# Towards Future Adaptive Multiprocessor Systems-On-Chip: an Innovative Approach for Flexible Architectures

Fabrice Lemonnier*, Philippe Millet*, Gabriel Marchesan Almeida†, Michael Hübner‡, Jürgen Becker†, Sébastien Pillement§, Olivier Sentieys§, Martijn Koedam¶, Shubhendu Sinha¶, Kees Goossens¶, Christian Piguet||, Marc-Nicolas Morgan||, Romain Lemaire**

*Thales Research & Technology, 1 Avenue Augustin Fresnel, 91767 Palaiseau cedex

†Institute of Information Processing Technology, Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany

‡Embedded Systems for Information Technology, Ruhr-University Bochum, 44801 Bochum, Germany

§University of Rennes 1 - IRISA, 6 rue de Kerampont, 22300 Lannion, France

¶Eindhoven University of Technology, Eindhoven, The Netherlands

||Centre Suisse d'électronique et de microtechnique SA (CSEM), 1 rue Jaquet-Droz 2002 Neuchatel, Switzerland

**CEA-Leti, Minatec Campus, Grenoble, France

***Abstract*—This paper introduces adaptive techniques targeted for heterogeneous manycore architectures and introduces the FlexTiles platform, which consists of general purpose processors with some dedicated accelerators. The different components are based on low power DSP cores and an eFPGA on which dedicated IPs can be dynamically configured at run-time. These features enable a breakthrough in term of computing performance while improving the on-line adaptive capabilities brought from smart heuristics. Thus, we propose a virtualisation layer which provides a higher abstraction level to mask the underlying heterogeneity present in such architectures. Given the large variety of possible use cases that these platforms must support and the resulting workload variability, offline approaches are no longer sufficient because they do not allow coping with time changing workloads. The upcoming generation of applications include smart cameras, drones, and cognitive radio. In order to facilitate the architecture adaptation under different scenarios, we propose a programming model that considers both static and dynamic behaviors. This is associated with self adaptive strategies endowed by an operating system kernel that provides a set of functions that guarantee quality of service (QoS) by implementing runtime adaptive policies. Dynamic adaptation will be mainly used to reduce both overall power consumption and temperature and to ease the problem of decreasing yield and reliability that results from submicron CMOS scales.**

## I. Introduction

The increasing amount of power-hungry applications appearing in actual products such as mobile telephones and tablets are pushing industry to rapidly develop very complex and robust architectures capable of dealing with both power consumption and performance constraints. On the one hand, these platforms must provide high performance without using excessive amounts of hardware to do so. On the other hand, they have a very limited power budget which has to be carefully allocated in order to avoid compromising the whole design.

Multi-core architectures appear as a promising solution to deal with all these scenarios and limitations raised by complex applications. Industry is aware of the need of using multi-core and shortly manycore chips to raise the performance/power ratio especially in embedded systems when power consumption is one of the main constraint. When a technological breakthrough is needed, particularly to satisfy the applications requirements, the R&D teams design their own dedicated accelerators using data parallelization (e.g. SIMD) or instruction parallelization (e.g. VLIW). Nevertheless, industry is reluctant to take the plunge into the manycore world. Among the many understandable reasons for such behavior, the impossibility of reusing most legacy code (which is mainly sequential C code), coupled with the the risk of using an unsustainable solution and the increase in development cost are strong factors that make industry being more conservative. Additionally, there is also a problem of accessibility for small product volumes where it is not profitable to design a custom heterogeneous manycore architecture.

It is a fact that today's heterogeneous multi-core architectures present better performance and are more efficient in terms of power consumption, which makes them the de-facto standard in embedded systems [1]. Usually they are domain oriented architectures that allow faster execution at low operating frequencies, resulting in low power consumption solutions. However, due to programmability issues, the complexity of building such architectures is very high. In the FlexTiles project[2], we are convinced that embedded FPGA (eFPGA) with hardware providing also fixed processor cores, is required for being included in future SoC designs in order to fulfill the requirements of embedded high performance applications of the future.

The work presented in this paper relies on a self-adaptive heterogeneous manycore architecture based on flexible tiles called FlexTiles and it is organized as follows. Section II summarizes the state of the art while Section III introduces the proposed architecture. Section IV describes the virtualization layer which is mainly responsible for information management and decision making, both at run-time. Finally, conclusions are drawn in Section V.

## II. State of the Art

The heterogeneous manycores combine parallelisation with customization in order to raise the efficiency in terms of GOPS per Watt. In the case of the mobile market with huge volumes, the companies are able to develop heterogeneous multicore like the OMAP family. The manycore with several hundreds of cores is already difficult to program although in most of the cases, a dedicated language is defined to ease the programmation. TILE-Gx family$^{TM}$ from Tilera[3] dedicated to audio and video processing and advanced networking embeds up to 100 cores and programmable in C, C++ languages with a SMP execution model. We can not avoid to speak about the GPGPU like the Fermi processor from NVIDIA based on 512 cores for graphics applications and programmable with CUDA. For its future 256 cores MPPA$^{TM}$ chip, Kalray [4] has devoted an important effort on the tools chain to ensure the programmability. The heterogeneity will not ease this issue. STM proposes a new manycore - P2012 [5] - which is based on cluster in which we can find general purpose processors and dedicated hardware processing elements. For the moment, they have developed a homogeneous version and they try to solve the programmability through different projects like SMECY[6]. In the FlexTiles project, we will define the programmability model at the same time as the architecture. The heterogeneity in the FlexTiles architecture is provided by very low power DSP of CSEM and by dedicated IPs on eFPGA. The issue of heterogeneity programmability is recurrent in the FPGA domain. In the FOSFOR project[7], a common multi-thread execution model is proposed with the corresponding services whatever the thread is executed on a general purpose processor or in a reconfigurable zone as a dedicated IP[8]. In the project FREIA[9], a common interface has been defined to control any type of accelerators in a master-slave execution model.

## III. FlexTiles Architecture

### A. FlexTiles Platform

The whole solution developed in the FlexTiles project targets high-performance and yet dynamic algorithms in embedded products. The programmer's view is a set of concurrent threads with different priorities where the parallelism is expressed in a way that allows dynamic resource allocation for each of them at run-time. In addition, each thread can address domain-specific accelerators to meet the real-time or HPC constraints of the application.

The description of the application is captured thanks to a tool chain that helps application programmers implementing and deploying their applications on the targeted architecture.

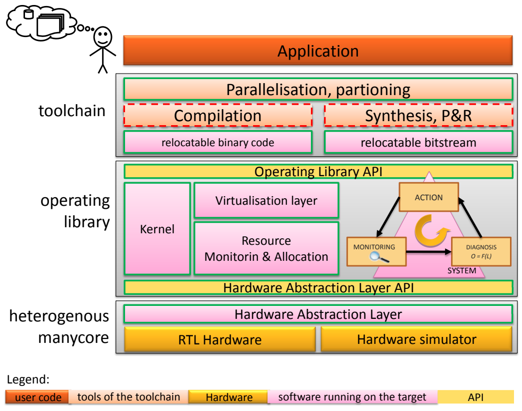

Fig. 1. FlexTiles Platform

With this tool chain, designers will be able to easily express the parallelism of applications by describing application threads as series of dataflow graphs and combining them with priority orders and synchronization mechanisms. That makes the application partitioning - i.e. which part goes to which accelerator - easier. The main application will be programmed in C or C++ code. The accelerated parts of the applications are supposed to be written in the language of the targeted accelerator: for DSP they are programmed in C code (possibly using assembly-level optimized libraries), for eFPGA they are written in VHDL code or in C code converted through High-level synthesis (HLS) tools. An operating library will embed a system monitoring to be able to allocate the resources and load balance the work over the processors according to several sensors inside the chip. Each part of this platform will be presented in the following sections.

### B. Global architecture

The FlexTiles architecture will use three main innovations from the hardware implementation point of view. i) an innovative embedded FPGA supporting placement free bitstream. This property is important for supporting task migration or at least dynamic placement of task in the matrix. ii) a very-low power DSP architecture. This block will enable energy efficient implementation of signal processing application. and iii) the use of 3D stacking implementation. This technology will enable resources sharing among the different layer of the architecture. For communications purpose a flexible NoC supporting different Quality of Services (QoS) will be used as communication infrastructure.

The architecture is seen as a set of nodes based on GPPs, DSPs or dedicated accelerators on eFPGA, I/Os and DDR access as represented in the figure 2. These are the basic stones of the architecture. The GPPs are used as masters while the DSPs, eFPGA and I/Os are used as slaves and the DDR as passive. The NoC domain represented on this figure may be physically implemented with several NoC interconnects to match different QoS requirements as explained in the section III-E.

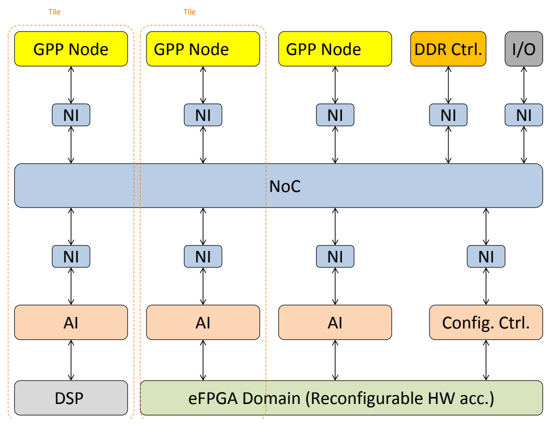

Fig. 2. Global view of the hardware architecture of the FlexTiles heterogeneous manycore. The eFPGA is stacked over an manycore SoC. The manycore layer is build using a NoC and mixing GPP and DSP block.

A Network Interface (NI) connects each node to the NoC. An Accelerator Interface (AI) is defined to implement the master-slave execution model between GPP nodes and accelerators nodes.

The eFPGA is on a dedicated layer in a 3D-stacked chip with a manycore layer (see figure 3). The two layers are linked together through microbumps (Copper Pillars [10]). This approach avoids the classical limitation of area when eFPGA is embedded inside the same silicon layer. Moreover, it simplifies the migration of soft IPs which is necessary to implement the dynamic re-allocation.

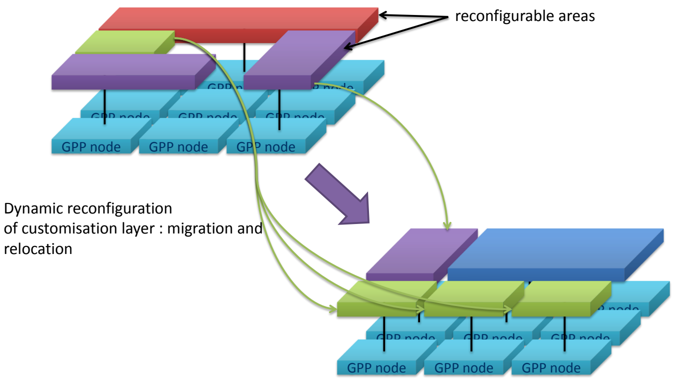
reconfigurable areas
GPP node
GPP node
GPP node
Dynamic reconfiguration of customisation layer : migration and relocation
GPP node
GPP node
GPP node

Fig. 3. Reconfigurable layer

## C. eFPGA

In FlexTiles, the reconfigurable resources are seen as a homogeneous set of resources that can be allocated at runtime and on demand by a processor. This leads to a better resources sharing among the manycore SoC, and thus enable the implementation of large accelerators if required. The increase in resource usage efficiency will also enable advanced management strategies such as power-saving for example.

The main innovative point on the eFPGA, linked with the previous features, is the definition of a real re-locatable bitstreams. The tasks will therefore become independent of their placements in the eFPGA fabric and the bitstream storage requirements will be reduced since one unique bitstream is stored for multiple positions. This ability has been defined as "virtual bitstream". Finally, as we aim at supporting dynamicity and advanced management of tasks (like relocation at runtime) the eFPGA will require a very efficient reconfiguration process.

In this project we envision a classical basis for the internal structure of the eFPGA, designed as a matrix of Logic Blocks (LB) interconnected by a regular mesh-based configurable interconnection network. The LB will include fine-grain logic functions (4- or 6-input LUT and a Flip-flop). Memory blocks for local data storage will also be included. We may also include some arithmetic computing resources (arithmetic configurable datapaths) to improve the overall processing performance, if necessary.

However, the eFPGA proposed in FlexTiles will also have very specific features:

- As the stacked eFPGA layer will not have direct access to the I/O pads of the chip, the input/output cells of a classical FPGA are replaced by 3D Network Interfaces (3DNIs) that give access to the manycore layer. This will have a big impact on the fabric structure since no I/O cells will surround the fabric. The matrix will be accessible via 3DNIs that will contain the necessary logic (potentially different clock and power domains) to interface though microbumps with the FlexTiles NoC.
- The configuration memory will be a two-context configuration memory, i.e. one new context could be downloaded while the second one is still active [11]. The area overhead of the second context will be minimized and the context switch will be as fast as possible.
- The configuration bits will be input as a serial stream, into a configurable scanpath. This is mainly to reduce the area overhead of the configuration plan.
- Finally, the reconfiguration controller will be designed specifically so that it can manage the previous structures and take benefit of the "virtual bitstream".
- We also may include an on-chip reconfiguration RAM for bitstream "caching". The access to the external reconfiguration memory containing task virtual bitstreams will be a bottleneck. Therefore, an on-chip reconfiguration memory will be implemented to optimize access to the virtual bitstreams.

The above-mentioned features have several impacts on the definition of the architecture, and they will be taken into account during the design phase of the architecture.

The main constraint in this architecture comes from the bitstream relocation ability that we want to support. In that sense, the architecture has to be very symmetric, i.e. a repetitive pattern needs to be found in the resource implementation so that moving the configured logic inside the fabric will not require a new complete place and route. The more symmetry the matrix will contain, the more flexible and easy to manage will be the architecture. One of the problems is then to decide where to put all heterogeneous blocks. Asymmetric blocks are memories, 3DNIs, and eventually arithmetic operators. The second impact of the relocation of tasks concerns the routing resources. It will be necessary, depending on the final placement of a bitstream in the matrix, to enable access to 3DNIs and memories. Three strategies need to be studied and evaluated:

1) Bus-based: using dedicated routing resources with a bus-based architecture to access to asymmetric resources.
2) NoC-based: implementation of a NoC structure to access to asymmetric resources
3) Dynamic routing: compute local routing to create the access to the particular resources.

### *D. DSP*

DSP processors can be used as accelerators alongside a microprocessor. There are two types of DSP processors: the Mephisto DSP [12] designed by CEA for baseband processing in the context of software-defined radio (SDR) applications and the CSEM icyflex4 which is more dedicated to video and audio applications.

The Mephisto DSP core is based on a 32-bit data path VLIW structure organized around a MAC dedicated to complex arithmetic but able to perform scalar operations as well, and two dedicated operators, a cordic/divider block and a compare/select one. It implements FIFO-based interfaces well-suited for dataflow streaming processing. Internally, data can be fed directly from two RAM banks (1Kx32b each) and a register array. The control part extracts from the 1Kx64b instruction memory the compacted instructions needed to feed the data path and the five address generators. The whole architecture presents a deep pipeline of 10 stages in order to achieve 400 MHz worst-case frequency in 65nm. The use of a reconfigurable profile and instruction cache strategy leads to a 10x reduction of the control power consumption. As a result, an average 50 mW power consumption is measured after implementation in a low-power 65 nm technology while delivering 3.2 GOPS.

The icyflex4 DSP core has been optimized to operate on 16- and 32-bit real or complex data types and makes it ideal for demanding signal processing applications such as audio or video as well as in digital communication systems.

An operation bundling mechanism enables the compaction of up to three independent operations in a single 64-bit instruction, executed either in sequence or in parallel (or a combination). It is built around a 5- to 8-stage pipeline. It contains a scalable vector processing unit (VPU) organized in “slices” containing registers and data processing elements which are used by single instruction multiple data (SIMD) vector operations. At synthesis time, one can select from 0 (scalar unit only) to 8 vector processing slices (VPS), the width of the memory busses being simultaneously scaled between 64 bits and 512 bits to feed the VPU slices with sufficient data.

The datapath of a single slice contains sixteen 64-bit registers (accessible as independent 16- or 32-bit registers) and eight 64-bit accumulators. A quad 16-bit multiplier and a recombination unit can perform a complex 16x16 bit multiplication or a scalar 32x32 bit multiplication at each cycle. The VPU slice additionally contains a 64-bit arithmetic and logic unit (ALU), a 64-bit barrel shifter and a 64-bit move unit, each able to compute up to four 16-bit operations per cycle.

The data move unit (DMU) contains two address units which drive two dedicated data busses. These two addressing units each support extended addressing modes (e.g. reverse-carry addressing typically used in FFT computation, or modulo addressing with start and end indexes) in addition to the more standard indexed modes with pre- or post-modification, used by a C compiler.

The DMU and VPU are reconfigurable at run time by means of the preconfigured addressing modes and the micro-operation (MOP) mechanism. The user is thus able to generate completely new complex instructions exploiting very efficiently all the computational units present in the VPU and in the DMU.

TABLE I
TYPICAL KEY NUMBERS FOR THE ICYFLEX4 DSP CORE (WITH TWO VPU SLICES) SYNTHESIZED IN FLIP-FLOPS IN TSMC 65 NM TECHNOLOGY, AT DIFFERENT SUPPLY VOLTAGES AND AT DIFFERENT FREQUENCIES

| Technology Node | TSMC 65 nm | TSMC 65 nm |
|---|---|---|
| Supply voltage | 0.9 V | 1.0 V |
| Frequency | 10 MHz | 100 MHz |
| Area | 0.54 mm$^2$ | 0.59 mm$^2$ |
| Power consumption | 36-125 $\mu$W/MHz | 44-163 $\mu$W/MHz |
| Max. Frequency | 75 MHz | 170 MHz |

### *E. Network on Chip*

Interconnection is a tremendous issue for using powerful accelerated functions capable of processing huge loads of data in real-time. The manycore of FlexTiles is based on computing nodes linked together via a network on chip. Since each data type has its own constraints a specific study in this project is focusing on the 3D-stacking technology that links the two layers and that must be able to transfer both control instructions and the data to be computed from the GPP nodes to dedicated accelerators.

We have identified the following types of functionalities that the NoC shall be able to manage with different kinds of quality of service:

- *Instructions*: the GPP, the DSP and the eFPGA (when a soft processor is implemented inside an eFPGA accelerator) need to get instructions from the network. This will imply transfers of chunks of memory with relatively low latency,
- *Data*: chunks of memory without a strong constraint on latency but requiring high bandwidth,
- *Bitstream*: only needed by the eFPGA, and only to one point on the network. Large chunks of memory with low latency,
- *Control*: this network is used to exchange status, mutex, synchros, semaphores, signals... These are very short

pieces of memory (typically registers) but with very low latency requirement and some level of predictability,

- *Test and debug*. there is neither constraint on latency nor on throughput.

Two NoCs implementation will be evaluated in the context of FlexTiles: AEthereal [13] and ANoC [14]. The Æthereal NoC offers guaranteed services (GS) such as uncorrupted, lossless, ordered data delivery, guaranteed throughput, and bounded latency. It uses TDMA, where the time is divided into slots that are globally synchronised. Application connections are set up or removed at run time [15], without affecting concurrently running applications. ANoC stands for Asynchronous network on chip (NoC) and has been developed by the CEA. The ANoC technology offers an high speed, low-latency, low-power and reliable communication solution. The router and links are implemented in fully asynchronous logic (clockless) that easily enables GALS architecture with independent clock and power domains for each resource interconnected by ANoC. ANoC implements wormhole-based packet-switching source-routing mechanisms. The router architecture is flexible and allows any kind of topology and communication model (request/response, streaming). Virtual or multiple physical channels can be added to bring different levels of QoS. In FlexTiles, These two NoC approaches will be considered depending on the different constraints (control, data and debug). We will have also to define the best way to interconnect the NoC with the reconfigurable area.

### *F. 3D-stacking integration*

For flexible heterogeneous manycore design, the use of 3D stacking can lead to very attractive optimizations. Within two dimensions, a reconfigurable (fine-grain or coarse-grain) accelerator has to be customized to one particular processor, which might lead (i) to a waste in resources, since some tasks of the application may not require acceleration and (ii) can jeopardize the acceleration since the reconfigurable resources are limited, the complexity of the accelerator being bounded by the reconfigurable resources physically implemented. In the context of a third dimension, if a global reconfiguration layer is stacked over a classical multi-core layer , then the accelerators can be allocated at runtime and on demand by different processors and the hardware resources can therefore be shared among different processors. This architecture will increase the resource usage and then decrease the implementation costs.

The challenges to address in case of a face-to-face staking approach are then the definition of efficient interfaces between the different layers:

- *Inter-die connection*: it will be implemented through microbumps and high level metal redistribution layer (RDL),
- *Package connection*: Through Silicon Vias (TSVs) are required to propagate I/O (including power pads) to the package.

As the eFPGA will be stacked over the manycore architecture some technological tips need to be addressed. The first one is about the number and the positioning of microbumps and TSVs. The designer has to respect design rules such as minimum pitch, which can lead to large area overhead. Moreover implementing a TSV has a significant area cost since it consumes silicon in the transistor active area. It is already clear that a 3DNI cannot be implemented in each router of the architecture. Furthermore, as 3D connection exhibit better timing performance than (long) wires but at a price of a cost overhead and lower density, serializer-deserializer (SERDES) blocks will certainly have to be implemented to limit the number of wires. The second important concerns is the definition of clock and power domains, since it is not relevant to consider that the manycore and the eFPGA will share the same frequencies and supply voltages. We then need to take this constraint into account for the implementation of the glue-logic required for the synchronization process.

Also, with current integration technologies, TSV sizes may vary from 1 $\mu$m to 100 $\mu$m. For example, the area of a 4-$\mu$m TSV is greater than the area of 500 SRAM cells in a 45 nm technology. It is therefore important to minimize the number of TSVs and their sizes to reduce manufacturing costs.

### *G. Programming Model*

An application is a set of static clusters. A cluster is described using Synchronous Data Flow (SDF) or Cyclo-Static Data Flow (CSDF) models of computation [16].

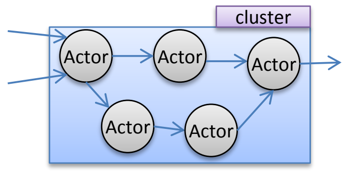

Fig. 4. Static cluster

Within a dataflow, each consumer/producer of tokens is called an actor. An actor is featured by nested loops implementing the operator and the rules of token consumption/production. Two actors communicate through FIFOs of tokens.
Each cluster can be started or stopped depending on events acting globally like a Discrete Event (DE) representation. This dynamicity is represented by groups of clusters. In each group, several clusters having the same data flow inputs and outputs are sensible to different events or event combination. At a given time, only one cluster of the group is active. Three possibilities are represented in the figure 5:

- Cluster group 4: This computing element has 3 possible behaviors.
- Clusters 1 and 2: The computing element is started when a dedicated event appears (state 2) else nothing is done (state 1).
- Cluster group 3: When there is no dynamicity, there is only one state.

- Cluster group 5 is able to dynamically duplicate. Depending on how the chip is loaded and how many nodes this cluster group is allowed to use, it can duplicate it-self to parallelize its processing. It is called data parallelization because the data are spread among the duplicated clusters. The designer decides the rules to parallelize and split the data depending on the number of instances of the cluster as well as the allowed possible numbers of instances.

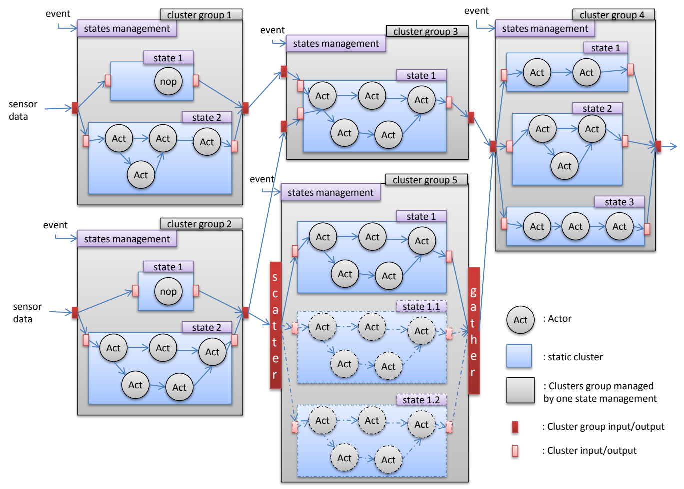

Fig. 5. Dynamicity: cluster groups

Each cluster can autonomously be designed and optimized with dedicated tools. During the design of a cluster, one must pay heed to the three following steps:

- Partitioning: grouping actors together into partitions. This is typically used to higher the locality of the actors.
- Scheduling: organize the actors' activities in a time scale within a partition. This is typically used for pipelining operators.
- Mapping: a partition is mapped on a target.

The FlexTiles architecture makes it possible to provide the chip with different types of computing nodes and explore several partitioning and mapping. A cluster may have several possibilities of partitioning that will lead to several mapping so that the system will have to choose between several cluster-maps to map a cluster. First, before mapping tasks on the target, one must partition each cluster. The objective of the partitioning and the scheduling is to spread the needs of the actors of a cluster on the resources. Not only the computation needs have to be taken into account but also the required data throughputs and the memory spaces. One must find good candidates for parallelization.

We have to make a trade-off between the reconfiguration capabilities of the solution and the required period of this reconfiguration. The decision to reconfigure the system is taken by the virtualization layer. We identify the following approaches:

1) Inside a cluster group: we activate one cluster or another. In this case, the full cluster is replaced and the behavior of the cluster group is changed. This allows us to take into account the following approaches:
   a) A processing chain can be activated or deactivated just like in cluster group 1 and 2 where a static cluster (state2) or a "nop" cluster can be selected.
   b) A processing chain can be replaced by another one, according to external given constraints that can depends on the data to be processed or to chip sensors.
   c) A processing chain can be replaced by a less efficient one but also less consuming one in terms of resources or power.
   d) Elements of the static cluster can be remapped according to operating system instructions.
   e) Elements of a static cluster can be duplicated like in cluster group 5.
2) Outside a cluster group: the operating system evaluates each application according to its priority and can react in two ways:
   a) Remapping application: an application is relocated following the chosen partitioning in order to reach a better resources sharing with the other applications.
   b) Using degraded mode: an application is relocated in a degraded mode in order to execute an application with a higher priority.

## H. Software tools

As described above, the application is described as a set of static cluster. Each cluster can be independently optimized by using the proposed tools SpearDE [17] and Cosy [18]. These tools follow the model-based approach for dataflow applications and they are only able to optimize the applications with a static behavior. It is the reason why the separation between static part and dynamic part is necessary.

The first step is a graphical modelling based on the actor model described in the previous section and with an expression of the potential parallelism. The second step consists in mapping the actors on the different heterogeneous nodes represented in the architecture model of FlexTiles. SpearDE also allows to parallelize on several identical nodes. SpearDE is able to insert communication when needed between nodes. SpearDE is also able to determine the best elementary block of data to be computed with respect to the memory size (data strip-mining). Finally, SpearDE generates the mapped code used by Cosy.

In the Cosy framework, ACE has developed a Streaming compiler to convert annotated C code into SDF programs compatible with CompOSe kernel (see section IV-B). The CSDF programs are mapped on the MPSOC platform, i.e. tasks are allocated to processors, instructions and actor data are mapped to local memories, and communication channels are mapped to NoC connections, and memories in tiles or distributed memories. All embedded driver C code to configure processors, DMAs, NoC, and memory arbiters is generated.

The SDF3 tools [19] and the Æthereal tools [20] are used for this. NoC connections can also be computed at run time [21]. The hypothesis of this approach is to use a library of Elementary Processing (EP) executed by the accelerators. Each EP is generated by using the corresponding tool chain depending on the target DSP or eFPGA.

## IV. Virtualization Layer

### A. Information Management

The FlexTiles project will adopt the MDA (Monitoring, Diagnosis, Action) model proposed in [1]. Figure 6 illustrates three phases of the adaptation process to be performed on FlexTiles architecture: (m) monitoring, (d) diagnosis and (a) action. Each tile monitors a number of metrics (m) that drive an application-specific mapping policy. The information is then analyzed and a diagnosis of the current state of the NPU is given (d). Later, the tile may decide to push or attract tasks, which result in respectively parallelizing, or serializing the corresponding tasks execution (a). Additionally, each node may scale frequency up or down in order to either dealing with application requirements or saving power.

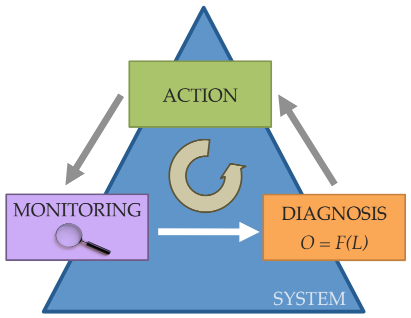

Fig. 6. Three Phases of Adaptation Process Performed on FlexTiles

*1) Monitoring System:* Table II summarizes the monitored information that must be provided by the architecture to the virtualization layer. In order to define the scope of the architecture we propose implementing the three monitoring systems: $(i)$ application performance, $(ii)$ (CPU / DSPs / Accelerators) workload and $(iii)$ temperature. Additionally, voltage and frequency might be considered as input for helping the architecture to make efficient decisions at run-time.

*2) Diagnosis:* The diagnosis phase is intended to draw some conclusions based on the monitored information and provides the most relevant information to the decision-making mechanism present in the action phase. Basically the diagnosis mechanism tries to extract instant or averaged platform behaviors such as: $(i)$ CPU is getting overloaded; $(ii)$ application's performance is decreasing due to perturbations in the system; $(iii)$ CPU is running most of the time in idle mode or even $(iv)$ the tile's temperature is too high;

Depending on the desired level of reactiveness of the system, this phase can be bypassed where monitored information is passed directly to the appropriate decision making mechanism. For example, whenever the temperature of the circuit is too high, the decision-making is activated and a rapid decision would be reducing processor frequency. It is important to observe that under such scenarios, a decision must be taken as soon as possible in order to avoid any damage to the system. On the other hand, for non-critical information, the diagnosis phase is executed before the decision-making mechanism starts.

TABLE II
Monitored Information in the FlexTiles architecture

| MONITORING SYSTEM | ACTION |
|---|---|
| Application Performance | Software services implemented in the operating system periodically monitor the performance of different applications running onto the GPP while hardware IPs keep track of the application performance in the DSPs and accelerators. |
| CPU / DSPs / Accelerators Workload | The operating system running on each tile monitors its CPU usage (General Purpose Processor - GPP) and hardware monitors keep tracking about the current usage of DSP and accelerators of each tile. |
| Temperature | Dedicated hardware IPs monitor the temperature of each tile as well as the temperature of the routers of the NoC. |

*3) Action:* It is in the action phase that decisions are taken based on the information obtained from both monitoring and diagnosis phases. This mechanism can take one of these actions: $(i)$ reduce processor frequency [22] whenever it is in idle mode or tile's temperature is too high; $(ii)$ increase processor frequency in order to meet application performance requirements; $(iii)$ migrate a task whenever CPU becomes overloaded; $(iv)$ re-allocate hardware blocks by using defragmentation approach upon new applications arrival; among others.

### B. CompOSe Operating System

The GPP nodes will be the only nodes embedding an OS to be able to schedule several threads. This OS will be based on the CompOSe [23] real-time operating system (RTOS). Each node runs the same CompOSe instance. CompOSe implements composable time-division multiplexing (TDM) between different applications on each node. Each application has its own model of computation, although in Flextiles we concentrate on Cyclo-Static-DataFlow (CSDF) graphs. Each application has its own task scheduler [24] and power manager [25], and the behaviour of each application is completely independent of and unaffected by that of other applications. We envisage that CompOSe will be extended such that applications can be started, stopped, and migrated, dynamically at runtime.

### C. Virtual bitstream

A virtual bitstream (V-BS) is the configuration of one task accelerated on the eFPGA, but without any precise information on the location of the hardware inside the fabric. The V-BS only contains the logic configured in each used Logic Block (LB) and how these LBs are connected with each other (including routing). As the eFPGA will be partially reconfigurable, the size of the V-BS will depend of the task implemented. The size of the V-BS should be smaller than a real "finalized" bitstream since it contains only the required and useful information, while the real bistream should embed the value of all reconfiguration bits (including all unused bits that need however to be fixed to value) in a particular domain. The de-virtualization is a process that read a V-BS corresponding to one accelerator, that finalize it with the information on its precise location in the fabric (typically x,y coordinates) and that generates the real bitstream as a set of Boolean values which will be downloaded as a configuration inside a portion of the eFPGA. To ease this de-virtualization process without placing and routing the bitstream again at run-time, we will define a regular and flexible routing structure. This routing technique will be based on relative and free placement of processing element and will ensure relocatability of V-bitstreams. A key point in this definition relies to the access of the tasks to external resources and to the corresponding master of the task inside the many-core architecture. In a first approach, we will consider that physical interfaces, i.e. access to the NoC by the 3DNIs, will be constrained and at the same place in all architecture domains. A dynamic local re-routing of the interface could be envisaged if necessary. This approach can be seen as a "just-in-time routing" of interface signals (few wires, not the entire tasks) to connect the task to a 3DNI, and will rely on the definition of an adaptable and flexible wrapper. The first solution adds complexity to the control, while the second one may have a large hardware overhead.

## V. Conclusion

The objective of this paper was to present an innovative approach of heterogeneous manycore. The main issues targeted by this architecture are the programmability in a context of heterogeneity and the self adaptivity to provide the best performances depending on the runtime requests of the more and more dynamic applications, to reduce the power consumption and to harness the temperature. The programmability efficiency is obtained through the use of data flow optimisation tools associated to the use of programmation model separating the static parts from the dynamic parts of the application. The adaptivity has been considered at the hardware level and at the software level. We propose to implement a reconfigurable technology on a dedicated layer in a 3D staked chip. This technology will be able to re-allocate or migrate the IPs in zero duration which will provide a high level of adaptivity. At the software level, the system will be able to monitor the workload, the power consumption and the temperature, to take decisions and to launch a re-allocation of the application. These new concepts will developed and validated in the context of the FlexTiles FP7 project.

## Acknowledgment

This research is sponsored by the European Commission under the 7th Framework program within the FlexTiles project (FP7 ICT-288248).

## References

[1] G. M. Almeida, "Adaptive multiprocessor systems-on-chip architectures: Principles, methods and tools." Lap Lambert Academic Publishing, 2012.

[2] FlexTiles FP7 project: Self adaptive heterogeneous manycore based on Flexible Tiles. [Online]. Available: http://flextiles.eu

[3] [Online]. Available: http://www.tilera.com

[4] Kalray: Manycore processors for embedded computing. [Online]. Available: http://www.kalray.eu

[5] L. Benini, E. Flamand, D. Fuin, and D. Melpignano, "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," in *DATE*, march 2012, pp. 983 –987.

[6] SMECY: Smart multicore embedded systems. [Online]. Available: http://www.smecy.eu

[7] FOSFOR : Flexible Operating System FOr Reconfigurable platform. [Online]. Available: http://users.polytech.unice.fr/ fmuller/fosfor

[8] L. Gantel, A. Khiar, B. Miramond, A. Benkhelifa, F. Lemonnier, and L. Kessal, "Data-flow programming for reconfigurable computing," in *ReCoSoC*, 2011, pp. 1–8.

[9] FREIA (FRamework for Embedded Image Applications) is a project founded by the ANR (the French National Science Foundation). [Online]. Available: http://freia.enstb.org

[10] D. Henry *et al.*, "3D integration technology for set-top box application," in *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on*, sept. 2009, pp. 1 –7.

[11] J. Lallet, S. Pillement, and O. Sentieys, "Efficient and Flexible Dynamic Reconfiguration for Multi-Context Architectures," *Journal of Integrated Circuits and Systems*, vol. 4, no. 1, pp. 36–44, 2009.

[12] C. Bernard and F. Clermidy, "A low-power VLIW processor for 3GPP-LTE complex numbers processing," in *DATE*, march 2011, pp. 1 –6.

[13] K. Goossens and A. Hansson, "The Aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *DAC*, Jun. 2010.

[14] Y. Thonnart, P. Vivet, and F. Clermidy, "A fully-asynchronous low-power framework for GALS NoC integration," in *DATE*, march 2010, pp. 33 –38.

[15] A. Hansson and K. Goossens, "Trade-offs in the configuration of a network on chip for multiple use-cases," in *NOCS*, May 2007.

[16] J. Pino and E. Lee, "A comparison of synchronous and cycle-static dataflow," in *Signals, Systems and Computers, IEEE*, Oct. 1995, pp. 204–210 vol. 1.

[17] E. Lenormand and G. Edelin, "An industrial perspective: a pragmatic high-end signal processing design environment at thales," in *SAMOS*, 2003, p. 5257.

[18] CoSy compiler development system. [Online]. Available: http://www.ace.nl/compiler/cosy.html

[19] S. Stuijk, M. Geilen, and T. Basten, "SDF$^3$: SDF For Free," in *ACSD*, 2006.

[20] K. Goossens et al., "A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification," in *DATE*, Mar. 2005.

[21] R. Stefan, A. Beyranvand Nejad, and K. Goossens, "Online allocation for contention-free-routing NoCs," in *INAOCMC*, Jan. 2012.

[22] A. Molnos and K. Goossens, "Conservative dynamic energy management for real-time dataflow applications mapped on multiple processors," in *DSD*, Aug. 2009.

[23] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken, "CoMPSoC: A template for composable and predictable multi-processor system on chips," *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, no. 1, pp. 1–24, 2009.

[24] A. Molnos, A. Beyranvand Nejad, B. T. Nguyen, S. Cotofana, and K. Goossens, "Decoupled inter- and intra-application scheduling for composable and robust embedded MPSoC platforms," in *MAP2MPSOC*, May 2012.

[25] A. Nelson, A. Molnos, and K. Goossens, "Composable power management with energy and power budgets per application," in *SAMOS*, Jul. 2011.