

# AN INTEGRATED PLATFORM FOR HETEROGENEOUS RECONFIGURABLE COMPUTING

*Bernard Pottier,  
Jalil Boukhobza and Thierry Goubier*

LESTER, FRE CNRS 2734  
Université de Bretagne Occidentale  
email: {pottier,boukhobza}@univ-brest.fr

## ABSTRACT

To our knowledge, it is the first time that a project creates a set of high level tools allowing to program different kind of reconfigurable units assembled in a System on Chip. This paper explains the general ideas followed to provide a software centric execution model, and techniques used to build a new framework open to various languages and technologies. Assessment of the framework will be achieved on a set of state of the art tools from industry and academy, applied to an heterogeneous multi-purpose reconfigurable circuit (MORPHEUS Project).

## 1. INTRODUCTION

Systems on chip are commonly built using off the shelf components: networks, processors, IPs and memories. Reconfigurable components are now emerging as a new alternative, with the choice of fine grain technologies (eFPGA), or coarse grain micro-architectures (data paths).

These components are useful for several reasons. They can provide a better match to some computational requirements, as it is the case of eFPGA for finite state machines, encoding and decoding data and non standard processing. Data paths also generally demonstrate a high level of concurrency in specific computations, providing speed-up and efficiency gains as compared to general purpose processors.

There are several known difficulties in using reconfigurable computing such as the absence of standard for specification, the technical difficulties in code generation and code management at execution, and moreover, the lack of criteria to measure the application performance inside the system. Another concern is the difficulties for general purpose

programming language (GPPL) to address efficiently such weak architectures.

The MORPHEUS Integrated Project<sup>1</sup> is working to address several of these problems at the same time. One objective of the project is to produce a consistent, reproducible set of tools allowing us to program a complex heterogeneous architecture from high level methodologies, ensuring efficiency of the execution and portability of the created development environments. The project is currently at mid-life and this paper will mainly relate the advances achieved in the execution model (section 2), and software synthesis techniques (section 3), in relation with architecture support. The objective of the execution model is to fix how data can be shared and exchanged between the GPPL working on data structures, stored in main memory, and accelerators working on data subsets stored in local memories. Synthesis techniques are based on an intermediate format for Control Data Flow Graph (CDFG) in which the accelerator algorithms are represented by high level compilers or methodologies. These CDFG are analyzed in correspondence with a model of the architecture to produce computation mappings for the target reconfigurable technologies. This framework is open and take benefits from APIs generated in various syntax to ease further developments.

The authors are mainly involved, with other actors<sup>2</sup>, in the definition and development of *Spatial Design* process organization and associated synthesis. This work also includes an extrapolation of the previous Madeo software for fine grain architecture modeling and synthesis (section 5), presenting recent advances achieved in this framework, and its contributions to the MORPHEUS efforts.

---

MORPHEUS is funded by grant IST 027342 of the FP6 program of EC. Thanks are due to the MORPHEUS Partnership, especially to TU Delft scientific direction that so much insisted to address autonomous processing for the accelerators. Task partners (Thales Research & Technology, Critical Blue) are thanked for the harmony found in the cooperation. The group contributing to MORPHEUS at UBO is Loic Lagadec (Synthesis, CDFG), Alain Plantec (STEP/CDFG), Ronan Keryell (Compilation), Jean-Christophe Le Lann (System modeling) and Damien Picard (Simulation).

---

<sup>1</sup><http://www.morpheus-ist.org/>

<sup>2</sup>Thales Research & Technology Embedded System Lab at Palaiseau and Critical Blue group at Edinburgh

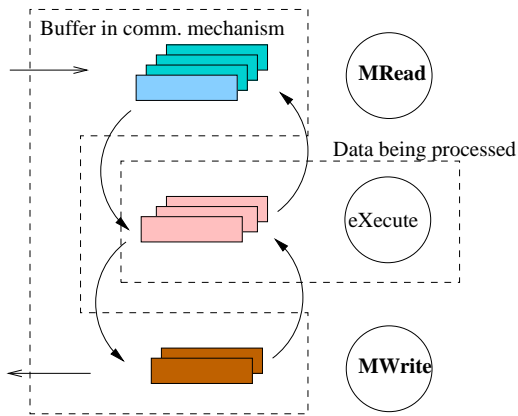
## 2. EXECUTION MODEL

### 2.1. Execution principle

One of the principles followed for the platform integration was that CPU and accelerators should share high level data transparently. The approach of execution is *software centric*, meaning that data rearrangements, communication, scheduling of execution, return to the high level program must be handled in a very simple way (to keep the explanations simple, we do not consider the case of computations spread over several reconfigurable units). Two key points in the software to hardware computation migration (figure 1) will be the structure of the configured function call (figure 4), and the delegation of main memory to local memories mappings to a communication device.

An accelerator-based execution is based on three conceptual process groups arranged as a pipeline where data are passed by buffers:

1. (MR) Memory reads, packing a set of high level data into buffers. These buffers are sent to the accelerator where they appear in local memories,
2. (X) Executing a computation step on a set of local data and storing data locally. Data can survive several steps locally, if necessary,
3. (MW) Memory write, from a set of buffers produced by the accelerator, to (possibly sparse) locations in main memory.



**Fig. 1.** Three stage micro-tasks pipeline partly running on a communication engine (CE), partly running on a reconfigurable unit (RU).

This pipeline fulfills the obvious interest to overlap communications and execution, with the guarantee of local data availability due to the prefetch of operands. The process pipeline is expected to be stable during a number of steps.

At each step the behavior of the three process groups can change, reflecting the computation's progress. Two evident motivations are the management of pipeline setup and finishing conditions. The process groups stand for several processes executing memory accesses, or execution level subprocesses. Step boundaries are synchronization barriers detected in the accelerator hardware.

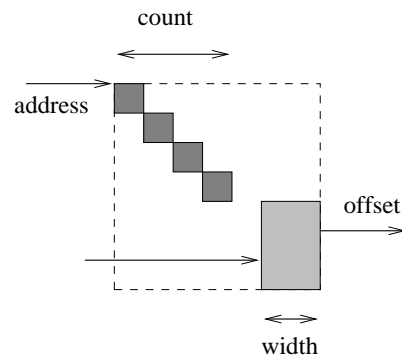
A preliminary observation on this execution model is the large grain of operands (chunk of data associated to matrix or image subsets as example), and the nature of the local code (graphs representing configured fixed or changing loop kernels).

*This machine is a large grain load-execute-store dedicated processor working directly on main memory for the benefits of a sequential program thread.* The mechanism develops a significant level of parallelism in terms of number of concurrent operations, communications, and local memory access capabilities.

### 2.2. Data access, transport and remapping

We are now considering the MR and MW stages merged as a group of processes. These stages need to extract data from memory, computing addresses on the fly and possibly optimizing for the bandwidth on main memory. They also execute the communications from/to main memory to/from local memories, possibly distributing data on local buffers locations.

Two mechanisms are merged in this stage: address computation and transport (over a SoC network, in the MORPHEUS case). Figure 2 show some kind of data distributions relative to high level constructs to illustrate the needs for address computations on main memory.



**Fig. 2.** Mapping of data structures in memory: rows, columns, blocks, diagonals. . .

Usually the memory access definitions are relative to a starting address known at run-time. Beside this, other important information are the count of data to be transferred, the width of data, and the offset from one element to the next

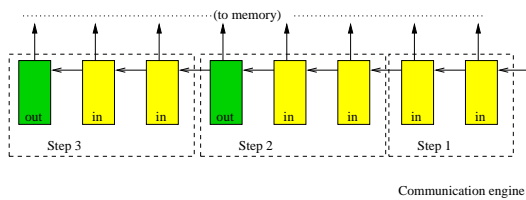
one. Some DMA are able to support this kind of issue, fetching linear pattern of data to pack or directly transport them to distant buffers. Other devices such as dedicated processors could be designed to process more complex situation, such as random indexing in an object memory.

### 2.3. Communication queues

The communication queues role is to define all the transfers and memory operations occurring during an accelerator call (figure 3), one step in advance of execution for the MR stage, one step later for the MW stage. An abstract program counter pointing to the step number structures a data oriented program:

1. executed by a communication device
2. synchronized with the local process of the execution

Each step of the program defines a group of tasks for MR, and another one for MW.



**Fig. 3.** Communication queue with 2 merged exchange directions. Groups of descriptors are associated to each step of the computation.

Practically, in the case of a DMA-based implementation, the DMA executes the group of task in one step, and waits for an acknowledgement coming from the computation barrier controller. Remapping of communication to local buffers is dependent from the hardware capability to address these buffers in a single memory space, or more simply, it fills one buffer for each of the activated communication processes from MR/MW.

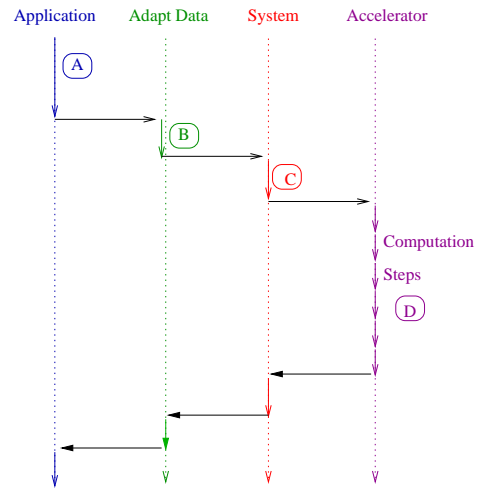
The high level data are now available in local buffers, or will be written back replacing these buffers.

### 2.4. Structure of the system call

Figure 4 shows the call structure from an high level function (left) to an accelerated one (right). On the left, the sequential program thread has decided to stop and to transfer the activity to an accelerator. This begins by an early schedule of the reconfiguration (A) generated from the GPPL compiler (if necessary), then (second vertical line-B) memory access and transport activities are defined, still in user mode.

Third, the system is called with a small parameter set characterizing the target device and the location of data definition in memory (C). Finally the system starts the accelerator that will progress step by step overlapping *communication to and from memory* with *local execution* (D). The operation schedule is decided statically during the set-up phase. This can be implemented in very different ways, either based on the communication queue structure, or from a dedicated controller, but it is critical that the communication and computation progresses remain synchronized on pipeline step boundaries.

The processes involved in the execution are produced as a composition of CDFG as it is described in section 3.



**Fig. 4.** Accelerator call from the high level compiler, showing the Set-up procedure, a preemptive system call, pipelined communication and execution overlapped steps, and return to the high level.

### 2.5. Execution model first assessment

The execution model has been implemented as a set of processes representing an high level computation. We have chosen a matrix product defined in a C program. This program includes a set-up phase of the communication queue in memory by a series of jumps over the data structure to define queue elements as part of rows or columns in the product operand and results.

A second process is a thread executing a simulation of a communication engine [1] written in SystemC. This process executes the communication steps as concurrent activities, transferring data between main memory and an array of buffers.

The third process was a simple C thread executing inner products on the array of buffers, synchronized with the communication process. Numbers collected during the simula-

tion had given interesting indicators on the practical interest of the effective concurrency of the MR/MW activities and the sequential execution on local buffers. The set-up of the queue requires much attention, with the possibility to do this at compile-time.

Another interrogation was related to the possibility to partition automatically a program to produce the code for the queue generation and the kernel code for the accelerator. This was also concluded positively based on PIPS source to source rewriting[2], at least for program libraries production.

## 2.6. MORPHEUS architectural support

MORPHEUS architecture provides enough support to allow the implementation of this model. The architecture is a SoC assembly with communication devices (bus and packet oriented network on chip), and a memory interface.

Reconfigurable units (RU) considered in MORPHEUS have very different micro-architectures, with two coarse grain IPs[3, 4], and one fine grain eFPGA[5]. RUs are encapsulated in wrappers and connected to the network on chip (NoC)

The wrappers provide a set of mechanisms to unify the use of the RUs: presence of local memories addressable by the NoC and the RUs, local configuration memory and configuration controller handling the set-up of application process graphs in the RUs.

The last important hardware devices are the DMA handling communication and memory access under the control of application processes.

A real time operating system (RTOS) supports the management of hardware resources, in relation with a C Compiler [6].

The MORPHEUS development system described in the following section will demonstrate how to address these components from a high level methodology for signal processing.

## 3. TOOLS ARCHITECTURE

As shown in figure 6, the entry point to present an algorithm as a computation graph is the CDFG representation model. In this section, we will focus on the software architecture allowing to translate the computation into an application circuit preserving portability and neutrality relatively to the input languages.

This translation is only one part of the mapping problem. It must be noticed that bringing data to the local memories must be achieved consistently with the local synthesis of the circuit.

As we proceed top-down from high level compilers, or methodologies, binding operations to local data is feasible if we track precisely operand locations (see section 2.5).

## 3.1. Algorithm representation

### 3.1.1. CDFG structure

The CDFG data organization captures the control, the data flow, and the program structure due to hierarchical nodes. The structure of the algorithm is reflected from condition, loops, replications, function call. Concurrency appears at two levels : as application process nodes and as a control structure inside the CDFG. Besides this, a type system can be used as labels on the graph edges.

In more details, this CDFG gives support for:

- hierarchy: with the ability to define graph hierarchies,
- concurrency: with nodes that contain branches to be executed concurrently in the application circuit,
- communication: nodes defined to represent data exchanges with implicit synchronization.
- computation operation: to cope with the computations performed by the application.
- control operations: dynamic `do while` and `while`, static loops `for`, tests `if then else`,
- function calls,
- memory operations: reading and writing data from and to arrays.
- data manipulation: reordering and data management in a structure as it will be seen in section 3.3.

It is expected that this intermediate structure will be useful for several purposes. As a target for high level compilers, it can be suitable to represent the common control structures, labelling edges with types, or having some inference machine to produce data characterization. The type system would allow very precise data definition (set of objects), as well as evasive definition (integers). It is compatible with previous versions of the Madeo tools (section 5).

A second point is the ability to ensure the transition between abstract computations and structured execution by circuits. A set of classes has been designed to support the mapping between operations and operators, and objects that are suitable for hardware devices are available in the CDFG (as example arrays to be allocated to memories). Concerning the execution structure, the process level allows to construct replications of processing elements at the algorithm level.

### 3.1.2. Simple example

The ping pong example is a variant of the one described in [7]. It consists of two processes that exchange data through channels. The receiver behaves differently depending on the value it picks up from the channel. This example focuses

on issues such as concurrent process execution, channels send/receive operations, loops, tests and hierarchical organization of the global application. The goal of this example is to provide a full yet simple-to-understand example of communicating processes expressed in one CDFG.

We go from a high level language specification (Smalltalk[8] for this example) to generate the CDFG by a compiler:

```

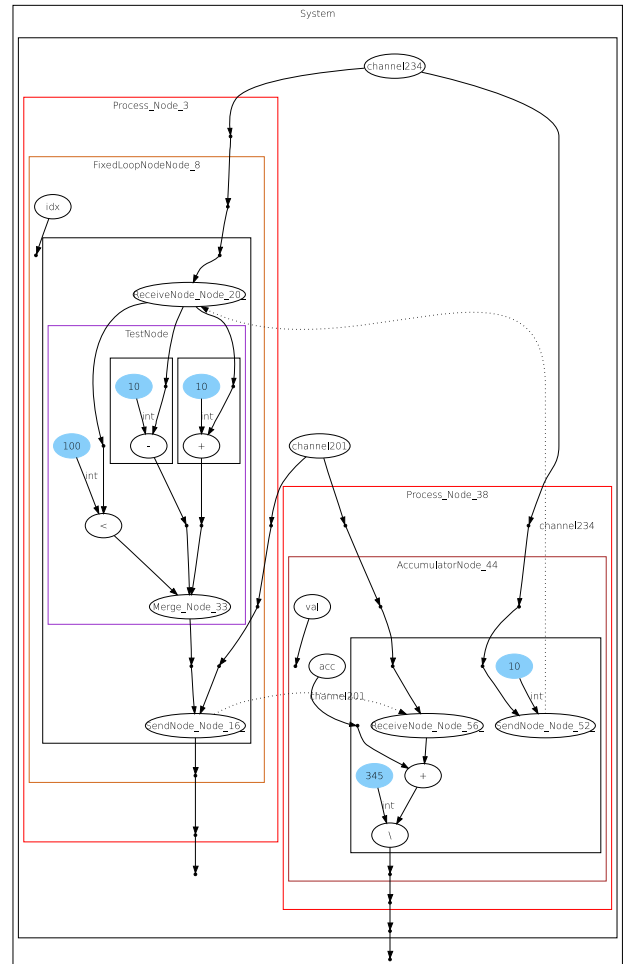
pingPongLoop
  "c1 et c2 represent the channels"
  |c1 c2|
  c1:= Channel policy: #blocking.
  c2:= Channel policy: #blocking.
  "This is the process Ping"
  [| x |
  1 to: 500
  do:
    [:idx |
    x := c2 receiveValue.
    x := x < 100 ifTrue:[10 - x]
    ifFalse:[10 + x].
    c1 sendValue: x]]
  fork.
  "This is the process Pong"
  [| y |
  (1 to: 500) inject: 0
  into:
    [:val :acc |
    c2 sendValue: 10.
    y := c1 receiveValue.
    (acc + y) \\ 345]]
  fork

```

... and we automatically end up with the CDFG representation shown in figure 5.

The first box named *system* corresponds to the global hierarchical CDFG containing the Smalltalk method CDFG. Both processes of the method correspond to two blocks to which the fork message is sent. Those processes are depicted as two separate hierarchical nodes in the method's CDFG: Process\_Node\_3 (P1) and Process\_Node\_38 (P2). They communicate via the channels channel1234 and channel201 named c1 and c2 in the method.

- The first process (P1) is composed of one loop (Fixed-LoopNodeNode\_8) which behavior is to wait for a data on the c2 channel (ReceiveNode\_Node\_21), execute a test (TestNode) which chooses the data to send (Merge\_Node\_33) and send them (SendNode\_Node\_16) on the c1 channel of the process P2.
- The new thing in P2 is the Accumulator node that models the inject: into: Smalltalk method.



**Fig. 5.** Ping-pong CDFG automatically produced from an high level language. Each hierarchical node is represented by a box and can be composed in its turn of other hierarchical nodes or atomic nodes. The latter are represented by an ellipse.

### 3.2. CDFG mapping description

The MORPHEUS *Spatial design*[9] framework ensures the translation between an algorithm description, presented as a CDFG, and an implementation description in formats required by reconfigurable targets.

- The former format is dependent on a choice of expression capabilities found in different high level languages or tools (sequential or parallel computation dependencies, loops, calls, . . . )
- The later format is dependent on the hardware elementary components (LUTs, operators, memories) and organization (connectivity, tiling). Architecture organizations are also described following a common model close to EDIF (extrapolated from a previous tool in Madeo).

The components are organized in a hierarchy of classes following high level functions, with attributes representing hardware characteristics.

This is an extrapolation of Madeo representation for FPGAs (still existing in figure 6 on the left).

Figure 6 shows a global view of the framework. The upper layer (input common format) is specified as an instance of an Express model, allowing front-end compilers from different programming languages to address the framework. The left part of the figure shows formal description of target architectures. The bottom blocks in the figure shows input programs mapped to target architectures as a network of components still associated to high level constructs.

The spatial design tools allow to check the semantic information on a CDFG. It also possible to check the feasibility of a CDFG on a given architecture model so that it can be translated into a mapped graph based on a particular architecture description. Those tools can interact to complete and improve translations. It is finally possible to check the consistency of the output graph and to manage the correspondence between the two CDFG and synthesize feed back information to upper layers tools.

### 3.3. Level of concurrency and synthesis

Two types of concurrency have been dissociated:

- System concurrency between processes:
  - many processes represented by CDFG can be issued and executed in parallel. Those processes interact explicitly at the system level, which behavior must be managed in the communication schedule. Those processes can embed graph of operators, arrays, and access to arrays. They are assembled by point to point synchronization

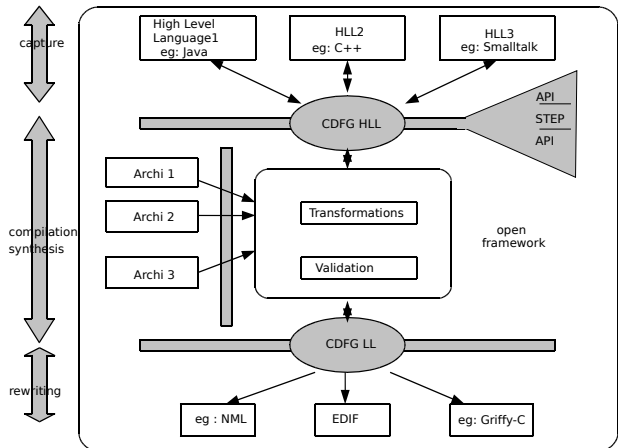


Fig. 6. The spatial design framework

mechanisms denoting rendez-vous, coordination on stream oriented data exchanges, or other solutions that synthesis would be able to handle.

- in complement with these processes, system coordination generated by the architecture synthesis framework will appear to enable step boundary barriers to occur, as well as data buffer switching (reallocation) to take place.

- Concurrency inside processes: this is taken into consideration at the level of CDFG in which we can define parallel nodes that can communicate and synchronize via channels.

To implement circuits, basic synthesis tools sweep simultaneously the concurrent CDFG from their starting point, always at system pipeline barriers. Once resource allocation becomes effective, each node is evaluated in terms of delays. The synchronisation points between CDFG processes are solved by delaying transfer operations until the partners are ready.

Another important issue in synchronisation is the interchange of memories between processes. This task is achieved by the controller process at barrier boundaries.

### 3.4. Portability and exchange

CDFG graph model enables program specification exchanges between the different tools in the Spatial Design activity, as well as the means to refine program structures for target reconfigurable architectures. The CDFG can come from different programming environment using a common API (Application Programming Interface) that insures portability. This portability is obtained from the ISO 10303 STEP /Express tools, and an API generator called Platypus [10].

The reference model for the CDFG is an Express model describing data structures and relationships organization. Platypus analyzes this model and automatically produces a set of APIs to build and exchange CDFG instances conforming to this model. Furthermore, the STEP file format is also automatically accessible from this set of APIs to guarantee accurate exchanges between different environments. This way, the CDFG can evolve quickly and interoperability is obtained between common development tools (C++, Java, Smalltalk...).

## 4. MORPHEUS INPUT TOOLS

### 4.1. SPEAR

The SPEAR DE tool [11] is at the top-level of the Spatial Design flow. SPEAR is a methodology for description of signal processing (data flow, array oriented) algorithms. Using SPEAR, the designer can compose and synchronize processes. SPEAR aims to manage for a data streaming application all that concern communications between processes and scheduling, except the processes cores. SPEAR can generate a model of *connectors* assembling processes to be implemented on reconfigurable units. Due to the model built in SPEAR, these connectors are presented to the synthesis framework in the form of a CDFG that defines order and index used to access a particular array (to be mapped on a local memory).

Another interesting point is that the source and target arrays for signal samples, located in main memory, can be addressed uniformly with local arrays. The difference is that in this case the transport mechanism will remap samples to local memories (or the inverse) consistently with the signal processing flow. The transport mechanism implies asynchrony while the local processing will use synchronized exchanges with the processing functions.

### 4.2. CASCADE

CASCADE [12] takes as an input a binary ARM program and performs an analysis on this program. The purpose of this analysis is to extract operation sub-graphs that could be executed faster using a dedicated coprocessor. After making a choice on the set of functions to be accelerated, CASCADE normally synthesizes a dedicated coprocessor that a normal CPU (i.e. the ARM processor) can call, and generates micro-coded VLIW instructions to be executed by the coprocessor. The coprocessor can access directly the data space of the program, as well as its stack. The original program is transformed automatically to call the coprocessor to perform the accelerated functions.

In the case of MORPHEUS, CASCADE is used to produce a CDFG representing a function for SPEAR. Therefore the two tools are complementary and allow to define

the structure and the behaviours in a processing chain.

In relation with the framework, this chain is an assembly of processes that read/write memories (SPEAR connectors) and compute transformations on the data flow (CASCADE behaviours for functions). The assembly is based on a stream of values exchanged at a rate decided by the synthesis tools.

### 4.3. Interaction between tools

Figure 7 shows how the tools interact around the CDFG models. Each connector in the figure is a process that defines how data are accessed in memories (arrays), in terms of order and indexes. The two processes are SPEAR functions specified in C, and translated into CDFG by the CASCADE work. Assembling all these processes is achieved by architecture synthesis software that schedules transfers according to the graph mapping on a target architecture.

On the left side of the figure, one can see the memories interfaced by the SPEAR connectors.

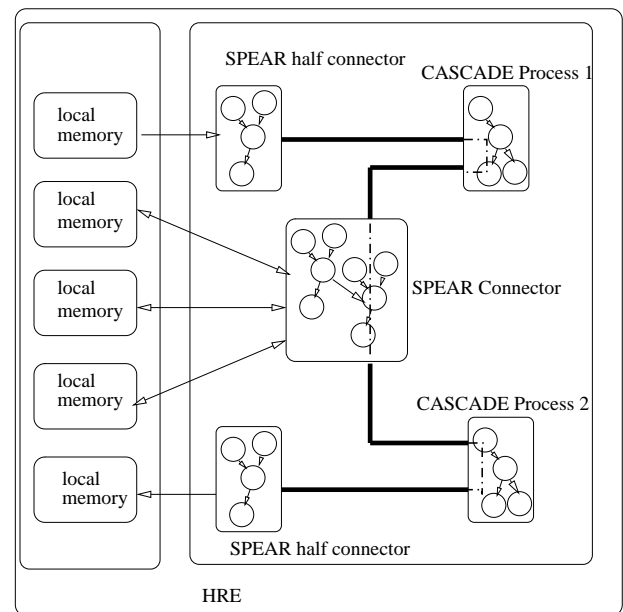


Fig. 7. Interaction between tools involved in the spatial design

## 5. MADEO CONTRIBUTION AND EVOLUTIONS

Madeo is initially a physical and logic synthesis tool chain targeting FPGAs. The specification is based on independent descriptions for arithmetic (implemented into system or user classes), high-level algorithms operating on these classes, architectural description of processing (processor networks,

operators), and description of the target architecture organization. Each of the description can be seen as a parameter in the synthesis process.

As an example, an algorithm designer can define one function using a general purpose object oriented language (Smalltalk-80, and varies the arithmetic support for data, observing the impact on hardware cost, efficiency, numeric issues. As an other example, a choice of architectures can be tested for a constant elementary or complex function, provided that this architecture is described in the Madeo framework.

Madeo is currently being integrated in the MORPHEUS platform with significant evolutions to bring its scope to system level activities. This section shows the integration between high-level algorithmic description, architectures description, and low level configurations or code for the hardware target, as well as the necessary interfacing components.

### 5.1. Status of Madeo

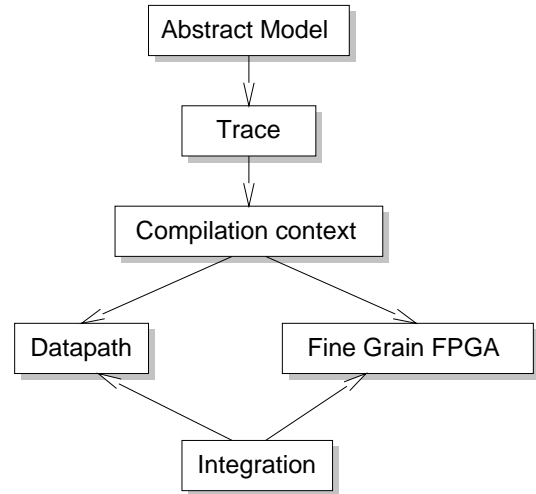
Madeo ([13], [14]) is divided in two parts: logical synthesis, and physical synthesis. Madeo compiles an algorithm by building a DAG (direct acyclic graph) of it, type edges with the arithmetic, propagate and evaluate the types along the DAG, and apply a chain of optimisations before the logical synthesis. The input is an algorithm written in Smalltalk and typed arguments, and the output is a optimised computational graph where every node is a look-up table carrying objects. The physical synthesis is done according to an architecture model and hardware interface. Madeo is used to:

- Model architectures and use them as target in its compilation or logic synthesis.
- Build libraries of components, targeted to one of the architecture modeled.
- Build generators.
- Compile to a coarse grain reconfigurable fabric (datapath).
- Do logic synthesis for a fine grain reconfigurable component (FPGA).

### 5.2. Recent results

Our recent development with Madeo is on block turbo codes decoding ([15], [16]) and the search for a software architecture adapted to this platform. Some elements of a block turbo decoder have already been synthesized on eFPGA with good results ([17], [18]), and our objective is to optimise it for the heterogeneous platform described above. Our approach is to use a functional abstract model, written at high-level in Smalltalk without any hardware dependency or arithmetic dependency, and to make our way down to the im-

plementation as a set of coarse and fine grain components interfaced on the target system (see figure 8).



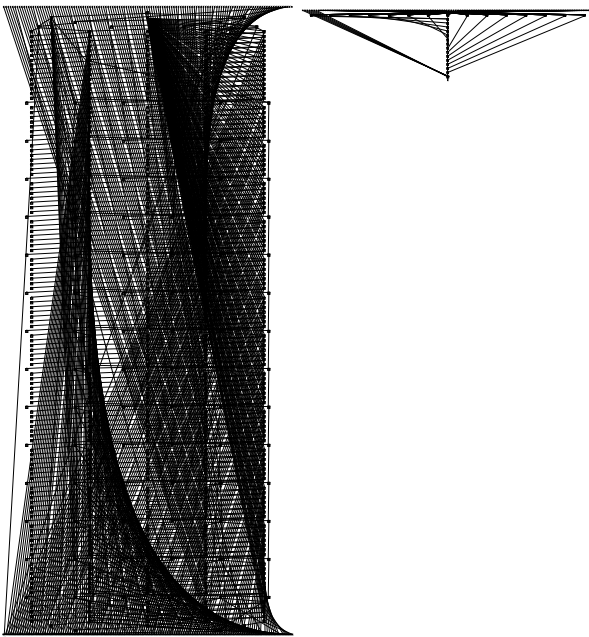
**Fig. 8.** From an Abstract Model to a compiled program on an heterogeneous reconfigurable platform: example of block turbo decoding

By so doing, we have been able to maintain an abstract model which is close to the mathematical description of turbo-decoding, while being able to estimate and optimise its mapping on this system. Madeo logic synthesis brings here very good results when handling finite field arithmetic in decoding, and Madeo physical synthesis allows us to map on different targets at will. The following process is applied:

- The first step is an abstract functional model of a block turbo decoder, very close to the algorithm described in [15]. It allows validation and testing of the application, and exploration of some of the domain parameters (arithmetic encoding, choice of block code).
- The algorithm is matched with inputs and an arithmetic (embedded in the inputs classes and types), to be executable and simulated. Here, the inputs define the block code used and the encoding of the received words of data to be decoded, the number of decoding iterations and a few others.
- The code is instrumented, and run to collect a trace of the operations, the types and their values, and the execution path. From that information, we are able to: dimension parts of the algorithm in terms of I/O and memory bandwidth, and number of operations. An additional possibility offered by the trace is to schedule the recorded operations on different targets, such as vector operations expressed by loop iterations on a data-path, and so explore the effect of different mappings of the code.

- Once partitioned, the algorithm components are specialised and given as input to Madeo, which will produce a configuration for fine grain reconfigurable units, linked with the necessary support logic extracted from a library. Madeo is also considered for targeting data-path-type units; its internal structure is well suited for that mapping.
- Additional tools, based on the trace, are used to generate the high-level program to control the execution and prepare the data transfers between the units and the memory.

The capabilities of Madeo are best illustrated with the synthesis of the syndrome calculation for a vector encoded with a block code  $BCH(128, 120, 4)$ . The original code uses directly  $GF(2^7)$  arithmetic; the operators (nodes of the graph) have all their outputs in  $GF(2^7)$ , and the input is a 127 elements binary vector. The DAG created by Madeo is initially large, with 654 operators, and is reduced through optimisations before logical and physical synthesis. The resulting graph (Figure 9) has finally 30 operators, with an average input size of 5 variables and one output variable, and synthesize to a small, but dense, circuit.



**Fig. 9.** Syndrome logical synthesis: the left graph is the initial DAG of the computation, the right is the reduced graph after Madeo logic synthesis

The result is an algorithm architecture well suited for execution on an heterogeneous reconfigurable architecture, with highly optimised processing components mapped to a fine grain reconfigurable unit, and highly parallel vector operations mapped to a coarse grain reconfigurable unit. This

mapping match the I/O capabilities of each reconfigurable unit: large, vector intensive operations on the large I/O capability of the data-path, while compact, bit vectors, finite field arithmetic is processed by the narrower fine grain unit.

## 6. PERSPECTIVES

An important objective of MORPHEUS is to keep application design as a software related activity. To reach this objective it has been decided to use a shared memory execution model.

A second important decision was to overlap communication and computation activities in fixed sequences. In addition to an expected immediate availability of data for the accelerator, this choice also rejects address computations to an external mechanism that could be optimized in relation with the network on chip, and the memory interfaces. This data exchange sequenced loop is also an excellent basis to match resource allocation with real data bandwidth during accelerator synthesis.

The application execution is split between a general purpose program and an *intelligent* accelerator/processor, sharing a queue that defines data location, and accelerator sequencing.

The queue is close to a message passing interface between processor and coprocessor, offering possibilities for more parallelism.

Technically, it is expected that the proposition of a language neutral standard for CDFG will encourage more software compiler developers to address reconfigurable System On Chip in a uniform way, and from different languages, including object-oriented languages.

## 7. REFERENCES

- [1] J-C. Le Lann and B. Pottier. A communication and control support mechanism for an heterogeneous reconfigurable soc. Technical report, LESTER, UBO, July 2006.
- [2] C. Ancourt, F. Coelho, B. Creusillet, F. Irigoien, P. Jouvelot, and R. Keryell. Pips : a workbench for program parallelization and optimization. In *European Parallel Tool Meeting'96*, France, October 1996.
- [3] A. Lodi, M. Toma, and F. Campi. A pipelined configurable gate array for embedded processors. In *ACM/SIGDA eleventh international symposium on FPGAs*, 2003.
- [4] V. Baumgarte, G. Ehlers, F. May, A. Nàœckel, M. Vorbach, and M. Weinhardt. PACT XPP-a self-reconfigurable data processing architecture. In *The Journal of Supercomputing*, volume 26. Springer, September 2003.

- [5] FLEXEOS cell library. Technical report, M2000, 2006.
- [6] E. Moscu Panainte, K.L.M. Bertels, and S. Vassiliadis. The molen compiler for reconfigurable processors. In *ACM Transactions in Embedded Computing Systems (TECS)*, February 2007.
- [7] B. Lin. Software synthesis of process-based concurrent programs. In *ACM DAC98*, June 1998.
- [8] A. J. Goldberg and D. Robson. *Smalltalk-80 : The Language and Its Implementation*. Addison-Wesley, 1983.
- [9] Seth Copen Goldstein. Computing without processors. In *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'04)*, pages 29–32, Las Vegas, NV, June 2004.
- [10] A. Plantec and V. Ribaud. PLATYPUS : A STEP-based integration framework. In *14th Interdisciplinary Information Management Talks (IDIMT-2006)*, September 2006.
- [11] E. Lenormand and G. Edelin. In *An industrial perspective: A pragmatic high end signal processing design environment at Thales*, volume 3133, Samos, Greece, 2003. Springer.
- [12] P. Ienne and R. Leupers. Chapter 9. In *Customizable Embedded Processors*. Elsevier, August 2006.
- [13] L. Lagadec, B. Pottier, and O. Villellas-Guillen. An LUT-based high level synthesis framework for reconfigurable architectures. In *Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation*, pages 19–39. Dekker, November 2003.
- [14] E. Fabiani, C. Gouyen, and B. Pottier. Intermediate level components for reconfigurable platforms. In *Synthesis, Architectures and Modeling of Systems (SAMOS 3)*, volume 3133, Samos, Greece, 2003. Springer.
- [15] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq. Near optimum decoding of product codes. In *IEEE GLOBECOM '94 Conference*, December 1994.
- [16] P. Adde, R. Pyndiah, and O. Raoul. Performance and complexity of block turbo decoder circuits. In *Third International Conference on Electronics, Circuits and System ICECS'96*, October 1996.
- [17] C. Dezan, C. Jago, B. Pottier, C. Gouyen, and L. Lagadec. The case study of block turbo decoders on a framework for portable synthesis on fpgas. In *HICSS (IEEE), Mobile Computing Hardware Architectures workshop*, January 2006.
- [18] C. Dezan, E. Fabiani, C. Gouyen, L. Lagadec, B. Pottier, C. Andriamisaina, and A. Pougou. Synthèse portable pour micro-architectures à grain fin, application aux turbo décodeurs et nano-fabriques. In *TSI*, 2006.