Computer Assisted Design and Integration of FPGA Accelerators in Aerospace Systems

Marco Lattuada, Fabrizio Ferrandi
Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Piazza Leonardo da Vinci, 32
20133 Milano, Italy
{marco.lattuada,fabrizio.ferrandi}@polimi.it

Maxime Perrotin
European Space Agency, ESTEC
2201AG Noordwijk, The Netherlands
{Maxime.Perrotin}@esa.int

Abstract—**The integration of Field Programmable Gate Arrays (FPGAs) in an aerospace system allows to improve its efficiency and its flexibility thanks to their programmability. To exploit these devices, the designer has to identify the functionalities that have to be executed on them and provide their implementation by means of Hardware Description Languages. Generating these descriptions for a software developer could be a very difficult task because of the different programming paradigms of software programs and hardware descriptions. To facilitate the developer in this activity, High Level Synthesis techniques have been developed aiming at (semi-)automatically generating hardware implementations of specifications written in high level languages (e.g., C). State of the art tools implementing such methodologies have not been designed for the integration with aerospace systems design flows, so significant adaptations could be required to the designer for integrating the hardware implementations with the rest of the design solution. In this paper the integration of a High Level Synthesis design flow in the TASTE framework (http://taste.tuxfamily.org) is presented. TASTE is a set of freely available tools for the development of real time embedded systems developed by the European Space Agency together with a set of its industrial partners. This framework allows to integrate specifications described in different languages (e.g., C, ADA, Simulink, SDL) by means of formal languages (AADL and ASN.1) and to early verify the correctness of the produced solutions. TASTE has been extended with Bambu (http://panda.dei.polimi.it), a tool for the High Level Synthesis developed at Politecnico di Milano. In this way the TASTE users have the possibility to specify which functionalities, provided by means of high level languages such C, have to be implemented in hardware on the FPGA without having to directly provide the hardware implementations. Thanks to the integration of the High Level Synthesis tool indeed, the framework is able not only to produce the hardware implementations, but also to integrate them in the rest of the aerospace system by automatically generating the whole architecture to be implemented on the FPGA. This architecture contains not only the implementation of the hardware accelerators, but also of the components required to transfer the data from and to the rest of the system and to correctly manage their size and endianness. The application of the extended framework to a real case study shows its effective usability.**

TABLE OF CONTENTS

1. INTRODUCTION

The evolution of the aerospace systems is characterized by the improvement of the on-board sensors which are able to capture larger and larger amount of data. However, the transmission bandwidth between them and the Earth stations has not equivalently grown. For this reason, to actually exploit the availability of larger amount of data, more and more pre-processing has to be executed directly on the aerospace system to reduce the data to be sent to Earth. Traditional high performance computing devices such as general purpose processors could provide the required computational power, but they are not suitable for these systems because of the requirements in terms of low power and high dependability. On the contrary, micro processors developed for space environment meet such requirements but cannot provide the required computational power. The solution which has instead been identified for solving this issue is the use of Rad-Hard Field Programmable Gate Array (Rad-Hard FPGA). This type of devices indeed has good characteristics in terms of reliability and power consumption and guarantees a significant amount of computational power.

The inclusion of FPGA devices in aerospace systems worsens one of the major issues in their design that is their heterogeneity. Indeed, not only the hardware components of a system can be very heterogeneous, but also the features of the specifications to be implemented on them can be very different. Moreover, in case of very large complex systems, a third source of heterogeneity can arise, that is the presence of different companies involved in the project, which potentially implies the presence of different design flows to be integrated.

Several approaches have been proposed to address the heterogeneity of space embedded systems. For example, Ludtke et al. [1] proposed a reconfigurable system composed of different processing elements (including a FPGA device), an ad-hoc operating system, and a middleware to allow the exploitation of the same architecture for the implementation of different specifications. The focus of this work is mainly on the characteristics that the hardware/software architecture must have, while no specific framework or tool are proposed to program it. On the contrary, the framework proposed in [2] and [3] aims at helping the designer in choosing the best combination of FPGA device and fault-tolerant strategy, but the design of the hardware accelerator for the particular combination is still demanded to the developer. Another possible approach to help the designer in the exploitation of FPGA devices has been proposed by Greco et al. in [4]. The USURP framework has been extended adding a Hardware Abstraction API, very similar to the GNU Scientific Library API, which allows to transparently access to a set of already implemented hardware accelerators. This approach has the advantage of not requiring knowledge of hardware design nor of HDL languages, but only a limited set of hardware accelerators are available: the functions not already included in the hardware library cannot be mapped on the FPGA. A whole framework for the design of space systems has been proposed also by Deshmukh et al. [5]. The authors proposed a new Domain Specific Language for describing the different components of an application targeting space systems and provided a framework to automatically generate the code to implement it. FPGAs devices are however not considered in this work and developer is forced to use a new unique language for developing the whole application. The TASTE framework [6], whose extension is proposed in this paper, instead hides the modeling language to the developer allowing to use different languages (HDL included) to describe and implement the single components of the application.

Most of the presented design frameworks aim at helping the designer in integrating FPGA accelerators in complex systems. However their integration is not the only issue about hardware accelerators: the other main problem is how to generate these accelerators. Their design indeed requires specific skills, such as the knowledge of the HDL languages, which often are not owned by software designers or by aerospace engineers. The work presented in this paper aims at solving this issue. Its main contribution is the integration of High Level Synthesis methodologies, which aim at the automatic generation of hardware accelerators, in a framework for the development of aerospace systems. This integration allows to design space systems which include hardware accelerators without any knowledge of hardware design techniques nor HDL languages.

The rest of this paper is organized as follows. Section 2 presents the background of this work describing the TASTE framework, the High Level Synthesis and the *Bambu* tool. Section 3 presents the integration among them while Section 4 presents an example of application of the integrated design flow. Finally, Section 5 presents the conclusions of this work and proposes some possible future works.

2. BACKGROUND

In this section the background of the design flow presented in this paper will be described. First the TASTE framework will be presented, then the High Level Synthesis will be introduced and finally *Bambu*, the open source tool implementing High Level Synthesis which has been integrated in TASTE, will be described.

The TASTE Framework

TASTE (*The ASSERT Set of Tools for Engineering*) [6] is a development framework for the design of applications for real time safety-critical embedded systems. The framework was originally created in 2008 as the final result of ASSERT (*Automated proof based System and Software Engineering for Real-Time applications*), a research project co-founded by the Sixth Framework Programme for Research and Technology Development of the European Union, which was coordinated by the European Space Agency and which involved about 30 industrial and accademic partners. In the following years European Space Agency has continued to support the framework by funding several follow-up activities to maintain and extend it.

The TASTE framework is composed of a collection of tools, most of which released under GPL/LGPL license, aimed at building in a semi-automatic way a distributed real time system. It supports different operating systems (RTEMS, Linux and Linux with Xenomai), different processors (x86, x86-64, LEON2, LEON3, and ERC32 BSP) and, by means of the integration of device drivers as functional models, external devices (e.g., ethernet network interfaces, serial ports, Spacewire interfaces, etc.). The main aim of the framework is to allow to the system designers to focus their attention on the design of the algorithms composing the specification and not on the implementation details related to the particular combination of operating systems and hardware components. Nevertheless, the adoption of a single unique language (e.g., UML) for modeling each aspect of a potentially very heterogeneous system has been considered unrealistic. On the contrary, the TASTE framework does not try to remove or reduce the heterogeneity of the system, but tries to hide this heterogeneity to the designer without requiring to adopt a unique formal modeling language. Indeed the single parts of the systems can be described by the designer in the preferred language: TASTE currently supports Matlab/Simulink, SDL, C, Ada, and VHDL as input description languages. Starting from so different languages, the TASTE framework hides in a transparent way all the details about communication among heterogeneous subsystems, but still allowing to verify at design time in a formal way the critical properties of the generated system. Section 3 will show how the TASTE framework generates all the code necessary to implement the calling of a hardware function by a software function, without requiring to the designer any knowledge about how this is actually implemented.

The design flow implemented in the TASTE framework is presented in Figure 1. The main steps are:

- *Capture Functional Architecture*: in this phase the designer builds the *Interface view* of the application. This view defines which are the different functions which compose the designed application and which are the interfaces which allow interaction among them. The TASTE framework provides a graphical tool (*Interface view editor*) to accomplish this activity.

- *Design SW Components*: the single software components are implemented by the designer with the languages supported by the TASTE framework. Different components can be described with different languages. The interfaces of such components are automatically generated by the TASTE framework so that the designer has only to focus on their implementation and not on their interaction.

- *Software Implementation*: in this phase the designer builds the *Deployment view* of the application which describes the target architecture (i.e., which are the hardware components, how they are connected, and the operating system) and then assigns each function of the *Interface view* to one of these components; the TASTE framework includes a graphical tool (*Deployment view editor*) to accomplish this task; next, the framework automatically generates the glue code which allows the different software components to interact and exchange data; finally the software system is deployed on the target.

Beside the tools aimed at implementing these steps, the TASTE framework integrates also a set of tools for analyzing the designed application and in particular to perform scheduling analysis (MAST [7] and CHEDDAR [8]).

During the different steps of the design flow, the TASTE

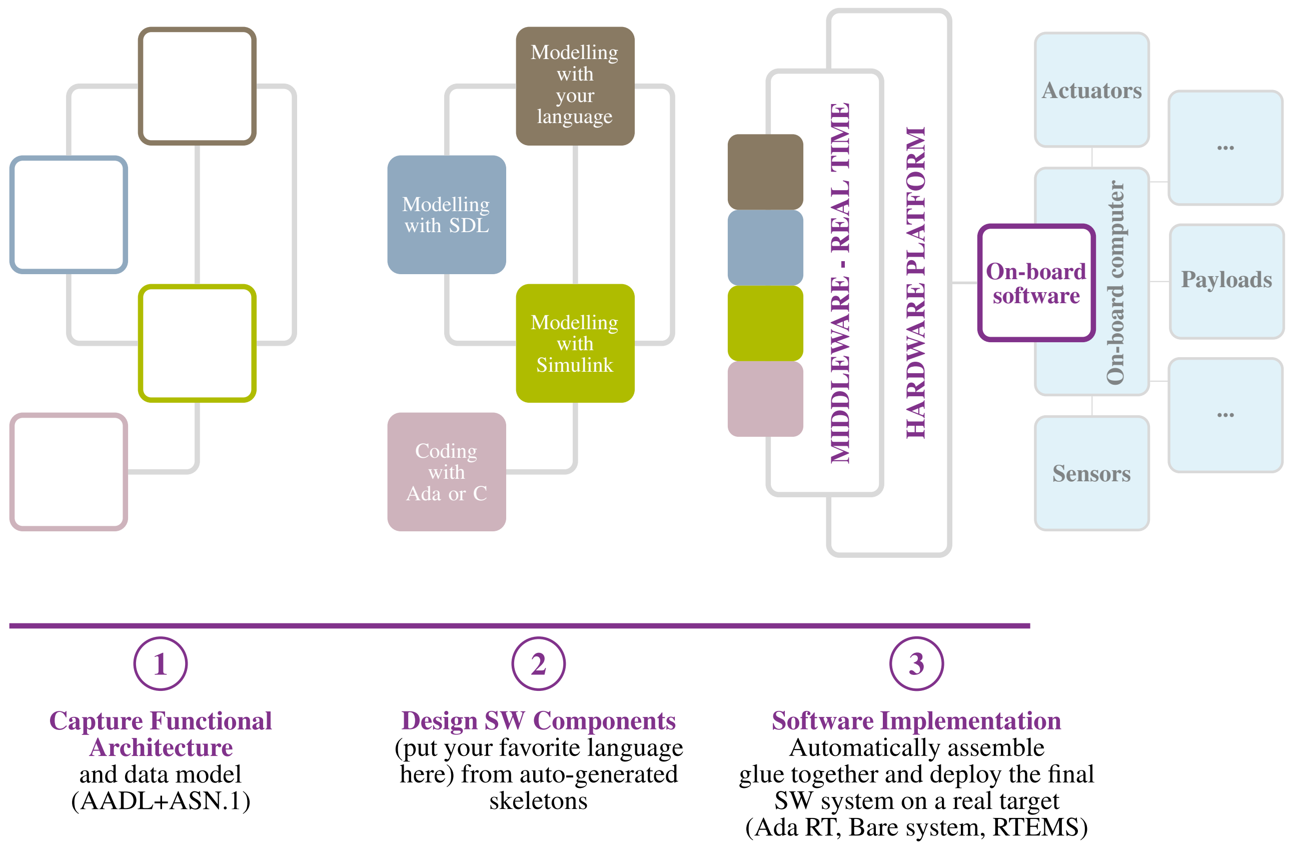
Modelling with your language
Modelling with SDL
Modelling with Simulink
Coding with Ada or C
MIDDLEWARE - REAL TIME
HARDWARE PLATFORM
Actuators
...
On-board software
On-board computer
Payloads
...
Sensors
1
Capture Functional Architecture
and data model (AADL+ASN.1)
2
Design SW Components
(put your favorite language here) from auto-generated skeletons
3
Software Implementation
Automatically assemble glue together and deploy the final SW system on a real target (Ada RT, Bare system, RTEMS)

Figure 1. The design flow of the TASTE framework.

framework exploits two languages to formally describe the characteristics of the designed application:

- *AADL* [9], which is used to describe the *Interface view* and *Deployment view*; note that AADL descriptions are automatically generated by the tools of the TASTE framework, so its knowledge is not required to the designer.

- *ASN.1* [10], which is used to specify the type of the data exchanged between the different components of the designed application; the framework already provides the ASN.1 descriptions for the basic types (e.g., integer, float), but the description of more complex types has to be provided by the designer. The adoption of this standard in the TASTE framework guarantees the correct interaction among functions assigned to different processing elements with different endianness and different data size.

While the adoption of AADL can be completely ignored by the designer, the adoption of ASN.1 has to be taken into account since it impacts on the *Design SW Components* phase which is mainly performed by the designer. Not only the designer has to describe by means of this language all the types used in the function interfaces, but he or she must adapt function implementations to them. In particular, in case of C functions, the original parameter types must be replaced in the source code with C structure types wrapping them. These structures are then used as parameter types during this phase and in the following guaranteeing the correct communication among different TASTE functions. The TASTE framework can automatically generate the C function signatures with ASN.1 based types, but the designer has to adapt the implementation of the C function bodies to them. For example given the C function:

```
void swap_array(int in_arg[2], int out_arg[2])
{
   out_arg[0] = in_arg[1];
   out_arg[1] = in_arg[0];
}
```

the C implementation must be modified to use the C ASN.1 compliant signature (which is automatically generated by the TASTE framework):

```
void function1_PI_swap_array(\
   const asn1SccIntPair * IN_in_arg, \
   asn1SccIntPair * OUT_out_arg)
```

where `asn1SccIntPair` is a structure containing a field of type integer array. Changing the C types of the formal parameters implies that the bodies of the functions have to be accordingly modified by the designer to be analysed and exploited in the TASTE design flow. Note that this holds only for those C functions which implement TASTE interfaces and so which can be called from components different from the one to which they are assigned. The functions which have to be synthesized in hardware have not to be modified as it will be shown in Section 3. Finally, all the C functions which are local to a TASTE function and all the library C functions (i.e., all the functions which do not implement an interface) have not to be modified.

The High Level Synthesis

High Level Synthesis [11] is a design flow composed of a set of methodologies aimed at automatically generating an ASIC or FPGA implementation of a high level specification.

The high level specification, which is the main input of the whole design flow, is usually formally defined by means of standard high level languages or by means of a subset of them; the languages supported by most of state of the art methodologies and tools are ANSI C/C++ and SystemC. The output of the High Level Synthesis flow is the description at *Register Transfer Level (RTL)* of the hardware architecture implementing the functionality. This description is written by means of a *Hardware Description Language (HDL)* such as VHDL or Verilog. For complex specification (e.g., composed of different functions) a hierarchical architecture composed of different modules can be generated.

The modules implementing the single functions include two different parts: the control logic and the data path. The control logic is modeled as a Finite State Machine which handles the routing of the data within the data path and the execution of the single operations. The data path actually implements all the operations that have to be executed and stores their input and output.

The whole High Level Synthesis flow is quite similar to a software compilation flow: it starts form a high level specification and produces low level code after a sequence of analysis and optimization steps. Nevertheless, since the target of the High Level Synthesis flow differs so significantly from the target of compilers, only a limited set of these steps are shared. Even if the same HDL language can be used to describe architectures implemented for different families of devices, the High Level Synthesis flow is not target independent but takes into account information about the target device. Moreover, FPGAs do not have a fixed operating frequency, but this can be decided by the designer or forced by devices (e.g., sensors or actuators) connected to it. Which is the target operating frequency of the circuit is another information which the High Level Synthesis must consider to generate an accelerator correctly working at that speed. In Section 3 it will be shown that the target frequency of accelerators developed for the TASTE FPGA architecture is 100Mhz because of the constraints of the rest of the architecture.

Like in a software compilation flow, three different phases can be identified in the High Level Synthesis flow: *front-end*, *middle-end* and *back-end*. In the *front-end* the input code is parsed and translated in a intermediate representation which will be used in the following parts of the flow. In the *middle-end* target independent analyses and optimizations are performed. Some of these steps are the same applied in a software compilation flow (e.g., data flow analysis, loop recognition, dead code elimination, etc.). Note however that not all the software code optimizations are profitable also when the target is a hardware accelerator. For example, the effects of transformations like function inlining and loop unrolling can impact much more on resource utilization in case of hardware devices. Finally, in the *back-end* the actual hardware architecture is generated. The steps composing this phase, which are the main difference with respect to software compilation flow, are:

- *Functions Allocation*: the hierarchy of the modules implementing the functions of the specification is built.

- *Memories Allocation*: the memories for storing aggregate variables (arrays and structures), global variables, and dynamically allocated data are instantiated.

- *Resource Allocation*: the functional units for executing all the operations are allocated; note that the characteristics of a functional unit can differ significantly according to the considered target device; moreover, there can be several available implementations of the same functional unit for a given device which differ in terms of area usage and performances.

- *Scheduling*: the order of execution of operations is decided; multiple operations can be scheduled simultaneously.

- *Finite State Machine Construction*: the Finite State Machine controlling the evolution of the computation is built.

- *Functional Unit Binding*: each operation is assigned to a particular instance of a functional unit.

- *Register Binding*: the opportune number of registers is instantiated and the scalar variables are assigned to them; note that differently from a general purpose processor the number of registers is not fixed and their number can be significantly larger.

- *Interconnection Binding*: the interconnections connecting elements of the datapath (registers and functional units) are instantiated and the data transfers are assigned to them.

- *Code Generation*: the HDL description corresponding to the designed architecture is produced.

The RTL architecture produced by the High Level Synthesis flow is not the architecture which will be actually implemented on the FPGA. The RTL architecture indeed is transformed in a logic gates architecture by means of Logic Synthesis [12]. In case of FPGA devices, this process is performed by means of tools provided by device vendors.

Bambu

The integration of a High Level Synthesis flow in TASTE framework has been obtained by including *Bambu* [13] in the design flow. *Bambu* is an open source tool, part of the PandA framework [14], developed at Politecnico di Milano and aimed at assisting the designer during the High Level Synthesis of complex applications. *Bambu* is written in C++, it can be freely downloaded under GPL license and it has been tested with different linux distributions (e.g., Centos, Debian, Ubuntu).

In the default High Level Synthesis flow, presented in Figure 2, *Bambu* takes as input one or more C source code files containing the application or the parts of the application which have to be synthesized in hardware. The output of the tool is a HDL description of the hardware accelerator, the scripts for its logic synthesis and eventually the scripts and the testbench for simulating its execution. Previous versions of *Bambu* were able to generate only Verilog descriptions: the backend for the generation of VHDL descriptions has been added to make easier the integration of the generated modules with the rest of the TASTE FPGA architecture. The flow is fully automatic, but can be controlled by the designer by means of a set of options and configuration files (i.e., xml files). For example it is possible to specify which is the C function that has to be synthesized in hardware in case the C source code contains more than one. *Bambu* supports different devices produced by the three main FPGA vendors (i.e., Altera, Lattice, and Xilinx). The list of supported devices can be easily extended by means of xml configuration files. Bambu relies on the GCC [15] frontend, which is interfaced by means of a set of GCC plugins, to perform

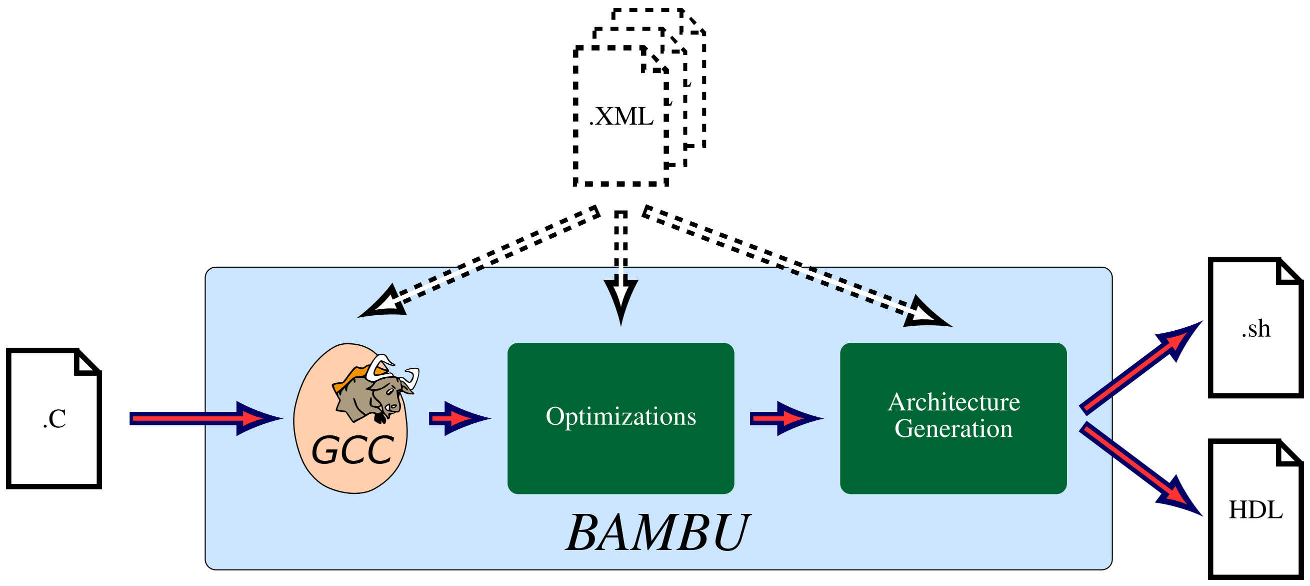

Figure 2. Default High Level Synthesis flow implemented in *Bambu*
.

the parsing and the initial analysis of the C source code. The plugins dump the intermediate representation of GCC after the target independent optimizations in a text format which can be parsed by *Bambu*. All the versions of GCC since GCC 4.5 are currently supported. By exploiting GCC frontend, *Bambu* can parse all the C source code which is compliant with the ANSI C Standard. Moreover, since the most significant state of the art target independent compiler optimizations are already applied by GCC, they have not been implemented in *Bambu*. Some FPGA oriented optimizations, not implemented in GCC because relevant only when the target is a hardware accelerator, have been implemented as intermediate steps of the *Bambu* High Level Synthesis flow.

Bambu is able to synthesize most of the C constructs, so that it does not impose any relevant restriction on the input code. For example, *Bambu* supports function calls, pointer arithmetic, dynamic allocation of memory, and floating point arithmetic. On the contrary *Bambu* does not support recursive function calls and function returning structures. All the pointer arithmetic performed by the hardware accelerators produced by *Bambu* is based on 32bit pointer size. Moreover all the internal data of hardware accelerators generated by *Bambu* have little endianness and the same endianness is adopted for exchanged data.

To sum up, the main features which have made *Bambu* suitable to be integrated in the TASTE framework are:

- it allows the automatic generation of hardware accelerators implementing C functions;
- it is a open source tool;
- it supports the FPGA board supported by the TASTE framework;
- it can be used at command line so that it can be easily integrated in more complex design flow;
- its High Level Synthesis flow can be customized by means of XML configuration files.

3. AUTOMATIC GENERATION OF FPGA ACCELERATORS

This section presents the new design flow which allows the automatic generation of hardware accelerators in the TASTE framework. This objective is accomplished by integrating an enhanced version of *Bambu* in the TASTE framework. Selecting which are the functions that have to be synthesized in hardware is demanded to the designer of the application: the integration of automatic techniques for performing HW/SW partitioning [16] is out of the scope of this work. The assignment of a function to the FPGA is performed by selecting VHDL as its implementation language. In the following it will be shown how *Bambu* retrieves this information from the *Interface view* produced by the rest of the TASTE framework to identify which are the functions to be synthesized. Since *Bambu* accepts only C source code files as input, only functions written in this language can be synthesized as hardware accelerators. The support to the automatic generation of hardware accelerators starting from SDL has been indirectly added by modifying *OpenGEODE* [17]. The tool has been extended by adding a backend for the generation of C source code starting from SDL representation. This C source code can then be used as input of *Bambu*, so that, by combining *OpenGEODE* and *Bambu*, it is possible to generate hardware modules starting from SDL descriptions. All the other languages supported by the TASTE framework are instead not supported and hardware accelerators cannot be automatically created starting from them.

The only FPGA board supported by the current version of the TASTE framework is the GR-CPCI-XC4V board [18]. This board has been developed as a co-operation between Aeroflex Gaisler and Pender Electronic Design. It is a compact PCI board containing a Virtex4 XC4VLX FPGA, 16 Mbyte of FLASH prom and up to 256 Mbyte of SDRAM. Access to this device is provided in the TASTE architecture by means of the PCI bus: the TASTE framework automatically generates

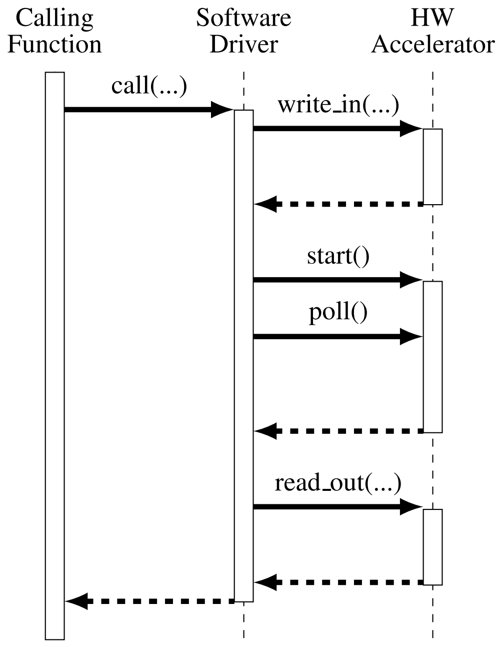

Figure 3. The Sequence Diagram of Hardware Function Invocation.

the software drivers for interfacing the accelerators mapped on the FPGA. Each input and output parameter of each synthesized function is assigned to a particular memory address. The steps performed during the invocation of a function mapped on the FPGA are shown in Figure 3. When a function has to invoke a function mapped on hardware, it calls its driver (`call(...)`) which writes (`write_in(...)`) in the opportune FPGA memory addresses the input parameters through the PCI bus. After that all the input parameters have been written, the driver starts the execution of the hardware accelerator by writing (`start()`) a memory mapped control register. Next, it continuously checks (`poll()`) for the value stored in the memory mapped control register until the hardware accelerator ends its computation. Finally it performs a set of memory readings (`read_out(...)`) aimed at retrieving the output of the hardware accelerator computation, and then returns these data to the function which performs the call to the hardware module. Note that a different driver is automatically generated for each designed application since it depends on the particular signatures of the functions mapped on hardware.

The software drivers hide most of the implementation details of the interaction between the general purpose processor and the FPGA, but they are not sufficient to make software functions and hardware modules communicating. Indeed, as in the software part of the application it is necessary to use drivers to fill the gap between the low level bus and the application, in a similar way on the FPGA device it is necessary to instantiate some components to connect the PCI bus with the accelerators. The architecture which has been designed to make accelerators generated by *Bambu* accessible from the PCI bus is presented in Figure 4. The hardware accelerators are not directly connected with the PCI bus, but are directly connected to an internal communication infrastructure based on the *ARM Advanced Microcontroller Bus Architecture (AMBA)*[19]. AMBA bus is an open standard on-chip interconnect specification, originally developed by ARM, aimed at facilitating the interconnection on System-on-Chip of components developed by different designers. The components used in the TASTE FPGA architecture to implement such type of communication infrastructure are taken from the GRLIB IP Library [20], a set of reusable IP components designed for FPGA released under GNU GPL License. The advantages of using such type of communication infrastructure are:

- it allows to generate architecture containing more than one hardware accelerator;
- each hardware accelerator can be designed independently from the rest of the system;
- it hides the implementation details of the selected FPGA board since for some components (e.g., the PCI target) it provides ad-hoc implementations for all the supported boards;
- it facilitates the porting of the TASTE FPGA architecture to different boards.

More in details the exploited components are:

- the *PCI target interface* component which implements the PCI slave interface toward the extern of the board and a AHB master interface towards the internal communication infrastructure. This component acts as a bridge allowing the communication between the PCI bus and the AMBA bus.
- the AMBA bus, composed of a AHB bus, a APB bus and a bridge connecting them. The hardware accelerators are connected directly to the APB bus, so they have to implement a slave interface for this type of bus. APB bus presents some limitations with respect to the AHB bus (e.g., reduced size of address space assignable to the single slave), but it has been preserved to maintain compatibility with the previous versions of the TASTE FPGA architecture.

Figure 5 shows the design flow implemented in *Bambu* for the integration in the TASTE framework. *Bambu* uses four different types of input files to generate the TASTE FPGA architecture:

1. *Interface View*, i.e., the file specifying the different functions composing the application;
2. *Data View aadl files*, i.e., the aadl descriptions of the types adopted in the interfaces of the functions;
3. *Data View ASN.1 files*, i.e., the ASN.1 descriptions of the types used in the interfaces of the functions;
4. *C Source files*, i.e., the files containing the implementation of the functions which have been mapped to the FPGA.

Since *Bambu* is able to retrieve information about the data types directly from the *Interface View*, the automatic generation of the C ASN.1 compliant signatures and the implied modification of the bodies of the functions, which have been described in Section 2, are not required (i.e., the tool takes directly as input the legacy C source code). The High Level Synthesis flow starts from the parsing of the *Interface View* file, which is the only one that has actually to be explicitly provided to the tool. All the other files previously listed indeed are automatically identified by the tool recursively

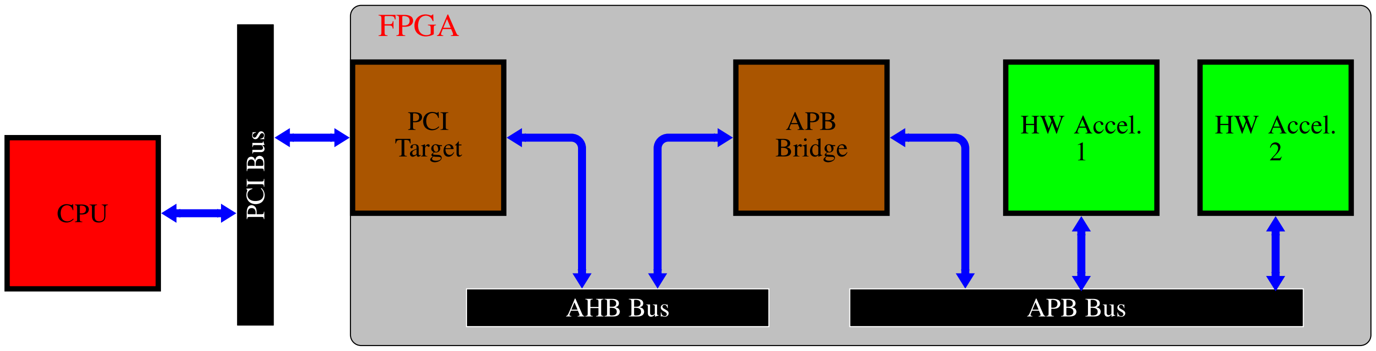

Figure 4. The TASTE FPGA Architecture.

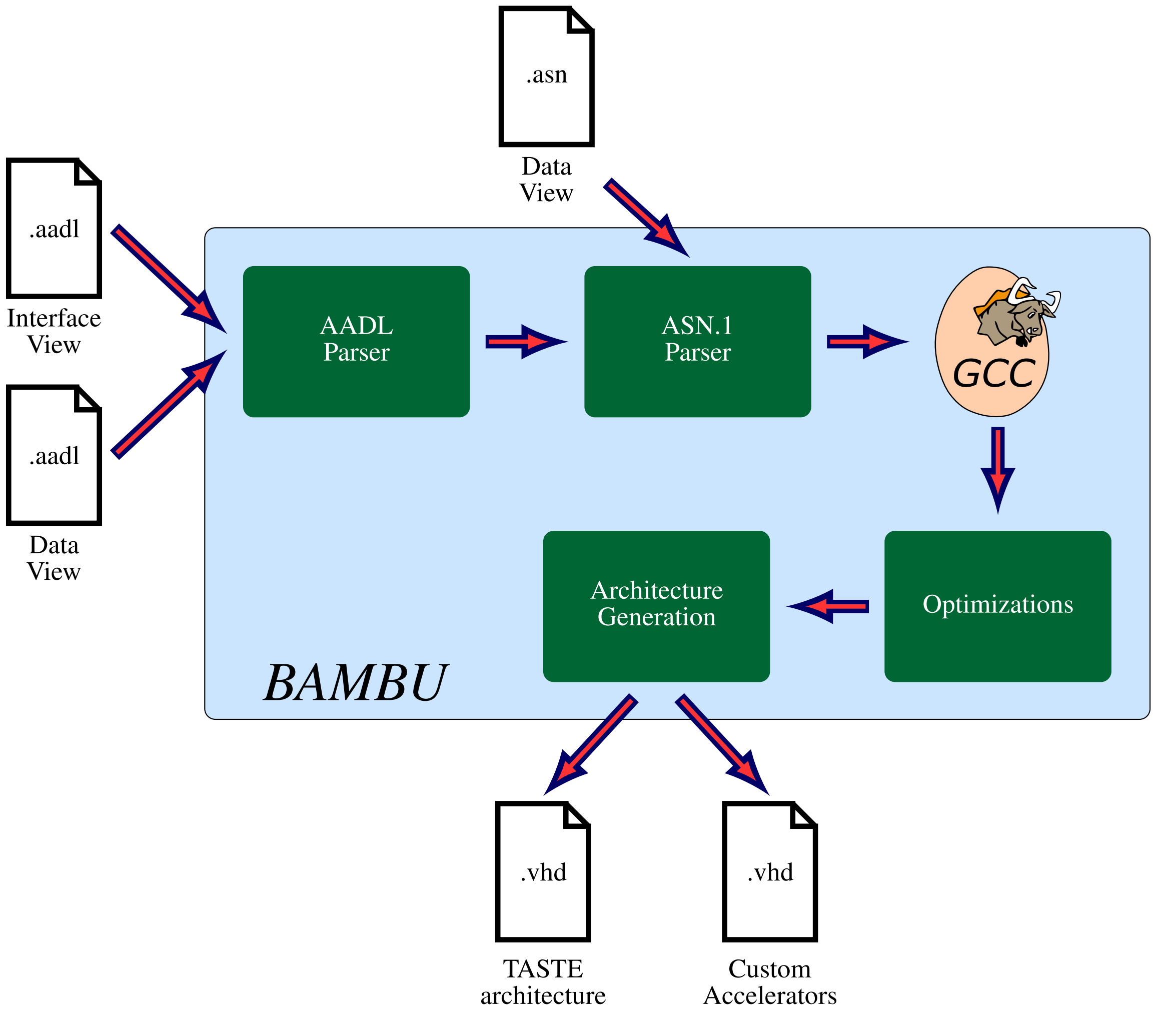

Figure 5. The High Level Synthesis flow targeting the TASTE FPGA architecture.

analysing the extracted information. From the *Interface View* file three types of information can be retrieved:

- The interfaces which must be implemented in hardware and the type of the input and output parameters.

- The source files containing the C implementation of these interfaces.

- The list of the *Data View aadl* files to be analysed; from these aadl files, the tool extracts the information about the *Data View ASN.1* files describing the types used by the interfaces to be implemented.

Next step of the new design flow implemented in *Bambu* is the analysis of the *Data View ASN.1* files. For each ASN.1 type the collected information is:

- *Its structure*: the software driver includes the padding data in aggregate parameters (i.e., structures and arrays) to guarantee the portability of exchanged data; *Bambu* must know this information to remove padding when necessary.

- *Its size*: ASN.1 allows to specify the size of the exchanged data also for basic types. For example, it is possible to specify the range of an integer parameter. *Bambu* can exploit this information to improve results of its bit value optimization.

- *Its endianness*: the hardware accelerators generated by *Bambu* internally adopt little endianness; if one or more parameters have different endianness, the final hardware accelerator must also include the module necessary to correctly manage this difference.

Finally, as last parsing step, *Bambu* reads by means of a GCC plugin the C implementation of the interfaces to be synthesized in hardware. After that all the necessary information has been collected, the rest of the High Level Synthesis flow can be applied. When *Bambu* targets the TASTE FPGA architecture, it assumes that the target board is the GR-CPCI-XC4V board [18] which is the currently only board supported by the rest of the TASTE framework. If the support to other boards will be added in the TASTE framework, their support can be easily integrated in *Bambu* by means of XML files. The target frequency is set to 100MHz which is the operating frequency of the APB bus to which the Hardware accelerators are connected.

The outcome of the High Level Synthesis design flow targeting the TASTE FPGA architecture is:

- a VHDL file containing the description of the top architecture to be implemented on the FPGA, i.e., the PCI target and the AMBA bus;

- a VHDL file containing the structural descriptions of all the synthesized C functions.

The adoption of VHDL as output language has been chosen to facilitate the integration of different components (GRLIB IP Cores are written in VHDL), but has required the inclusion of the backend for this language in *Bambu*, since this can previously only generate Verilog code.

Because of the adoption of the ASN.1 standard in the TASTE framework and because of the interaction with the software driver, the generated components for the synthesized functions differ from the components generated by *Bambu* with the default High Level Synthesis flow. The structure of the hardware accelerators generated for the TASTE FPGA architecture is shown in Figure 6. Each accelerator is composed of:

- *C Function Implementation*, i.e., the module actually implementing the C function; this module corresponds to the outcome of *Bambu* when it does not target the TASTE FPGA architecture; it is characterized by a minimal interface which allows the exchanging of data between the module and the rest of the system.

- *Control Register*: it allows to the software driver to control the status of the accelerator.

- *Local Memory*: it is used to store the input and the output aggregate parameters. The overall maximum size of the parameters is limited by the address space assigned to each HW accelerator; the APB protocol limits it to 4KB.

- *IN Registers*: they store the input scalar parameters of the function. These registers, directly connected to the C Function Implementation input ports, are memory mapped so that the software driver can write parameters inside them.

- *OUT Registers*: they store the output scalar parameters of the function. These registers, directly connected to the C Function Implementation output ports, are memory mapped so that the software driver can read parameters from them.

- *Address Translator*: the hardware accelerators implements a local memory address space which is different from the global memory address space adopted in the whole TASTE FPGA architecture; this module performs the translation from an address space to the other.

- *Data Restructuring*: it manages the eventual different data representation between hardware accelerators and software components. These differences can consist of:

  – different endianness between the transferred parameters and the hardware accelerators internal data (they always adopt the little endianness);

  – different padding size in the structure parameters.

It is worth noting that with respect to the components generated by the default High Level Synthesis flow, the component generated for the TASTE FPGA architecture includes several additional modules. Most of these modules are dedicated to the correct management of parameters and represent the overhead of integrating hardware accelerators in the systems targeted by the TASTE framework.

4. Case Study

In this section a case study of the application of the proposed design flow to a real world specification is proposed. The considered specification is the *CCSDS 122.0-B-1 Recommended Standard* [21], which is a lossless to lossy image compression standard. It has been designed for the compression of two-dimensional 16-bit grayscale images produced by payload instruments and it is suitable for use on-board spacecraft because of its low complexity and its reduced intermediate

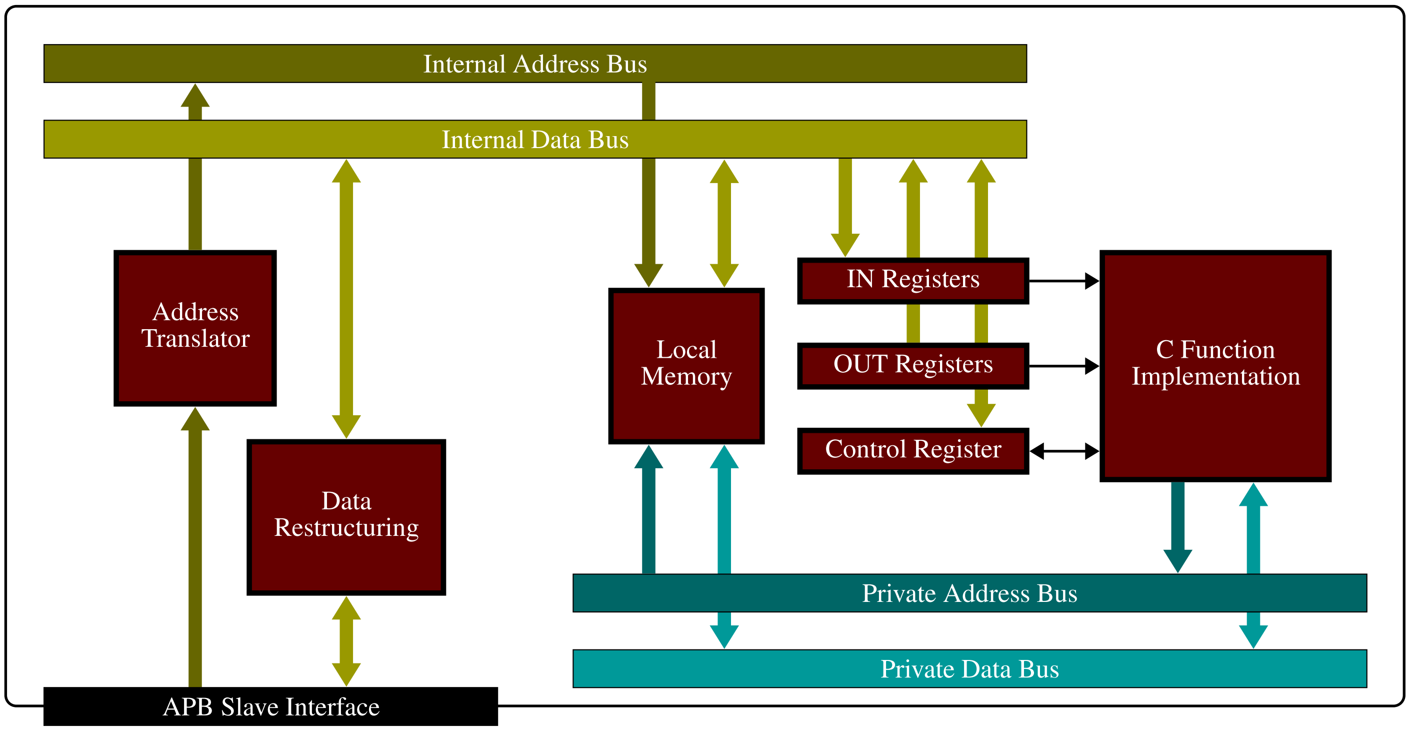

Figure 6. The structure of a hardware accelerator designed to be integrated in the TASTE FPGA architecture.

buffers. The Recommendation includes both a lossy and a lossless compression algorithm which rely on integer and floating point Discrete Wavelet Transforms (DWTs). The floating point DWT has better compression effectiveness at low bit rate, but provides only lossy compression. On the contrary the integer DWT can provide lossless compression and has reduced implementation complexity since it requires only integer computation.

The implementation of this specification considered for this case study is the *CCSDS Image Data Compression Implementation* [22] developed by the University of Nebraska-Lincoln. Despite the fact that the High Level Synthesis helps the designer in porting a complex C application to an architecture which includes a FPGA, there are still some activities required to the designer. Given a legacy C implementation such as the CCSDS image compression, the steps that a designer has to perform in order to design the final system by means of the TASTE framework are the following:

1. *Application Decomposition*: the designer must decompose the application in functions so that the parts of the application that have to be executed by hardware accelerators are separated from the rest. In the case study this requires to restructure the implementation of the compression algorithm, since the computation part and the file writing part are mixed (i.e., the application writes pixel data immediately after their computation). The separation of the computation part of the application from the input/output part is a common approach which has to be applied in most of the design flow for heterogeneous systems. In the following, the computation kernel of the compression algorithm will be identified as `compress`.

2. *Algorithm Customization*: the possibility of configure at run-time the algorithm must be removed if this will not be actually exploited. The CCSDS image compression implementation allows to apply both the integer DWT and the floating point DWT compression. If only one of them will be actually exploited on the target system, the other one can be removed reducing the size of the generated hardware accelerator. In a similar way, all the parameters of the algorithm which will be fixed must be replaced with the corresponding constants. For example in a typical scenario of application of image compression in space system, the resolution of the image coming from sensors is fixed and this information can be embedded in the implementation.

3. *Kernels Cleaning*: all the parts of the kernel functions which are not relevant must be removed from the implementation since they can increase the hardware resource usage or they can make the synthesis impossible. In case of the CCSDS image compression the error management routines and the routines for collecting profiling information data have been removed from the implementation of `compress`.

4. *Exchanged Data Identification*: the designer has to identify which are the data exchanged between the functions mapped on the FPGA and the rest of the application. The legacy implementation of the CCSDS image compression has functions with a limited number of parameters which use global variables to exchange data. Since the communication in the TASTE framework is based on the message passing model (i.e., the data exchanged among different functions must be explicitly listed in the interfaces) and not on shared memory, the signature of `compress` has to be modified to explicitly include all the necessary parameters. Moreover, the size of these parameters must be known: the size of arrays pointed by pointers must have an upper bound known at design time. Note that the *Algorithm Customization* and the *Kernels Cleaning* can significantly reduce the amount of data that have to be exchanged. In case of the CCSDS image compressor these data are the input image, the compressed image and a limited set of configuration parameters.

However, because of the limited address space which can be assigned to each hardware accelerator for data transfers, the input and output images have to be split in tiles. Moreover, for evaluating the size of the output image the worst compression ratio with the selected parameters has to be considered.

5. *Data View Building*: the ASN.1 *Data View* has to be enriched with the data types used in the kernels interfaces. In the considered case study the only data types to be added are the arrays of fixed size which are used to store the tiles of input and output images.

6. *Interface View Building*: the designer builds the *Interface View* by means of the *Interface View editor*. In a typical scenario of usage of a FPGA like the CCSDS image compression, there is at least one function which manages the input/output and one function which contains the kernels of the application. The designer has to specify VHDL as implementation language of the latter and has to provide the C source code file containing its implementation (in the considered example the C source code file must include the implementation of `compress`). Next the designer has to add `compress` as a *provided interface* of this function specifying the input and output parameters using the types defined in the *Data View*.

7. *Interfaces Generation*: the TASTE framework generates automatically the C signatures of the interfaces. In case of the `compress` function it will generate a driver exposing a function with the same parameters.

8. *C Source Code Modification*: the source code of the functions not mapped on the FPGA has to be modified to call the hardware components by means of the generated drivers. In the example the call to `compress` has to be replaced with the call to the corresponding driver.

9. *Deployment View Building*: the designer has to build the *Deployment View* by means of the TASTE graphical tool. The FPGA board is considered part of the processor board, so only this has to be instantiated and both the functions (i.e., the one containing the kernels and the one containing the rest of the application) have to be assigned to it.

10. *System Building*: from this point on, most of the process is directly performed by the TASTE framework: *Bambu* generates the hardware accelerator while the other tools build the rest of the system.

The development process just presented is composed of several steps which involve interaction with the developer. However, some of these steps (i.e., 5,6,7,9,10) are required by the TASTE design flow independently from the considered target system, so they have always to be executed. Other steps (i.e., 1,4,8) have always to be executed when the target system is composed of more than one processing element and the application is split and assigned to some of them. Finally, steps 2 and 3 are the ones which are more related to the generation of the hardware accelerators, but reducing the code size of the computation kernels can positively impact on the overall system even when they are not assigned to a FPGA.

It is worth noting that, despite the potentially increased effort required to application developer to exploit High Level Synthesis, all the activities required in the presented design flow only involve the usage of the TASTE graphical tools and the manipulation of the C source code of the applications: the usage of HDL languages and hardware design techniques is completely transparent to the designer so that their knowledge is not required.

5. CONCLUSIONS

In this paper the integration of High Level Synthesis (*Bambu*) in a design flow targeting real time safety-critical embedded systems (TASTE) has been proposed. The use of explicit data transfers and of ASN.1 has required to extend the High Level Synthesis flow to correctly manage inputs and outputs of the generated hardware accelerators. The presented case study shows that the introduction of hardware accelerators in the system does not increase the designer effort in a significant way and does not require knowledge of HDL languages, so that they can be easily exploited also by software developers.

Potential future works concern the optimization of the communication of the hardware accelerators with the rest of the system (i.e., the improvement of the software drivers and the removal of the APB bus) and the integration of Hardware/Software codesign methodologies in the TASTE framework to automatize the selection of the functions to be synthesized as hardware accelerators.

REFERENCES

[1] D. Ludtke, K. Westerdorff, K. Stohlmann, A. Borner, O. Maibaum, T. Peng, B. Weps, G. Fey, and A. Gerndt, “OBC-NG: Towards a reconfigurable on-board computing architecture for spacecraft,” in *Aerospace Conference, 2014 IEEE*, March 2014, pp. 1–13.

[2] N. Wulf, A. George, and A. Gordon-Ross, “A framework to analyze, compare, and optimize high-performance, on-board processing systems,” in *Aerospace Conference, 2012 IEEE*, March 2012, pp. 1–14.

[3] ——, “Memory-aware optimization of FPGA-based space systems,” in *Aerospace Conference, 2015 IEEE*, March 2015, pp. 1–13.

[4] J. Greco, G. Cieslewski, A. Jacobs, I. Troxel, and A. George, “Hardware/software interface for high-performance space computing with FPGA coprocessors,” in *Aerospace Conference, 2006 IEEE*, 2006, pp. 10 pp.–.

[5] M. Deshmukh, B. Weps, P. Isidro, and A. Gerndt, “Model driven language framework to automate command and data handling code generation,” in *Aerospace Conference, 2015 IEEE*, March 2015, pp. 1–9.

[6] European Space Agency, “TASTE,” http://taste.tuxfamily.org/.

[7] M. Gonzalez Harbour, J. Gutierrez Garcia, J. Palencia Gutierrez, and J. Drake Moyano, “MAST: Modeling and analysis suite for real time applications,” in *Real-Time Systems, 13th Euromicro Conference on, 2001.*, 2001, pp. 125–134.

[8] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, “Cheddar: A Flexible Real Time Scheduling Framework,” *Ada Lett.*, vol. XXIV, no. 4, pp. 1–8, Nov. 2004. [Online]. Available: http://doi.acm.org/10.1145/1046191.1032298

[9] P. Feiler, D. Gluch, and J. Hudak, “The

Architecture Analysis & Design Language (AADL): An Introduction," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2006-TN-011, 2006. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7879

[10] O. Dubuisson and P. Fouquart, *ASN.1: Communication Between Heterogeneous Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[11] P. Coussy, D. Gajski, M. Meredith, and A. Takach, "An Introduction to High-Level Synthesis," *Design Test of Computers, IEEE*, vol. 26, no. 4, pp. 8–17, July 2009.

[12] R. K. Brayton, A. L. Sangiovanni-Vincentelli, C. T. McMullen, and G. D. Hachtel, *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA, USA: Kluwer Academic Publishers, 1984.

[13] C. Pilato and F. Ferrandi, "Bambu: A modular framework for the high level synthesis of memory-intensive applications," in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, Sept 2013, pp. 1–4.

[14] Politecnico di Milano, "Panda framework," http://panda.dei.polimi.it.

[15] "GNU Compiler Collection (GCC)," http://gcc.gnu.org.

[16] G. De Micheli, R. Ernst, and W. Wolf, Eds., *Readings in Hardware/Software Co-design*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[17] "OpenGEODE," http://opengeode.net.

[18] Cobham Gaisler AB, "GR-CPCI-XC4V Leon CompactPCI Deveopment board," http://www.gaisler.com.

[19] "ARM Advanced Microcontroller Bus Architecture (AMBA)," http://www.arm.com/products/system-ip/amba-specifications.php.

[20] Cobham Gaisler AB, "GRLIB Ip Library," http://www.gaisler.com.

[21] Consultative Committee for Space Data Systems (CCSDS), "CCSDS 122.0-B-1," http://public.ccsds.org/publications/archive/122x0b1c3.pdf.

[22] University of Nebraska-Lincoln, "CCSDS Image Data Compression Implementation," hyperspectral.unl.edu.

BIOGRAPHY

Marco Lattuada *received the Master and the PhD degrees in Computer Engineering from Politecnico di Milano, Italy, in 2006 and 2010 respectively. In 2012 and in 2013 he was visiting researcher at European Space Agency. Since 2010, he has been temporary researcher and lecturer at Dipartimento di Elettronica, Informazione e Bioingegneria of Politecnico di Milano. His research interests include methodologies for embedded system design and in particular High Level Synthesis, performance estimation and automatic generation of code for multiprocessor heterogeneous architectures.*

Fabrizio Ferrandi *received his Laurea (cum laude) in Electronic Engineering in 1992 and the Ph.D. degree in Information and Automation Engineering (Computer Engineering) from the Politecnico di Milano, Italy, in 1997. He has been an Assistant Professor at the Politecnico di Milano, until 2002. Currently, he is an Associate Professor at the Dipartimento di Elettronica, Informazione e Bioingegneria of the Politecnico di Milano. His research interests include synthesis, verification simulation and testing of digital circuits and systems. Fabrizio Ferrandi is a Member of IEEE, of the IEEE Computer Society and of the Test Technology Technical Committee.*

Maxime Perrotin *received his Diploma in Engineering from the Conservatoire National des Arts et Mtiers in 2001 (Diplme d'Ingnieur). After working in the space industry he joined the European Space Agency as technical officer. He took the lead of the R&D activities related to formal methods, modelling and code generation, and is providing engineering support to satellite development, with a special focus on in-orbit demonstration missions, such as formation flying systems.*