 1309–1320, oct. 2008.

[8] "IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless," *IEEE Std 802.16e-2005*, 2006.

[9] C. Condo, M. Martina, and G. Masera, "VLSI Implementation of a Multi-Mode Turbo/LDPC Decoder Architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers, Early Access Articles*, 2012.

[10] H. Moussa, A. Baghdadi, and M. Jezequel, "Binary de Bruijn on-chip network for a flexible multiprocessor LDPC decoder," in *Proc. of the Design Automation Conference (DAC)*, 2008, pp. 429–434.

[11] 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.

[12] ARM, AMBA specifications v2.0. ARM. [Online]. Available: http://www.arm.com.

[13] IBM, CoreConnect Bus Architecture. IBM Microelectronics. [Online]. Available: http://www.ibm.com/chips/products/coreconnect.

[14] Altera, Avalon bus specification: Reference manual. Altera Corporation. [Online]. Available: http://www.altera.com.

[15] Sonics network technical overview. Sonics, Inc. [Online]. Available: http://www.sonicsinc.com.

[16] Xilinx, FSL V2.0 specification. Xilinx, Inc. [Online]. Available: http://www.xilinx.com.

[17] O. Muller, A. Baghdadi, and M. Jezequel, "Parallelism Efficiency in Convolutional Turbo Decoding," *EURASIP Journal on Advances in Signal Processing*, 2010. [Online]. Available: http://asp.eurasipjournals.com/content/2010/1/927920