J Real-Time Image Proc (2014) 9:251–262
DOI 10.1007/s11554-013-0326-5

SPECIAL ISSUE

High-level dataflow design of signal processing systems for reconfigurable and multicore heterogeneous platforms

Endri Bezati · Richard Thavot · Ghislain Roquier · Marco Mattavelli

Received: 30 March 2012 / Accepted: 21 January 2013 / Published online: 13 February 2013

Abstract The potential computational power of today multicore processors has drastically improved compared to the single processor architecture. Since the trend of increasing the processor frequency is almost over, the competition for increased performance has moved on the number of cores. Consequently, the fundamental feature of system designs and their associated design flows and tools need to change, so that, to support the scalable parallelism and the design portability. The same feature can be exploited to design reconfigurable hardware, such as FPGAs, which leads to rethink the mapping of sequential algorithms to HDL. The sequential programming paradigm, widely used for programming single processor systems, does not naturally provide explicit or implicit forms of scalable parallelism. Conversely, dataflow programming is an approach that naturally provides parallelism and the potential to unify SW and HDL designs on heterogeneous platforms. This study describes a dataflow-based design methodology aiming at a unified co-design and co-synthesis of heterogeneous systems. Experimental results on the implementation of a JPEG codec and a MPEG 4 SP decoder on heterogeneous platforms demonstrate the flexibility and capabilities of this design approach.

E. Bezati (✉) · R. Thavot · G. Roquier · M. Mattavelli
EPFL SCI-STI-MM, Lausanne, Switzerland
e-mail: endri.bezati@epfl.ch

R. Thavot
e-mail: richard.thavot@epfl.ch

G. Roquier
e-mail: ghislain.roquier@epfl.ch

M. Mattavelli
e-mail: marco.mattavelli@epfl.ch

Keywords Dataflow · FPGA · HW/SW co-design · Co-synthesis · Multicore computing · Openforge · ORCC · RVC-CAL

1 Introduction

Since latest generation of platforms includes more and more clusters of multicore processors and programmable logic units, portable parallelism, for both SW and reconfigurable HW, is certainly a key requirement for systems implementations that aim at efficiently running on such platforms. However, the traditional sequential specification is certainly not the appropriate design abstraction for the efficient usage of the underneath processing capabilities. In addition, the existing processor software and HDL designs, legacy of several years of development, are not the most appropriate starting point to program such platforms [1]. Needless to say, such legacy specification can lead to an inappropriate design abstraction when targeting heterogenous platforms. Parallelizing sequential code is an extremely time-consuming and error-prone process. In addition, it needs to be redone every time a design is implemented on a platform that possesses a different level of processing parallelism. The same considerations can be made for the portability of the already parallelized code, since the design process needs to be started from scratch when targeting another platform. An additional feature that becomes necessary on heterogeneous and parallel platforms, which are less significant for sequential processors, is a systematic exploration of the design space. This is due to the combinatorial explosion of design options. It requires the design to be sufficiently modular and portable, without any manual rewriting. As a matter of fact, manual rewriting practically reduces design space exploration to extremely limited

exploration ranges for the amount of design resources needed in the case of complex application designs.

In this direction, a dataflow design and programming paradigm as well as a design flow with the corresponding tools has been developed. Dataflow naturally exposes the potential parallelism of applications, which can be used to distribute computations according to the available parallelism supported by the platforms. This study in particular describes a methodology for hardware–software co-design. This methodology uses as an input a high-level dataflow program that directly yields implementations for heterogeneous parallel platforms. This methodology, as is proven by the experimental results given on this study, provides performance scalability and rapid prototyping by the means of portability. The study also presents how this dataflow-based design flow can also rely on the capability of the platform design environment for code profiling, profile-guided refactoring, application-to-architecture mapping, and automatic code synthesis of dataflow applications on heterogeneous architectures composed of processors and FPGAs.

The article is structured as follows: Sect. 2 summarizes the most relevant related work on co-synthesis and co-design for heterogeneous platforms. Section 3.1 introduces the proposed methodology by explaining the different stages that enable the automatic synthesis of dataflow programs onto the target platforms. Section 4.1 reminds the fundamental concepts of the dataflow programming approach developed, the advantages versus the traditional sequential programming model and the advantages that the features of the formal CAL dataflow language used for the application specification provides in terms of abstractions and optimization possibilities. The section also presents the basic components of the abstraction used to represent the architectural model of the target platforms. Section 5 describes the main dimensions of the feasible design space exploration stages. Section 6 describes in details the toolchain implementing the complete design flow. Section 7 presents some experimental results consisting of the implementation of well-known video codecs systems. Finally, Sect. 8 concludes the paper by discussing the advantages of the approach and some of the remaining challenges for further improvements as well as some perspectives of future work and further extensions for the support of other design options.

2 Related work

The concepts and fundamentals of HW–SW co-design have been introduced since the early nineties [2, 3]. Model-based design was proposed to raise the level of abstraction when designing digital processing systems. High-level models provide useful abstractions, such as platform independency, to ease analysis tasks.

Prior research related to model-based design using data- and control-dominated models and a combination of both is the subject of a wide literature. Essentially, the various methods proposed mainly differ by the model used and their underlying model of computation (MoC). It mainly consists of a trade between the expressive power and analyzability properties of the model. Expressiveness defines the set of applications that can be represented in that model. The more a model can express, the harder it becomes to analyze its inner properties, such as scheduling, bounded memory usage, liveness and so on.

For brevity reasons and without claiming to be exhaustive, we can mention the POLIS [4], a Codesign Finite State Machine (CFSM)-based framework which enables FSMs to communicate asynchronously. Such model has limited expressiveness, but a useful analyzability property. Indeed, the more a model can express, the harder it becomes to analyze its features such as liveness or bounded execution. By contrast to control-dominated models, when dealing with stream processing algorithms, it is preferable to use data-dominated models such as communicating sequential processes (CSP) [5], dataflow [6] or Kahn process networks (KPN) [7].

The INRIA's AAA methodology and its associated tool SynDEx are examples of such a design approach. The AAA is based on a restricted dataflow model that supports conditional statements using the hierarchy of the dataflow graph. The SynDEx model is deliberately restricted to enforce real-time constraints; consequently, it is not adapted to model more general class of applications. Moreover, the High-Level Synthesis (HLS) that turns the SynDEx model to an HDL implementation is not clearly defined, and results on that are not extensively reported in the literature.

A KPN-based approach called Compaan/Laura from Leiden University [8] is using a subset of MATLAB code to model applications. The HW back-end, Laura, transform the KPN model expressed by MATLAB code to VHDL. KPN-based models are much more expressive than more restricted MoC and can cover a much broader set of applications. However, since analyzability is, in most of the cases, inversely related to the expressiveness of a MoC, it is somehow difficult to figure out the ability to produce the corresponding KPN models of more complex applications written in MATLAB.

PeaCE from the Seoul National University can be considered a midway approach between synchronous dataflow (SDF) and FSM [9]. This model raises the level of expressiveness, by enabling the usage of control structures using FSM inside SDF nodes and vice versa. However, whereas PeaCE generates code for composing blocks of the model for both SW and HW components, it lacks the automatic code generation of the code inside the processing

blocks. In this case, the user should define HW–SW blocks in a later stage, an operation which can result resource consuming and error prone.

Another interesting approach in the field is SystemCoDesigner from the University of Erlangen-Nuremberg [10]. SystemCoDesigner is an actor-oriented approach using a high-level language, built on top of SystemC, named SysteMoC. It generates HW–SW SoC with automatic design space exploration techniques. The model is translated into behavioral SystemC model as a starting point for HW and SW synthesis. The HW synthesis is delegated to a commercial tool, viz., Forte's Cynthesizer, which generates RTL code from their SystemC intermediate model. In essence, the approach presented in the study is remarkably similar to the SystemCoDesigner design flow. A comparison highlighting the differences between the two approaches is provided in Sect. 7.

The system design approach presented in this study presents several innovative aspects. Those are built around and on top of the design flow tools developed through the years after the initial formal specification of the CAL language [11]. For brevity, only some essential innovations versus the state of the art related to SW/HW co-design portability are fully developed in the paper. As mentioned, the first innovation, which is also the base of all other steps of the design flow, is the usage of a formal high-level dataflow language. This language unifies the description of several dataflow classes and their associated MoC. Thus, the formal dataflow language can be used for both SW (single-core and multicore) and HW component syntheses. In fact this formal language has a domain-specific property and originates several techniques capable of removing most of the scheduling overhead that is theoretically needed for implementing dynamic dataflow programs [12–14].

A second fundamental innovation, which has been demonstrated, is that a standardized subset of the dataflow language, does not only offers scalability from a single core to multicore but is also synthesizable to HW. To achieve that, a specific modeling of the target platform has been given, a dataflow profiling tool has been created and different optimizations have been developed for the efficient usage of a targeted platform. For the metrics and heuristics driving the design space exploration, the reader can refer to [15, 16].

3 Methodology

3.1 Proposed design flow

The proposed methodology and associated design flow essentially consist of six steps as illustrated in Fig. 1. The initial step is the specification of the application and the

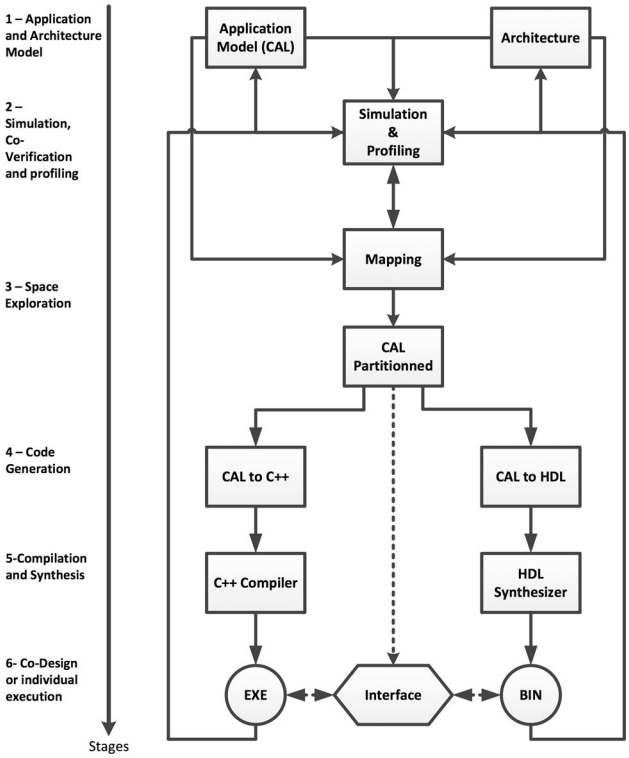

Fig. 1 Overview of the CAL dataflow design flow steps

definition of the platform architecture. The application specification expressed using the CAL dataflow programming language is entirely agnostic of the target platform. In other words, the specification is the same for partitions that finally will run on a single core processor, multicore or programmable logic unit (FPGA).

In a second step, the application is functionally validated by a set of behavioral simulations employing meaningful (for the specific application) input stimuli. This process can also provide inputs for the third stage in terms of profiling measures extracted during the simulations. Architecture-aware simulations (using the mapping of the third step) can also be done. They may be used to discover bottlenecks early in the design process and to refactor the application and/or the architecture mapping accordingly.

The third step consists of the design space exploration stage. In broad outline, this stage provides the mapping of the algorithm onto the architecture. The mapping is obtained by transforming (viz., scheduling and partitioning) the application according to the architecture. Profiling metrics (first-order approximations from simulations, cycle accurate from the platform execution, etc.) is used to find the close-to-optimal mapping according to a desired criteria.

The fourth step consists of taking each partition of the mapping (a subset of the application associated to a processing element of the architecture) including the interfaces

necessary for the platform component interconnections and synthesize them by the corresponding implementation source code generator (C/C++ for SW, HDL for HW).

Then the generated implementation source code for each SW or HW processing element of the platform is compiled using the native platform compilers. The execution of the application is the final validation step. Besides, this execution can also be part of an iterative space exploration process that uses different profiling measures given by external profiling tools.

3.2 The co-design environment

Section 6 describes in detail the co-design tool used in the design flow. Those tools sit on the top of two source code generators, ORCC and OpenForge. The co-design tool is responsible for the partitioning of a dataflow program only to Software (multicore, many-core or many systems), only to Hardware (interconnection of different hardware) or an interconnection between Software and Hardware. Its input is a dataflow application and an architecture description. The following steps are performed by the co-design tool:

- the mapping of the dataflow program according to the architecture (automatically or user-defined),
- the co-synthesis to generate software source code and synthesizable hardware,
- the synthesis of the inter-partition communications as well as their interfaces.

The dataflow application is transformed into an *implementation* model according to the platform architecture. The transformation consists of inserting additional nodes that represent inter-partition communications, according to the PE interconnections. Such transformation introduces distinct nodes in the dataflow program, which has the objective of encapsulating at a later stage the multiplexing and demultiplexing of tokens as well as the implementation of the corresponding interfaces between partitions. The multiplexing stage schedules the communications between actors (the processing elements of a CAL dataflow network) that are allocated on a different platform partitions. The demultiplexing is the reverse process. The inter-partition communication stage is also illustrated in Fig. 2, where gray blocks are inserted to represent the connections between partitions. For instance, the orange partition has two incoming channels from the blue partitions. Those FIFOs are going to be multiplexed on the blue partition and demultiplexed on the orange one. Then, the communication with other components is done via I/O with an associated communication protocol. Interfaces are introduced during the synthesis stage and are supported by libraries according to the nature of the interconnection component present on the processing platform. Currently, PCIe, Ethernet and UART have been tested, validate and are currently supported in the co-design tool.

4 Application and architecture models

Architecture modeling, in the proposed design flow, is based on the model developed in Ref. [17]. The platform architecture is modeled by an undirected graph where each node represents an operator [a processing element such as a CPU or an FPGA in the terms of [17]) or a communication element (bus, memories, etc...)]. An edge represents an interconnection interpreted as a transfer of data from/to operator and to/from communication elements. The architecture description is serialized into an IP-XACT description, an XML format developed by the SPIRIT Consortium, which defines the description of electronic components. The architecture description is hierarchical and enables us to specify architectures with different levels of granularity. For instance, a multicore processor can be represented as an atomic node or hierarchically by exposing lower level details, in which processing elements and memories become in turn atomic nodes. Figure 3 depicts as an example an architectural description of the QorIQ P4080 platform from Freescale. Such platform includes eight PowerPC e500mc cores with three-level of cache hierarchy, L1 and L2 as a private cache and L3 as a shared one.

4.1 Application model: dataflow with firing

Dataflow programming models have a rich history dating back to at least the early 1970s, including seminal work by Dennis [6] and Kahn [7]. For the purposes of this work, a dataflow program is a directed graph in which nodes represent computational units (called actors), while edges represent connections between actors used to communicate sequences of data packets (tokens). Several models that describe a dataflow application have been introduced in literature [7, 18–20], often referred to as different dataflow models of computation. Each Moc gives the behavior of an actor. Also, it indicated which techniques should be used for the actor execution, which results in different trade-offs between expressiveness and implementation efficiency.

One common characteristic across all these dataflow models is that individual actors encapsulate their own state, and thus do not share memory with one another. Instead, actors communicate with each other exclusively by sending and receiving tokens along the channels connecting them. The resulting absence of race conditions [21] makes the behavior of dataflow programs more robust to different execution policies, whether those be truly parallel or some interleaving of the individual actors.

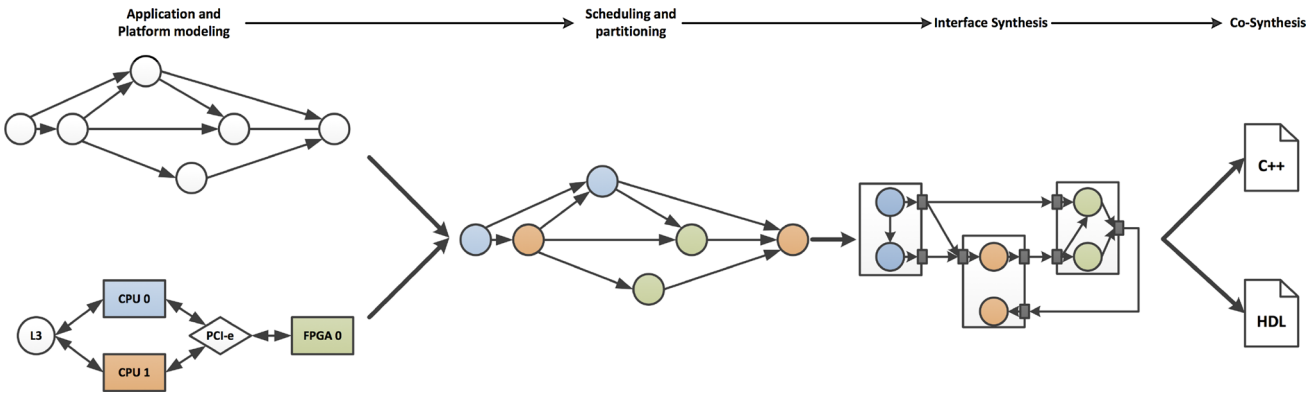

Fig. 2 The proposed design flow. From the application and platform modeling to co-synthesis

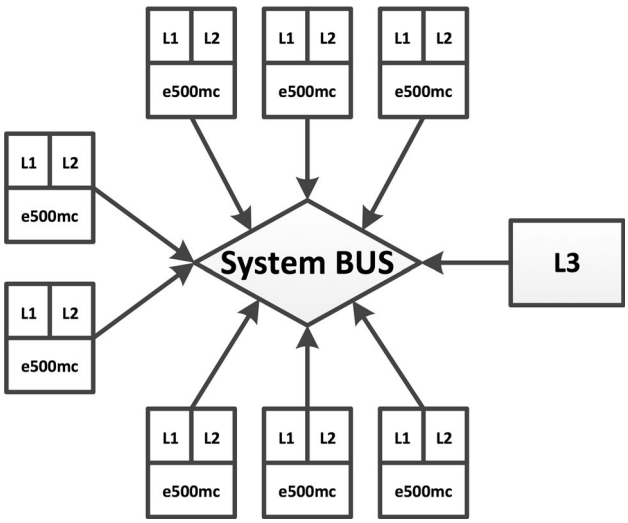

Fig. 3 An example of architectural description specifying a Freescale QorIQ P4080 platform

4.1.1 Dataflow process networks

The design flow and co-design component described in this work are based on a dataflow model of computation called dataflow process networks (DPN) [20]. In addition to the properties of dataflow mentioned above, each DPN actor executes by performing a sequence of discrete computational steps, called *firings*. In each such step, an actor may (a) consume a finite number of input tokens, (b) produce a finite number of output tokens, and (c) modify its internal state, if it has any.

The behavior of a DPN actor is specified as a pair of firing rule and firing function. The firing rule determines when the actor may fire, by describing the input sequences and actor state that need to be present for the actor to be able to make a step, i.e., for it to be eligible. The firing function determines, for each input sequence/state combination for which the actor is eligible according to the firing rule, the output tokens produced at that step and, if applicable, the new actor state. In general, an actor may be non-deterministic, which means that the firing function may yield more than one combination of output and next state for any given enabled actor.

4.1.2 The RMC dataflow language: RVC-CAL

The CAL actor language [11] directly captures the description of DPN actors as making discrete steps triggered by conditions on actor state and available input tokens. In it, an actor is defined as a set of actions, each action capturing a part of the firing rule of an actor along with the part of the firing function that pertains to the input/state combinations enabled by that partial rule.

An action is enabled according to its input patterns and guard expressions. Input patterns define the amount of data that are required on the input sequences, whereas guards are boolean expressions on the current state and/or on input sequences that need to be satisfied for enabling the execution of an action.

CAL is a domain-specific language designed to support code analysis techniques due to its DPN MoC nature. Thus, a set of directives is provided to a compiler code so that it can apply numerous optimizations during code synthesis. An example of such optimizations is the static scheduling of actors (a series of firings that can be executed without testing their firing rules) for some network partitions at compile time [12, 14, 22, 23]. As already mentioned, CAL can also be directly synthesized to software and hardware [21, 24, 25]. Within MPEG RMC, a subset of the more general CAL language, called RVC-CAL, has been standardized by ISO/IEC MPEG [26].

As an example, Fig. 4 depicts a CAL actor which is part of the JPEG encoder presented in Sect. 7.

5 Design space exploration

CAL language, by means of its operators, expresses the intrinsic parallelism of algorithms in the dataflow programs

```
import jpeg.encoder.common.Tables.QT;

actor Quantization() int In ⇒ int Out:
  int blk_type := 0;

  function div_round_nearest(int a, int b) → int :
    if a >= 0 then
      (a + b / 2) / b
    else
      (a - b / 2) / b
    end
  end

  action In:[val] repeat 64 ⇒ Out:[data] repeat 64
  var
    int(size=24) data[64], int qt
  do
    foreach int i in 0 .. 63 do
      qt := QT[blk_type >> 2][i];
      data[i] := div_round_nearest(val[i], qt);
    end
    blk_type := (blk_type + 1) mod 6;
  end
end
```

Fig. 4 The Quantization (*Q*) actor in the JPEG encoder written in RVC-Cal

at different granularity levels. However, when dealing with highly complex signal processing and communication applications, which are usually specified by several tenths of thousands of source code, it is not always easy to understand if the exposed degree of parallelism is appropriate or not for the chosen processing platform. In addition, the combinatorial explosion of the possible partitioning and scheduling options that a dataflow network offers, when its executed and mapped on a platform composed of several processing units, opens a large design space for the final implementation. Such design space is certainly an opportunity for searching the best implementation following a given criteria.

This is the reason why the design of a dataflow model, without any tool assisted or automatic design space exploration, may results in bad quality implementations. Indeed, nowadays, the design space exploration from dataflow abstraction is in itself a very active and wide subject of research. In this study, only some key elements and results show how such fundamental stage can be naturally integrated in the design flow.

More results can be found in [15] and related references. The following sections introduce two crucial dimensions of the space exploration that are more related to the definition of the co-design components: the scheduling/partitioning problem and the satisfaction of implementation constraints. Section 5.1 introduces the scheduling/partitioning problem. Section 5.2 describes an approach developed with the purpose of reducing the scheduling overhead. Finally, Sect. 5.3 describes the approach capable of minimizing the memory size requirement [27] of achievable implementations.

5.1 Mapping

The mapping stage consists in partitioning and scheduling the dataflow program according to the architecture. Partitioning consists of assigning a processing element (PE) for each actor of the dataflow program. Scheduling consists of ordering the execution of actors assigned to a given partition. In general, the number of processing elements is often lower than the number of dataflow actors. This implies that several actors can be assigned to a single PE, thus requiring the definition of a sequencer to schedule them in time. The problem of scheduling and partitioning a dataflow graph onto architecture with multiple PEs is NP-complete [28]. Therefore, heuristics with polynomial-time complexity are widely used when dealing with large-scale dataflow graphs. The scheduling/partitioning is illustrated in the second step of Fig. 2, where the color of actors on the dataflow program corresponds to the PE assignment.

There are several possible strategies for scheduling and partitioning a dataflow graph. Using the taxonomy of [29], the *static assignment* strategy consists of a partitioning defined at compile time, whereas scheduling is done at runtime. Indeed, since CAL belongs to the DPN MoC, actors have to be scheduled at runtime in the general case.

In the proposed methodology, the static assignment strategy is chosen for scheduling/partitioning dataflow programs. A heuristic-based technique is applied to the execution trace, which represents the execution of the dataflow graph, to determine the partitioning using the metrics extracted during the profiling stage. More precisely, an execution trace is a directed acyclic graph (DAG), obtained during the simulation of the application, where nodes represent *executed actions*, and edges represent *dependencies* on a state variable, a guard, a port or a token. More details about traced-based heuristics for yielding efficient partitioning and scheduling are given in Ref. [30].

5.2 Static scheduling

Once the partitioning is determined by the mapping, actors assigned on a given PE are scheduled dynamically, since all of them are in general assumed to belong to the DPN

MoC. This may result in an unnecessary runtime overhead, since actors are scheduled dynamically at runtime. However, scheduling statically (a subset of) those actors is sometimes possible when they belong to more restricted MoCs, viz., SDF and CSDF MoCs, which enable one to reduce the unnecessary overhead. Several approaches have been proposed to statically schedule CAL program [31]. Initial version of detecting and scheduling sub-partitions classified as SSRs has been integrated in the dataflow toolset known as ORCC (see Sect. ~6.1). A method to classify actors (MoC of an actor) as well as a static scheduler that merges SDF/CSDF into coarse-grain actors has been developed in ORCC [23].

5.3 Buffer dimensioning

According to the DPN MoC, actors communicate through unbounded channels (it is a key property to guarantee liveness). Nevertheless, channel capacity must be bounded in the real world applications. Unfortunately, bounding channels may introduce *artificial* deadlocks. A buffer dimensioning stage has been developed with the objective of minimizing the channel capacity, without introducing deadlocks. To this end, a simulation-based analysis is conducted where channels capacities are determined by updating their status after each firing. Statistically, representative input stimuli representing typical usage scenarios are used to drive the simulation analysis. A demand-driven scheduling [32] strategy for a dataflow network, known to minimize the buffer size requirement is used in the dimensioning procedure. It consists of selecting actors from the sinks (actors without output channels) to the sources (actor without input channels) via predecessors until the first one can fire. Once an actor has fired, the scheduler restarts from the sinks.

6 Code generation

The described design flow aims at generating both software components and hardware descriptions for the different partitions. On one hand, the software synthesis consists of generating pieces of code for the different application partitions assigned on processor-based PEs. C and C++ are the target language, since they can be ported on numerous architectures. On the other hand, the hardware synthesis has the objective of automatically generating the HDL code ready for RTL synthesis from the different application partitions assigned to FPGAs.

The following section describes the code generation tools which support the code generation of CAL programs onto different platforms. ORCC and OpenForge are presented in Fig. 5. Their input is a partition of the dataflow program. The input dataflow program is serialized in an XDF network, an XML format for the definition of hierarchical dataflow graphs. The outputs of the toolchains are both implementation software components and hardware descriptions which are automatically generated and ready for compilation or RTL synthesis.

6.1 ORCC

The Open RVC-CAL Compiler or ORCC [33] is a compiler infrastructure dedicated to the CAL language of which the paper authors are contributors to the main effort provided by the INSA of Rennes. ORCC can be seen as a collection of eclipse plug-ins that enable to synthesize code from CAL (see Fig. 5). The front-end consists in parsing the dataflow program and translating it to an intermediate representation (IR) in static single assignment (SSA) form. Transformations may be applied depending on the target language and platform. Finally, the back-ends generate code from IR. Several back-ends target various implementation languages (C, C++, LLVM, etc.). For the co-design tool, two back-ends have been developed by the paper authors.

6.1.1 C++ back-end

A first back-end translates the IR into C++ code. A naive implementation of a dataflow program would be to create one thread per actor of the application. However, distributing the application among too many threads results in too much overhead due to too many context switches. Instead, a more appropriate solution that avoids generating too many threads per core has been developed. The execution of actors is done by a single thread, which represents a user-defined scheduler. This scheduler selects the sequence of the actors to be executed. In addition, if desired by the user, the scheduler can create as many threads as existing cores on multicore platform. Finally, the generated code has dependencies to a portable support library. This library enables the application to instantiate actors and FIFOs, to schedule actors at runtime using a user-defined scheduler as well as to support the inter-partition communications by instantiating multiplexers, demultiplexers and I/O interfaces.

6.1.2 XLIM back-end

To generate synthesizable HDL code, a low-level IR called XLIM (XML language-independent model) has been defined for the generation of the input to the HDL synthesis tool. The front-end (OpenForge) that parses and translates CAL to XLIM was used in Ref. [21]. Since then it has been

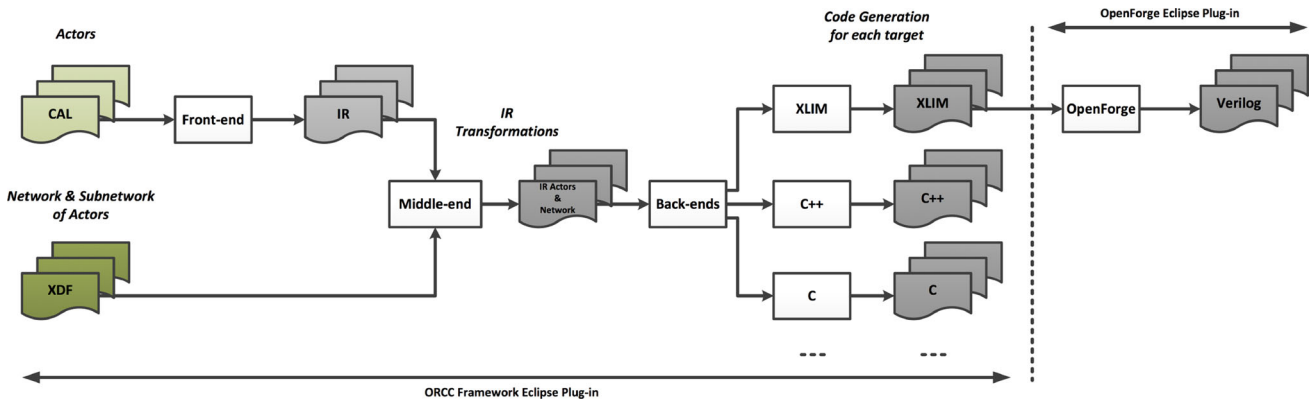

Fig. 5 The ORCC compilation flow and the OpenForge synthesizer

substituted by a new tool. The new XLIM back-end has been developed with the objective of an extended language support and to improve performance on the generated HDL code. This back-end is described in full details in Ref. [34]. Also, it should be mentioned that the XLIM back-end offers the automatic instantiation of different Xilinx IP-Core for different interfaces like Ethernet and PCI-Express.

6.2 OpenForge

OpenForge [35] is a behavioral synthesis tool that translates the XLIM IR into a hardware description expressed in Verilog. Initially it was developed by Xilinx, but since 2009, it has been released to the public domain. Since then, the papers authors have maintained and extended the OpenForge's capabilities to fully support the ISO standard version of CAL. OpenForge turns the XLIM into a web of circuits built from a set of basic operators (arithmetic, logic, flow control, memory accesses and the like). The synthesis stage can also receive directives driving the unrolling of loops, or the insertion of registers to improve the maximal clock rate of the generated circuit. The final result is a synthesizable Verilog file that can be synthesized to RTL for Xilinx FPGAs. The generated Verilog contains the circuit of the actor and exposes asynchronous handshake style interfaces for each of its ports. The generated Verilog actors are connected in the same way as in the XDF network graph using FIFO buffers into complete systems. Finally, the FIFO buffers can be synchronous or asynchronous, thus making the support of multi-clock-domain dataflow designs a feature out of the box.

7 Experiments

Two different image processing models were developed for the experiments: a motion JPEG codec and an MPEG 4 SP decoder. The experiments are separated in three categories: the first one is focusing on the reconfigurable hardware, the second one is on multicore, and finally the third one is highlighting the co-synthesis for heterogeneous platforms.

7.1 Hardware synthesis

This experiment compares the automatically generated HDL code of a CAL application with a handwritten HDL code in terms of throughput and resource occupation. A comparison between two models is given. The first one is a baseline profile JPEG encoder written in CAL, and the second one is a VHDL JPEG encoder from the OpenCores project [36]. Figure 6a depicts the CAL JPEG encoder where actors are at the encoding DCT, quantization and zigzag scan (Q-ZZ), variable-length encoding (Huffman) and the bitstream organizer and writer which generates a 4:2:0 JPEG file (Syntax Writer). As for the VHDL encoder, which is represented in Fig. 6b, the DCT, Q and ZZ are processed in parallel for the luminance and chrominance blocks. Finally, this VHDL encoder uses FIFO for its functional blocks, and this is reason why it was chosen for this experiment.

Both encoders use a Xilinx Virtex 6 FPGA. Table 1 indicates the throughput of both encoders for encoding two images with different resolutions and the throughput of this images for two different clock frequencies. The result of this experiment shows that the handwritten VHDL JPEG encoder is only 1.5 times faster when compared with the automatically generated HDL. The VHDL encoder is faster because of two reasons. Firstly, it uses the FDCT IP core accelerator from Xilinx and secondly, the Y-UV blocks are processed in parallel. The splitting of the Y, U and V components for DCT, Q, ZZ and VLC blocks is a possible optimization for the CAL encoder.

Table 2 compares the resource usage of the VHDL and CAL encoders. The CAL encoder uses 3.65 times less registers, 1.6 times less LUTs pairs and 4 times more of DSP48E1 than the former one. The number of DSP blocks used is negligible because it represents only 1.6 % of the available DSP

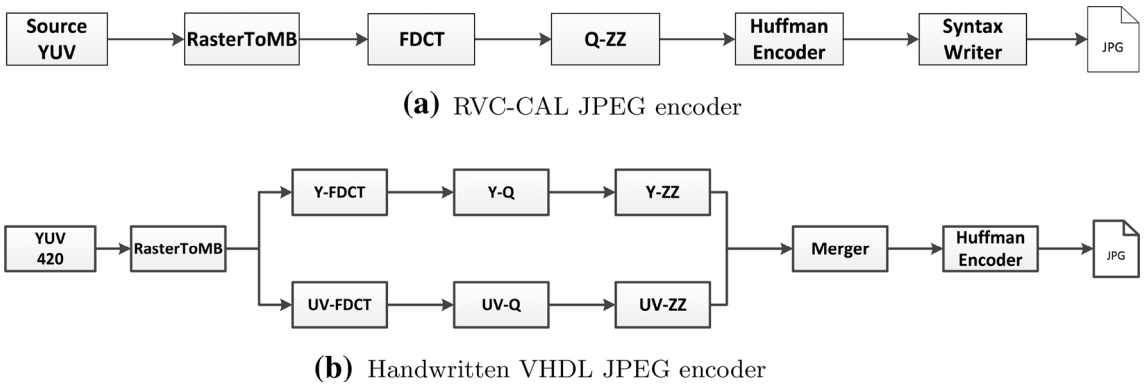

Fig. 6 Description of the RVC-CAL and the handwritten VHDL JPEG encoder

blocks. Thus, it requires less FPGA logic resources than the handwritten VHDL one in the given configuration. The topology of the two designs explains the difference of the results. The handwritten encoder is parallelized according to color components, while the CAL one is not.

Another important comparison between both designs is the number of source code lines. The handwritten VHDL encoder requires 5.4 times more source code lines than the CAL one, while the functionality blocks are the interconnection between them are almost the same. We can also notice that it can encode 4 Full HD images (1920 $\times$ 1080) in less than a second at 80 MHz, and it can encode in real-time 512 $\times$ 512 images, which is decent performance, keeping in mind that no optimization has been performed yet.

To compare the results with the state-of-the-art, we need to extrapolate the results from Ref. [10]. Since JPEG encoding and decoding process are symmetric, we assume that their complexities are almost the same. The encoder is three times faster than the decoder in [10] at 50 MHz. It uses 4 times less LUTs, 2 times less flip-flops and 1.5 times less BRAMs.

7.2 Software synthesis

In the literature, it is possible to find different implementations on the multicore using CAL [25, 37]. The experiment reported here shows the latest designs obtained using the described design flow in terms of speedup and throughput. The execution platforms are a desktop computer with an Intel Core i7-870 processor, with 4 cores at 2.93 GHz, and a Freescale P4080 platform, using a PowerPC e500 processor with 8 cores at 1.2 GHz. In both cases, it is only a single executable that is running up to the four cores. Foreman (QCIF, 300 frames, 200 kbps), crew (4CIF, 300 frames, 1 Mbps) and Stockholm (720p60, 604 frames, 20 Mbps) are the sequences used. The alternative partitions are described in Fig. 7. The blocks represented by a striped background are distributed over different partitions.

Figure 7 represents the MPEG-4 Simple Profile.

Table 3 describes the framerate (in frame per second or fps) of the decoder from 1 to 4 cores. The resulting speedup is illustrated in Table 4 which reveals that it is possible to achieve a significant speedup when more cores are available.

Table 5 reports that the scalability is preserved when changing the resolution of the video format. In terms of speedup factor versus the single-core performance, results are of the same order of magnitude than the ones presented in Ref. [37]. In terms of absolute throughput, we experiment a speedup of four compared to [37] for the P4080 when normalized at the same frequency.

Table 1 JPEG encoder throughput

Type	Resolution	FPGA frequency	
		50 MHz	80 MHz
RVC-CAL HDL	512 $\times$ 512	48 ms	28.9 ms
Generated code	1920 $\times$ 1080	373 ms	223 ms
Handwritten VHDL	512 $\times$ 512	31.2 ms	18.7 ms
	1920 $\times$ 1080	317 ms	190 ms

7.3 Heterogeneous implementation

Finally, we implemented a complete heterogeneous system composed of FPGAs and a processor to show the capabilities of the co-synthesis toolchain. Additionally to the JPEG encoder, we designed a JPEG decoder for creating a motion JPEG codec, represented in Fig. 8. A Xilinx Spartan-3 FPGA and a Virtex-6 FPGA are selected platforms for the implementation. TCP/IP Ethernet and PCIe are used to communicate between the host and the FPGAs. TCP/IP Ethernet is implemented on the spartan 3 using the lwIP IP-core while a PCIe IP-core was developed and ported on the Virtex 6 board. The mapping separates the encoding on the FPGA and the decoding on the host.

Table 2 FPGA occupation of each actor of the RVC-CAL JPEG encoder versus the handwritten VHDL JPEG encoder

Logic utilization	RVC-CAL JPEG Encoder						Handwritten VHDL	Available On Virtex 6
	RasterToMB	FDCT	Quant-ZZ	Huffman	Bits Writer	Total		
Registers	1082	1470	139	1599	625	4893	17869	160000
Slice LUTs	1007	3348	626	4145	1489	10167	16439	80000
BRAMs	8	x	x	6	1	30	35	264
DSP48E1s	x	2	5	1	x	8	2	480
Freq. (MHz)	93	89	208	90	171	86	74	x
Code Lines	200	579	75	416	416	1701	9242	x

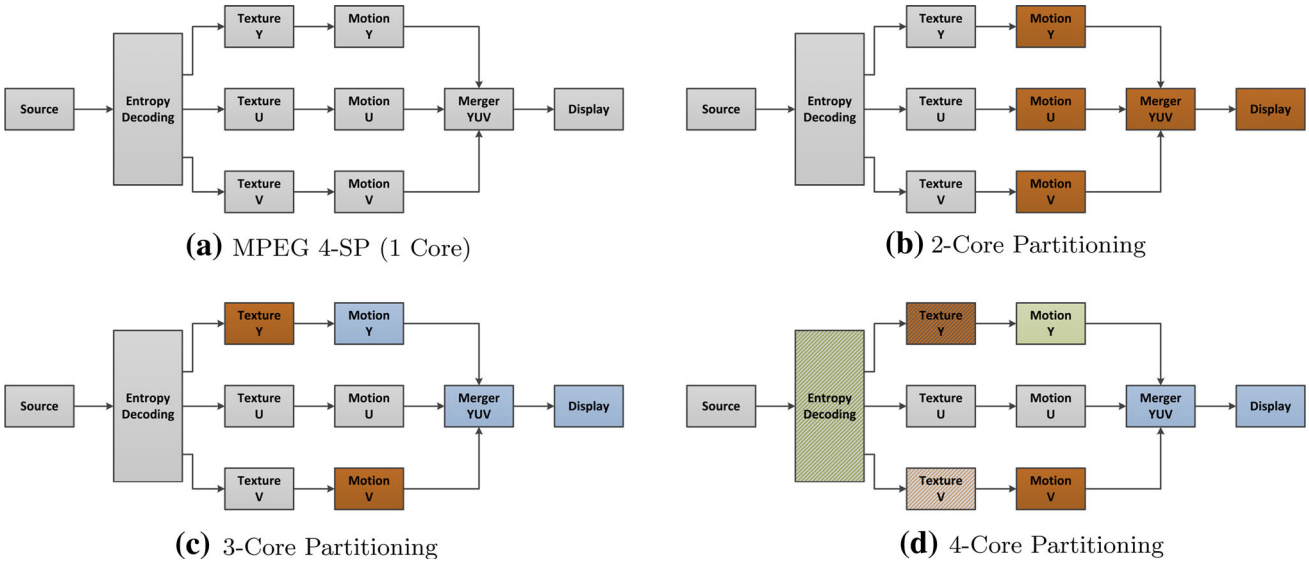

Fig. 7 The MPEG 4 SP YUV-parallel decoder and its partitioning from 1 to 4 cores

Table 3 Framerate of the YUV-parallel MPEG-4 SP decoder at QCIF, SD and HD resolutions

Platform	Resolution	Framerate (# of cores)			
		1	2	3	4
Intel i7-870	176 × 144	1580	2940	4303	5494
	704 × 576	104	178	267	340
	1280 × 720	34	62	75	89
Freescale P4080	176 × 144	223	465	711	853
	704 × 576	15	30	43	52
	1280 × 720	5	9	13	18

Table 4 Speedup of the MPEG-4 decoder running on an X86 quad-core and on a PowerPC e500 8-core processors

Platform	Cores			
	1	2	3	4
Intel Core i7	1	1.86	2.72	3.47
Freescale e500	1	2.08	3.18	3.86

Table 5 Speedup of the MPEG-4 SP decoder on an Intel i7-870 processor at QCIF, SD and HD resolutions

Resolutions	Cores			
	1	2	3	4
QCIF	1	1.86	2.72	3.47
SD	1	1.71	2.56	3.27
HD	1	1.8	2.6	3.6

In terms of throughput, we obtained 3.7 fps in the TCP/IP configuration and 14 fps for the PCIe configuration for a 512 × 512 sequence. Comparing to the state-of-the-art results, we note that it outperforms the ones from Refs. [8] and [10] in the PCIe configuration. However, we still face a communication-bound problem where the bandwidth slows down the overall system. Indeed, the FPGA can encode at 34 fps and the host decodes at 350 fps. In the experiment, it was measured that the platform with TCP/IP Ethernet implementation presents a bandwidth of 6 Mbps, while the PCIe provides a bandwidth of 8 Mbps. The theoretical bandwidths are enough for the input test sequence. The

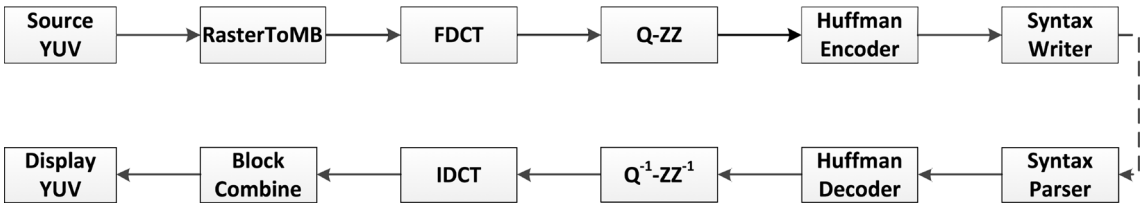

Fig. 8 The RVC-CAL JPEG codec

measures tend to indicate that the hand-checking protocol to synchronize accesses to external memory is (for instance, the PCIe driver does not yet use DMA to communicate data to the external memory) the bottleneck for the system performance. Future developments will include improvements to the bandwidth as well as the hand-checking protocol (DMA, burst-mode, etc.) that remains library components of the experimental design flow process.

8 Conclusions and future work

The paper presented a high-level dataflow-based methodology that provides a unified and portable approach for the hardware–software co-design. The portability of applications has been demonstrated by validating several configurations onto different platform architectures. This design feature makes possible to test and validate the performance of a large number of design option, fundamental feature enabling the rapid prototyping of applications onto heterogeneous architectures.

From the design space exploration side, future works of the design flow will focus on the improvement of the scheduling and partitioning heuristics. Particularly, some performance improvements may result from a refinement of the simple assumptions on the communication model between partitions. Another intriguing research direction consists of exploiting finer granularity and architecture-aware parallelism such as SIMD and MIMD within the source code synthesis of actor executions. Such opportunity may enable to take advantage of instruction-level parallelism of VLIW or GPU architectures. From the implementation side and synthesis of interconnections, works are in progress for improving the communication bandwidth obtainable between PEs for the most used interface components.

References

1. De Micheli, G.: Hardware synthesis from C/C++ models. In: Proceedings of Design, Automation and Test in Europe Conference and Exhibition, pp. 382–383 (1999)
2. Gupta, R., De Micheli, G.: Hardware–software cosynthesis for digital systems. Des. Test Comput. IEEE **10**, 29–41 (1993)
3. Kalavade, A., Lee, E.: A hardware–software codesign methodology for DSP applications. Des. Test Comput. IEEE **10**, 16–28 (1993)
4. Balarin, F., Chiodo, M., Giusto, P., Hsieh, H., Jurecska, A., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A., Sentovich, E., Suzuki, K., Tabbara, B.: Hardware–software co-design of embedded systems: the POLIS approach. Kluwer, Norwell (1997)
5. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM **21**, 666–677 (1978)
6. Dennis, J.B.: First version of a data flow procedure language. In: Symposium on Programming, pp. 362–376 (1974)
7. Kahn, G.: The Semantics of simple language for parallel programming. In: IFIP Congress, pp. 471–475 (1974)
8. Stefanov, T., Zissulescu, C., Turjan, A., Kienhuis, B., Deprette, E.: System design using Khan process networks: the Compaan/Laura approach. In: Proceedings of Design, Automation and Test in Europe Conference and Exhibition, vol. 1, pp. 340–345 (2004)
9. Ha, S., Kim, S., Lee, C., Yi, Y., Kwon, S., Joo, Y.-P.: Peace: a hardware–software codesign environment for multimedia embedded systems. ACM Trans. Des. Autom. Electron. Syst. **12**, 24:1–24:25 (2008)
10. Keinert, J., Streubühr, M., Schlichter, T., Falk, J., Gladigau, J., Haubelt, C., Teich, J., Meredith, M.: SystemCoDesigner—an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications. ACM Trans. Des. Autom. Electron. Syst. **14**, 1:1–1:23 (2009)
11. Eker, J., Janneck, J.: CAL Language Report, Tech. Rep. ERL Technical Memo UCB/ERL M03/48, University of California at Berkeley (2003)
12. Ersfolk, J., Roquier, G., Jokhio, F., Lilius, J., Mattavelli, M.: Scheduling of dynamic dataflow programs with model checking. In: IEEE Workshop on Signal Processing Systems (SiPS), pp. 37–42 (2011)
13. Ersfolk, J., Roquier, G., Lilius, J., Mattavelli, M.: Scheduling of dynamic dataflow programs based on state space analysis. In: IEEE Workshop on International Conference on Acoustics, Speech, and Signal Processing (ICASSP) (2012)
14. Janneck, J.: A machine model for dataflow actors and its applications. In: 45th Annual Asilomar Conference on Signals, Systems, and Computers (2011)
15. Lucarz, C.: Dataflow programming for systems design space exploration for multicore platforms. PhD thesis, Lausanne, (2011)
16. Ab Rahman, A.A.-H., Prihozhy, A., Mattavelli, M.: Pipeline synthesis and optimization of FPGA-based video processing applications with CAL. EURASIP J. Image Video Process. **2011**(1), 19 (2011)
17. Pelcat, M., Nezan, J., Piat, J., Croizer, J., Aridhi, S.: A system-level architecture model for rapid prototyping of heterogeneous multicore embedded systems. In: Conference on Design and Architectures for Signal and Image Processing (DASIP) (2009)
18. Lee, E.A., Messerschmitt, D.G.: Static scheduling of synchronous data flow programs for digital signal processing. IEEE Trans. Comput. **36**(1), 24–35 (1987)
19. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J.: Cyclo-static data flow. In: International Conference on Acoustics,

Speech, and Signal Processing (ICASSP-95), vol. 5, pp. 3255–3258 (1995)
20. Lee, E., Parks, T.: Dataflow process networks. Proc. IEEE **83**, 773–801 (1995)
21. Janneck, J., Miller, I., Parlour, D., Roquier, G., Wipliez, M., Raulet, M.: Synthesizing hardware from dataflow programs: an MPEG-4 simple profile decoder case study. J. Signal Process. Syst. **63**(2), 241–249 (2009). doi:10.1007/s11265-009-0397-5
22. Eker, J., Janneck, J.W.: A structured description of dataflow actors and its applications. Tech. Rep. UCB/ERL M03/13, EECS Department, University of California, Berkeley (2003)
23. Wipliez, M., Raulet, M.: Classification and transformation of dynamic dataflow programs. In: Conference on Design and Architectures for Signal and Image Processing (DASIP), pp. 303–310 (2010)
24. Wipliez, M., Roquier, G., Nezan, J.-F.: Software code generation for the RVC-CAL language. J. Signal Process. Syst. **63**(2), 203–213 (2009). doi:10.1007/s11265-009-0390-z
25. Amer, I., Lucarz, C., Roquier, G., Mattavelli, M., Raulet, M., Nezan, J.-F., Déforges, O.: Reconfigurable video coding on multicore. IEEE Signal Process. Mag. **26**, 113–123 (2009)
26. I. 23001-4:2009: Information technology—MPEG systems technologies—Part 4: Codec configuration representation (2009)
27. Liu, W., Gu, Z., Xu, J., Wang, Y., Yuan, M.: An efficient technique for analysis of minimal buffer requirements of synchronous dataflow graphs with model checking. In: Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '09), New York, NY, USA, pp. 61–70. ACM (2009)
28. Coffman, E.G.: Computer and Job Shop Scheduling Theory. Wiley, New York (1976)
29. Lee, E., Ha, S.: Scheduling strategies for multiprocessor real-time DSP. In: Global Telecommunications Conference (GLOBECOM '89), vol. 2, pp. 1279–1283. IEEE (1989)
30. Casale Brunet, S., Mattavelli, M., Janneck, J.: Profiling of dataflow programs using post mortem causation traces. In: IEEE Workshop on Signal Processing Systems (2012, in press)
31. Gu, R., Janneck, J. W., Raulet, M., Bhattacharyya,S.S.: Exploiting statically schedulable regions in dataflow programs. In: Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'09), Washington, DC, USA, pp. 565–568. IEEE Computer Society (2009)
32. Bunt,R.B., Hume, J.N.P.: A simulation study of a demand-driven scheduling algorithm. In: Proceedings of the 3rd symposium on Simulation of computer systems (ANSS'75), Piscataway, NJ, USA, pp. 117–126. IEEE Press (1975)
33. The Open RVC-CAL Compiler Suite. http://orcc.sourceforge.net/
34. Bezati, E., Yviquel, H., Raulet, M., Mattavelli, M.: A unified hardware/software co-synthesis solution for signal processing systems. In: Conference on Design and Architectures for Signal and Image Processing (DASIP), pp. 1–6 (2011)
35. OpenForge. https://openforge.sourceforge.net
36. OpenCores. http://www.opencores.org/
37. Carlsson, A.; Eker, J.; Olsson, T.; von Platen, C.: Scalable parallelism using dataflow programming. In: Ericson Review. http://www.ericsson.com (2011)

Author Biographies

Endri Bezati is a PhD student at the SCI-STI-MM Multimedia Group, Ecole Polytechnique Federale de Lausanne (EPFL), Switzerland. He received his Master Degree in Electrical Engineering and Computer Science from Institut National des Sciences Appliquees of Rennes (INSA), France. His research interests include high-level programming, co-synthesis and massively parallel computing. He is currently pursuing research on the topic "High Level Dataflow Programming of Hybrid Massively-Parallel Architectures".

Richard Thavot is a PhD student and research staff at the SCI-STI-MM Multimedia Group, Swiss Federal Institute of Technology (EPFL), Switzerland. He received his Bachelor Degree in Electrical Engineering and Computer science from University of Burgundy, France. Later, he received his Master Degree in Datatronics from ESIREM Dijon, France. His research interests include high-level synthesis, co-design, advanced driver assistance systems, real-time embedded systems, parallel processing and inter-process communication. He is currently pursuing research on the topic "High level communication interface synthesis for a rapid heterogeneous systems prototyping".

Ghislain Roquier is a post-doc researcher at Swiss Federal Institute of Technology (EPFL), Switzerland. He is a member of the SCI-STI-MM Multimedia Group. In 2005, he received the MSc degree in Signal Processing and Telecommuncation from the Université de Rennes I, France. Then, he joined the Institut National des Sciences Appliquees (INSA), France, where he received his PhD in Electronics in 2008. His main research interests include design and implementation of heterogeneous real-time embedded systems, rapid prototyping methodologies, multi-core computing, and multimedia signal processing.

Marco Mattavelli started his research activity at the "Philips Research Laboratories" of Eindhoven in 1988. In 1991, he joined the Swiss Federal Institute of Technology (EPFL), where he got his PhD in 1996. He has been the chairman of the Implementation Study Group of MPEG ISO/IEC standardization committee for more than 10 years. For his work, he received the ISO/IEC Award in 1997, 2003 and 2011. He is currently teaching and leading a research team on multimedia systems and architectures at École Polytechnique Fédérale de Lausanne (EPFL) in Lausanne Switzerland. His current major research activities include methodologies for specification and modeling of complex systems and architectures for video coding. He is the author of more than 100 publications and co-authors of several books.