Overview of the MPEG Reconfigurable Video Coding Framework

Shuvra S. Bhattacharyya, Johan Eker, Jörn W. Janneck, Christophe Lucarz, Marco Mattavelli, Mickaël Raulet

Springer manuscript No.
(will be inserted by the editor)

Overview of the MPEG Reconfigurable Video Coding Framework

Shuvra S. Bhattacharyya · Johan Eker · Jörn W. Janneck · Christophe Lucarz · Marco Mattavelli · Mickaël Raulet

Received: date / Accepted: date

Abstract Video coding technology in the last 20 years has evolved producing a variety of different and complex algorithms and coding standards. So far the specification of such standards, and of the algorithms that build them, has been done case by case providing monolithic textual and reference software specifications in different forms and programming languages. However, very little attention has been given to provide a specification formalism that explicitly presents common components between standards, and the incremental modifications of such monolithic standards. The MPEG Reconfigurable Video Coding (RVC) framework is a new ISO standard currently under its final stage of standardization, aiming at providing video codec specifications at the level of library components instead of monolithic algorithms. The new concept is to be able to specify a decoder of an existing standard or a completely new configuration that may better satisfy application-specific constraints by selecting standard components from a library of standard coding algorithms. The possibility of dynamic configuration and reconfiguration of codecs also requires new methodologies and new tools for describing the new bitstream syntaxes and the parsers of such new codecs. The RVC framework is based on the usage of a new actor/dataflow oriented language called CAL for the specification of the standard library and instantiation of the RVC decoder model. This language has been specifically designed for modeling complex signal processing systems. CAL dataflow models expose the intrinsic concurrency of the algorithms by employing the notions of actor programming and dataflow. The paper gives an overview of the concepts and technologies building the standard RVC framework and the non standard tools supporting the RVC model from the instantiation and simulation of the CAL model to software and/or hardware code synthesis.

Keywords Reconfigurable Video Coding · CAL actor language · dataflow programming · code synthesis

Shuvra S. Bhattacharyya
Dept. of ECE and UMIACS, University of Maryland, College Park, MD 20742, USA
E-mail: ssb@umd.com

Johan Eker
Ericsson Research, Mobile Platforms, SE-221 83, Lund, Sweden
E-mail: johan.eker@ericsson.com

Jörn W. Janneck
Xilinx Research Labs, San Jose, CA 95124, USA
E-mail: jorn.janneck@xilinx.com

Christophe Lucarz, Marco Mattavelli
Microelectronic Systems Lab, EPFL, CH-1015 Lausanne, Switzerland
E-mail: firstname.lastname@epfl.ch

Mickaël Raulet
IETR/INSA Rennes, F-35043, Rennes, France
E-mail: mickael.raulet@insa-rennes.fr

1 Introduction

A large number of successful MPEG (Motion Picture Expert Group) video coding standards have been developed since the first MPEG-1 standard in 1988. The standardization efforts in the field, beside having as first objective to guarantee the interoperability of compression systems, has also aimed at providing appropriate forms of specifications for wide and easy deployment. While at the beginning MPEG-1 and MPEG-2 were only specified by textual descriptions, with the increasing complexity of algorithms, starting with the MPEG-4 set of standards, C or C++ specifications, called also reference software, have became the formal specification of the standards. However, such descriptions composed

of non-optimized non-modular software packages have started to show many limits. If we consider that they are in practice the starting point of any implementation, system designers have to rewrite these software packages not only to try optimize performances, but also to transform these descriptions into appropriate forms adapted to the current system design methodologies. Such monolithic specifications hide the inherent parallelism and the dataflow structure of the video coding algorithms, features that are necessary to be exploited for efficient implementations. In the meanwhile the evolution of video coding technologies, leads to solutions that are increasingly complex to be designed and present significant overlap between successive versions of the standards. Consequently, adding new coding tools to a standard involves a new specification for which all its components are modified whereas only a few tools and interfaces are changed. Another problem raised by the wide variety of video coding tools is the selection of the subsets of coding tools used by a specific application. These (sub-)sets are also known as "profiles" in MPEG. The "a priori" specification of a small number of such profiles has become very problematic. Such "average" subsets prevent the possibility of optimally satisfy a variety of specific applications, whereas the specification of too many profiles would result in an obstacle for guaranteeing interoperability. The observation of these drawbacks of current video standard specification formalism led to the development of the Reconfigurable Video Coding (RVC) standard. The key concept behind the project is to be able to design a decoder at a higher level of abstraction than the one provided by current generic monolithic C based specifications. Instead of low level C/C++ code, an "abstract" model based on modular components taken from the standard library, is the reference specification. Functionality of the coding tools and their potential concurrency are explicitly exposed to implementers by the specification formalism chosen. This solution is by far a better starting point for any design and implementation methodology and process. Conversely the old C/C++ source code specification had to include by its nature of being a sequential formalism a specific scheduling of operation without explicitly providing what scheduling are needed for the intrinsic algorithm data dependencies and which one are arbitrary defined in the reference specification. Indeed RVC provides a high-level description of the MPEG standard using as new expressive and compact form of reference SW, for each module of the standard library and for each instantiation of a new decoder configuration. A specific language called CAL [7] is the core of the RVC dataflow model that expose the intrinsic concurrency of the algorithms by employing the notions of actor programming and dataflow. Concurrency and parallelism are fundamental aspects of embedded system design as we enter in the multicore era. Moreover, the standard ISO/IEC MPEG RVC framework is also supported by a non normative design framework composed by a simulation platform and by synthesis tools, Cal2C [27,31] and Cal2HDL [16,15], providing direct conversions to C and HDL implementations.

The paper also provides an overview on such code generator tools and on the principles on which they are based. Results on a real design case of a MPEG-4 Simple Profile decoder show that systems obtained with the hardware code synthesis from a CAL model outperform the hand written VHDL version both in terms of performance, resource usage and design efforts.

2 Reconfigurable Video Coding: ISO-MPEG standardization

MPEG has produced several video coding standards such as MPEG-1, MPEG-2, MPEG-4 Video, AVC (Advanced Video Coding) and recently SVC (Scalable Video Coding). However, the past monolithic specification of such standards (usually in the form of C/C++ programs) lacks flexibility and does not allow to use the combination of coding algorithms from different standards enabling to achieve specific design or performance trade-offs and thus fill, case by case, the requirements of specific applications. Indeed, not all coding tools defined in a *profile@level* of a specific standard are required in all application scenarios. For a given application, codecs are either not exploited at their full potential or require unnecessarily complex implementations. However, a decoder conformant to a standard has to support all of them and may results in non-efficient implementations.

So as to overcome the limitations intrinsic of specifying codecs algorithms by using monolithic imperative code, a profiled version of CAL language has been selected by the ISO/IEC standardization organization in the new MPEG standard called RVC which stays for Reconfigurable Video Coding. RVC is composed of two ISO/IEC standard specifications: [13] and [14]. The data-driven programming paradigm of CAL dataflow language that lends itself naturally to describing the processing of media streams that pervade the world of media coding, has been restricted and simplified from its original formulation so as to provide description that can be synthesized to software and hardware implementation by already existing tools. However, the original expressivity of the language and the strong encapsulation features offered by the actor programming model

have been preserved and provide a solid foundation for the compact and modular specification of media codecs.

MPEG RVC is a framework allowing users to define a multitude of different codecs, by combining together Functional Units (FUs) (i.e. actors in programming language) modeling video coding tools, from the MPEG standard library written in CAL, that contains video technology from all existing MPEG standards (i.e. MPEG-2, MPEG-4, MPEG-4 AVC , etc...). The reader can also refer to [19] for more information and details about RVC. RVC-CAL is used to provide the reference software for all coding tools (FUs) of the entire library. The essential elements of the RVC framework (Fig. 1) include:

- the standard Video Tool Library (VTL) [14] which contains the video coding tools. RVC-CAL is used to describe the algorithmic behaviour of the FUs that end to be video coding algorithmic components self contained and communicating with the external world only by means of input and output ports.
- a language called Functional unit Network Language (FNL) [13], an XML dialect, used to specify a decoder configuration made up of FUs taken from the VTL and the connections between the FUs. Also this language is specified and standardized in [13].
- a MPEG-21 Bitstream Syntax Description Language (BSDL) [12] schema which describes the syntax of the bitstream that a RVC decoder has to decode. In [26,20], tools and methodologies for the validation of BSDL syntaxes are described in full details as well as some examples of systematic procedures for the direct synthesis of parsers in the CAL dataflow specification formalism. Such BSDL to CAL translator is in its final stage of development as part of the Open Dataflow effort [30,3] (see sect. 6).

In summary the components and processes that lead to the specification and implementation of a new MPEG RVC decoder are based on the CAL dataflow model of computation and are:

- a Decoder Description (DD) written in FNL describing the architecture of the decoder, in terms of FUs and their connections.
- an Abstract Decoder Model (ADM), a behavioral (CAL) model of the decoder composed of the syntax parser specified by the BSDL schema, FUs from the VTL and their connections.
- the final decoder implementation that is either generated by instantiating any proprietary implementation, conformant in terms of I/O behavior, of the standard RVC FUs, or obtained directly from the ADM by generating SW and/or HW implementations by means of appropriate synthesis tools.

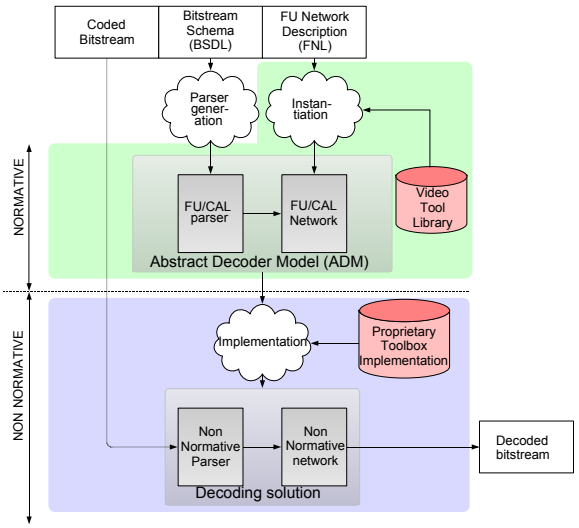

Fig. 1 Illustration of the standard components of Reconfigurable Video Coding framework.

Thus, based on CAL dataflow formalism, designers can build video coding algorithm with a set of self-contained modular elements coming from the MPEG RVC standard library (VTL). However, the new CAL based specification formalism, not only provide the flexibility required by the process itself of specifying a standard video codec, but also yields a specification of such standard that is the appropriate starting point for the implementation of the codec on the new generations of multicore platforms. In fact the RVC ADM is nothing else that a CAL dataflow specification that implicitly exposes all concurrency and parallelism intrinsic to the model, features that classical generic specifications based on imperative languages have not provided so far.

MPEG RVC toolbox includes about 150 Functional units that can be configured to specify the MPEG-2 MP, MPEG-4 SP and ASP, AVCBP, AVCHP and SVC. Some FU can already be used to generate new extended profiles, for instance supporting 422 and 444 chrominance patterns if not available in the original profile. Conformance testing of the RVC toolbox for single FUs and entire decoders are currently on going.

As illustrated in picture 1 the MPEG RVC standard does not specify anything concerning the implementation methodology and technology of a RVC decoder. The upper part of the picture illustrates the process that, starting from a normative description of a decoder composed by a bitstream syntax description (expressed in RVC-BSDL in a XML schema), by a FU network description (expressed in FNL) and by the knowledge of the I/O behavior of the FU specified in the RVC toolbox, provide a normative (executable) description

of the RVC decoder. Indeed what is called an abstract decoder description is a CAL dataflow program that constitutes the *conformance point* between the normative RVC specification and all possible proprietary implementations that be generated to decode the incoming bitstreams. Thus the MPEG RVC standard leaves open the platforms and the implementation methodologies that can be used to generate any RVC proprietary implementation. This provides all possibility of generating parallel and concurrent implementations for a wide variety of existing and emerging implementation platforms. An important consequence of the fact that no specific implementation technology is specified by the MPEG RVC standard is that it is not limited to applications that require direct download and on the fly instantiation of proprietary implementations. Several application scenarios and transport mechanism of the RVC decoder description are under development and will be included in appropriate amendment of the MPEG Systems standard. Thus, indirect generations of implementations, such as the one available via web download and other form of deployment, will be possible together with the direct synthesis of SW and HW from the ADM. All these possibilities enable, for each application scenario, the users to select the most appropriate implementation methodology.

More details on different implementation approaches, on the methodologies for generating parsers from bitstream syntax descriptions are available on papers published in this special issue. Detailed examples on new decoder configurations optimizing different implementation objectives such as concurrency and parallelism or overall complexity cannot be provided here for space reasons and only concept and principles behind all RVC formalism are discussed in the follow of the paper as well some the concept and examples of the tools synthesizing CAL to SW and HW implementations.

3 Why C etc. Fail

Having introduced the main motivations that lead MPEG to introduce CAL dataflow based specifications as well as the elements that compose the RVC framework, this section will briefly extend the discussion adding other reasons for which a paradigm shift in system specification and modeling has been necessary and could be advantageous not only for the field of video compression. The control over low-level detail, which is considered a merit of C, tends to over-specify programs: not only the algorithms themselves are specified, but also how inherently parallel computations are sequenced, how inputs and outputs are passed between the algorithms and, at a higher level, how computations are mapped to threads, processors and application-specific hardware. It is not always possible to recover the original knowledge about the program by means of analysis and the opportunities for restructuring transformations are limited.

Code generation is constrained by the requirement of preserving the semantic effect of the original program. What constitutes the semantic effect of a program depends on the source language, but loosely speaking some observable properties of the program's execution are required to be invariant. Program analysis is employed to identify the set of admissible transformations; a code generator is required to be conservative in the sense that it can only perform a particular transformation when the analysis results can be used to prove that the effect of the program is preserved. *Dependence analysis* is one of the most challenging tasks of high-quality code generation (for instance see [32]). It determines a set of constraints on the order, in which the computations of a program may be performed. Efficient utilization of modern processor architectures heavily depends on dependence analysis, for instance:

- to determine efficient mappings of a program onto multiple processor cores (*parallelization*),
- to utilize so called SIMD or "multimedia" instructions that operate on multiple scalar values simultaneously (*vectorization*), and
- to utilize multiple functional units and avoid pipeline stalls (*instruction scheduling*).

Determining (a conservative approximation of) the dependence relation of a C program involves *pointer analysis*. Since the general problem is undecideable, a trade-off will always have to be made between the precision of the analysis and its resource requirements [10].

4 Why dataflow might actually work

Scalable parallelism. In parallel programming, the number of things that are happening at the same time can scale in two ways: It can increase with the size of the problem or with the size of the program. Scaling a regular algorithm over larger amounts of data is a relatively well-understood problem, while building programs such that their parts execute concurrently without much interference is one of the key problems in scaling the von Neumann model. The explicit concurrency of the actor model provides a straightforward parallel composition mechanism that tends to lead to more parallelism as applications grow in size, and scheduling techniques permit scaling concurrent descriptions onto platforms with varying degrees of parallelism.

Modularity, reuse. The ability to create new abstractions by building reusable entities is a key element in every programming language. For instance, object-oriented programming has made huge contributions to the construction of von Neumann programs, and the strong encapsulation of actors along with their hierarchical composability offers an analog for parallel programs.

Scheduling. In contrast to procedural programming languages, where control flow is made explicit, the actor model emphasizes explicit specification of concurrency.

Portability. Rallying around the pivotal and unifying von Neumann abstraction has resulted in a long and very successful collaboration between processor architects, compiler writers, and programmers. Yet, for many highly concurrent programs, portability has remained an elusive goal, often due to their sensitivity to timing. The untimedness and asynchrony of stream-based programming offers a solution to this problem. The portability of stream-based programs is underlined by the fact that programs of considerable complexity and size can be compiled to competitive hardware [16, 15] as well as software [27], which suggests that stream-based programming might even be a solution to the old problem of flexibly co-synthesizing different mixes of hardware/software implementations from a single source.

Adaptivity. The success of a stream programming model will in part depend on its ability to configure dynamically and to virtualize, i.e. to map to collections of computing resources too small for the entire program at once. The transactional execution of actors generates points of *quiescence*, the moments between transactions, when the actor is in a defined and known state that can be safely transferred across computing resources.

5 The Cal Actor Language

Cal [7] is a domain-specific language that provides useful abstractions for dataflow programming with actors. Cal has been used in a wide variety of applications and has been compiled to hardware and software implementations, and work on mixed HW/SW implementations is under way. The next section provides a brief introduction to some key elements of the language.

5.1 Basic Constructs

The basic structure of a Cal actor is shown in the `Add` actor (Fig. 2), which has two input ports `A` and `B`, and one output port `Out`, all of type `T`. The actor contains one *action* that consumes one token on each input ports, and produces one token on the output port. An action may *fire* if the availability of tokens on the input ports matches the *port patterns*, which in this example corresponds to one token on both ports `A` and `B`.

```
actor Add() T A, T B ⇒ T Out :
  action [a], [b] ⇒ [sum]
  do
     sum := a + b;
  end
end
```

Fig. 2 Basic structure of a Cal actor.

An actor may have any number of actions. The untyped `Select` actor (Fig. 3) reads and forwards a token from either port `A` or `B`, depending on the evaluation of guard conditions. Note that each of the actions has empty bodies.

```
actor Select () S, A, B ⇒ Output:

    action S: [sel], A: [v] ⇒ [v]
    guard sel end

    action S: [sel], B: [v] ⇒ [v]
    guard not sel end
end
```

Fig. 3 Guard structure in a Cal actor.

5.2 Priorities and State Machines

An action may be labeled and it is possible to constrain the legal firing sequence by expressions over labels. In the `PingPongMerge` actor, reported in the Figure 4, a finite state machine *schedule* is used to force the action sequence to alternate between the two actions `A` and `B`. The schedule statement introduces two states `s1` and `s2`.

The `Route` actor, in the Figure 5, forwards the token on the input port `A` to one of the three output ports. Upon instantiation it takes two parameters, the functions `P` and `Q`, which are used as predicates in the guard conditions. The selection of which action to fire is in this example not only determined by the availability of tokens and the guards conditions, by also depends on the `priority` statement.

```
actor PingPongMerge () Input1, Input2 ⇒
Output:

  A: action Input1: [x] ⇒ [x] end
  B: action Input2: [x] ⇒ [x] end

  schedule fsm s1:
    s1 (A) --> s2;
    s2 (B) --> s1;
  end
end
```

Fig. 4 FSM structure in a CAL actor.

```
actor Route (P, Q) A ⇒ X, Y, Z:

    toX: action [v] ⇒ X: [v]
         guard P(v) end
    toY: action [v] ⇒ Y: [v]
         guard Q(v) end
    toZ: action [v] ⇒ Z: [v] end

    priority
        toX > toY > toZ;
    end
end
```

Fig. 5 Priority structure in a CAL actor.

5.3 CAL subset language for RVC

For an in-depth description of the language, the reader is referred to the language report [7], for the specific subset specified and standardized by ISO in the Annex C of [13]. This subset only deals with fully typed actors and some restrictions on the CAL language constructs from [7] to have efficient hardware and software code generations without changing the expressivity of the algorithm. For instance, Figures 3, 4 and 5 are not RVC-CAL compliant and must be changed as the Figures 6, 7 and 8 where *T1, T2, T* are the types and only typed paramters can be passed to the actors not functions as *P, Q*.

```
actor Select () T1 S, T2 A, T3 B ⇒ T3 Output:

    action S: [sel], A: [v] ⇒ [v]
    guard sel end

    action S: [sel], B: [v] ⇒ [v]
    guard not sel end
end
```

Fig. 6 Guard structure in a RVC-CAL actor.

A large selection of example actors is available at the OpenDF repository [30], among them can also be found

```
actor PingPongMerge () T Input1, T Input2 ⇒
T Output:

  A: action Input1: [x] ⇒ [x] end
  B: action Input2: [x] ⇒ [x] end

  schedule fsm s1:
    s1 (A) --> s2;
    s2 (B) --> s1;
  end
end
```

Fig. 7 FSM structure in a RVC-CAL actor.

```
actor Route () T A ⇒ T X, T Y, T Z:
    funtion P(T v_in)--> T:
    \\ body of the function P
      P(v_in)
    end
    funtion Q(T v_in)--> T:
    \\ body of the function P
      Q(v_in)
    end

    toX: action [v] ⇒ X: [v]
         guard P(v) end
    toY: action [v] ⇒ Y: [v]
         guard Q(v) end
    toZ: action [v] ⇒ Z: [v] end

    priority
        toX > toY > toZ;
    end
end
```

Fig. 8 Priority structure in a RVC-CAL actor.

the MPEG-4 decoder discussed below. Many other actors written in RVC-CAL will be soon be available as standard MPEG RVC tool repository once the conformance testing process will be completed.

5.4 Networks

A set of CAL actors are instantiated and connected to form a CAL application, i.e. a CAL network. Figure 9 shows a simple CAL network `Sum`, which consists of the previously defined RVC-CAL `Add` actor and the delay actor shown in Figure 10.

The source/language that defined the network `Sum` is found in Figure 11. Please, note that the network itself has input and output ports and that the instantiated entities may be either actors or other networks, which allows for a hierarchical design.

Formerly, networks have been traditionally described in a textual language, which can be automatically converted to FNL and vice versa - the XML dialect

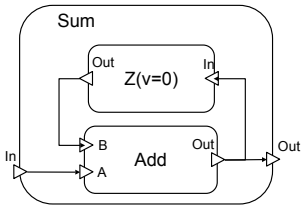

Fig. 9 A simple CAL network.

```
actor Z (v) T In ⇒ T Out:

  A: action ⇒ [v] end
  B: action [x] ⇒ [x] end

  schedule fsm s0:
    s0 (A) —> s1;
    s1 (B) —> s1;
  end
end
```

Fig. 10 RVC-CAL Delay actor.

```
network Sum () In ⇒ Out:

entities
  add = Add();
  z = Z(v=0);

structure
  In —> add.A;
  z.Out —> add.B;
  add.Out —> z.In;
  add.Out — > Out;
end
```

Fig. 11 Textual representation of the Sum network.

standardized by ISO in Annex B of [13]. The XML representation of the Sum network is found in Figure 12. A graphical editing framework called Graphiti editor [8] is available to create, edit, save and display a network. The XML and textual format for the network description are supported by such editor.

6 Tools

6.1 CAL Simulators and Code generators

CAL is supported by a portable interpreter infrastructure that can simulate a hierarchical network of actors. This interpreter was first used in the Moses [21] project. Moses features a graphical network editor, and allows the user to monitor actors execution (actor state and token values). The project being no longer maintained, it has been superseded by an Eclipse environment composed of 2 tools/plugins—the Open Dataflow environment for CAL editing (OpenDF [30] for short) and the Graphiti editor for graphically editing the network.

```
<?xml version="1.0" encoding="UTF-8"?>
<XDF name="Sum">
   <Port kind="Input" name="In"/>
   <Port kind="Output" name="Out"/>
   <Instance id="add"/>
   <Instance id="z">
      <Class name="Z"/>
      <Parameter name="v">
         <Expr kind="Literal"
               literal-kind="Integer" value="0"/>
      </Parameter>
   </Instance>
   <Connection dst="add" dst-port="A"
         src="" src-port="In"/>
   <Connection dst="add" dst-port="B"
         src="z" src-port="Out"/>
   <Connection dst="z" dst-port="In"
         src="add" src-port="Out"/>
   <Connection dst="" dst-port="Out"
         src="add" src-port="Out"/>
</XDF>
```

Fig. 12 XML representation of the Sum network.

OpenDF is also a compilation framework. Today there exists a backend for generation of HDL (VHDL/Verilog) [16,15], and another backend that generates C for integration with the SystemC tool chain [27,31]. A third backend targeting ARM11 and embedded C is under development [23] as part of the EU project ACTORS [1]. It is also possible to simulate CAL models in the Ptolemy II [25] environment.

6.2 Analysis Support

A major benefit of formulating RVC specifications in terms of a formal dataflow modeling language (CAL) is the potential for rigorous analysis and optimization of specifications. This is of increasing importance as more features are embedded and more demands are imposed in terms of real-time performance.

To this end, CAL-based tools for RVC have been integrated with the dataflow interchange format (DIF), which is a textual language for specifying mixed-grain dataflow representations of signal processing applications, and TDP [29] (the DIF package), which is a software tool for analyzing DIF specifications. A major emphasis in DIF and TDP is support for working with and integrating different kinds of specialized dataflow models of computation and their associated analysis techniques. Such functionality is useful, for example, as a follow-on step to the automated detection of specialized dataflow regions in CAL networks. Once such regions are detected, they can be annotated with corresponding DIF keywords — e.g., CSDF (cyclo-static dataflow) and SDF (synchronous dataflow) — and then scheduled and integrated with appropriate TDP-based analysis methods. Such a linkage between CAL and TDP is under

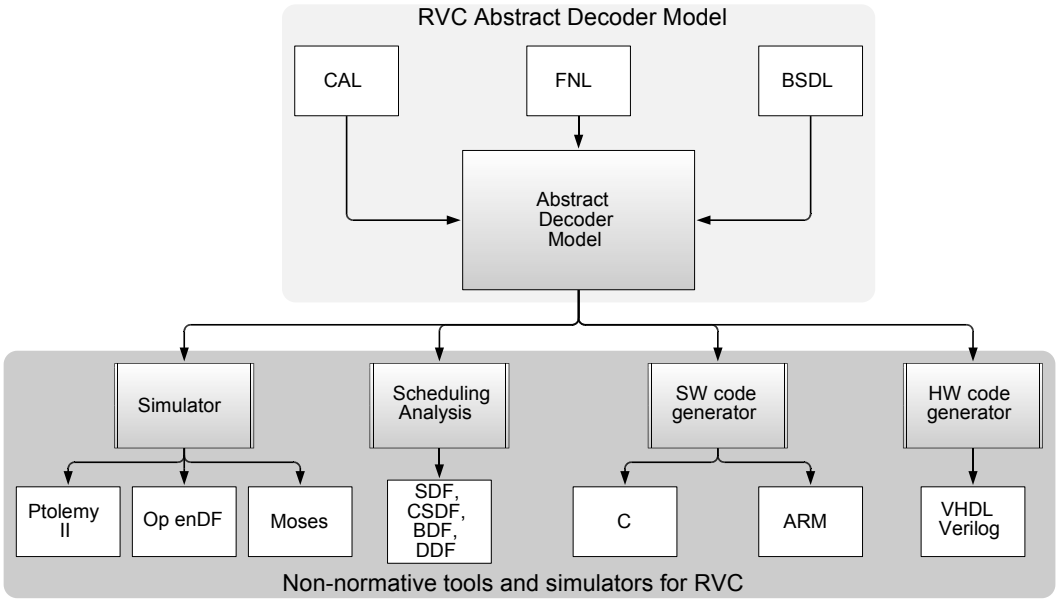

Fig. 13 OpenDF: tools

active development as a joint effort between the CAL and DIF projects.

A particular area of emphasis in TDP is support for developing efficient coarse-grain dataflow scheduling techniques. For example, the generalized schedule tree representation in TDP provides an efficient format for storing, manipulating, and viewing schedules [17], and the functional DIF dataflow model provides for flexible prototyping of static, dynamic, and quasi-static scheduling techniques [24]. Libraries of static scheduling techniques and buffer management models for SDF graphs, as well as an SDF-to-C translator are also available in TDP [11]. The set of dataflow models that are currently recognized and supported explicitly in the DIF language and TDP include Boolean dataflow [6], enable-invoke dataflow [24], CSDF [4], homogeneous synchronous dataflow [18,28], multidimensional synchronous dataflow [22], parameterized synchronous dataflow [2], and SDF [18]. These alternative dataflow models have useful trade-offs in terms of expressive power, and support for efficient static or quasi-static scheduling, as well as efficient buffer management. The set of models that is supported in TDP, as well as the library of associated analysis techniques are expanding with successive versions of the TDP software.

The initial focus in integrating TDP with CAL is to automatically-detect regions [9] of CAL networks that conform to SDF semantics, and can leverage the significant body of SDF-oriented analysis techniques in TDP. In the longer term, we plan to target a range of different dataflow models in our automated "region detection" phase of the design flow. This appears significantly more challenging as most other models are more complex in structure compared to SDF; however, it can greatly increase the flexibility with which different kinds of specialized, streaming-oriented dataflow analysis techniques can be leveraged when synthesizing hardware and software from CAL networks.

7 An example of implementation of a MPEG-4 SP decoder by direct synthesis of a MPEG RVC description

One interesting and very attracting implementation methodology of MPEG RVC decoder descriptions is the direct synthesis of the standard specification. It provides a source of relevant application of realistic sizes and complexity and also enables meaningful experiments and advances in dataflow programming. More details on the software and hardware code generators can be found in [15,31]. Some of the authors have performed an implementation study [16,15], in which the RVC MPEG-4 Simple Profile decoder specified in CAL according to the MPEG RVC formalism has been implemented on an FPGA using a CAL-to-RTL code generator called Cal2HDL. The MPEG-4 Simple Profile abstract decoder model that essentially results to be a dataflow program (Figure 14, Table 3), is composed of 27 atomic FUs (or actors in dataflow programming) and 9 sub-networks (actor/network composition); atomic actors can be instantiated several times,

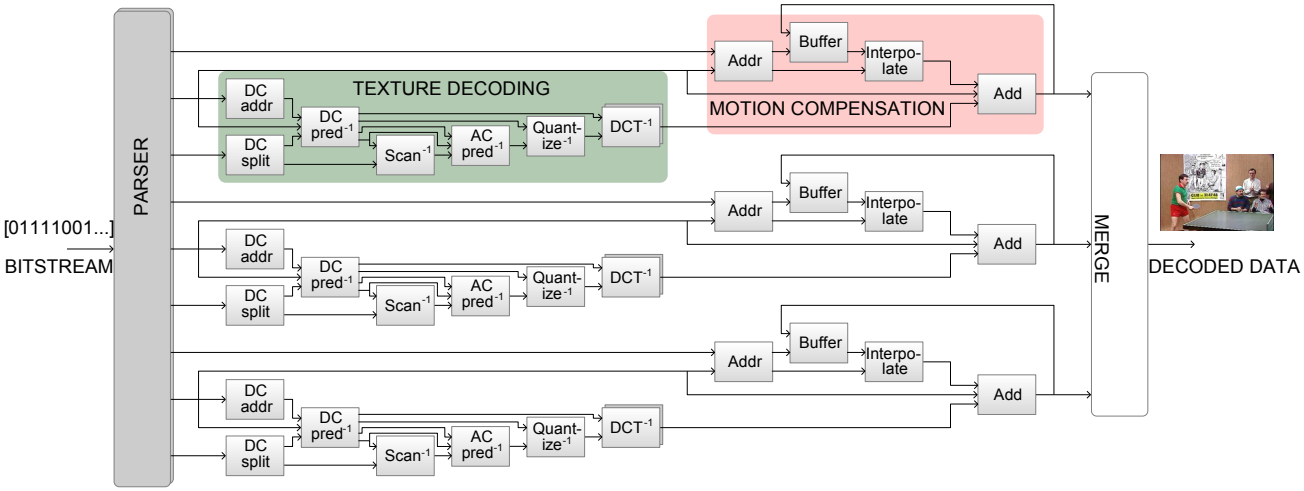

Fig. 14 MPEG-4 Simple Profile RVC specification network.

for instance there are 42 actor instantiations in this dataflow program. Figure 15 shows a top-level view of the decoder. The main functional blocks include the bitstream parser, the reconstruction block, the 2D inverse cosine transform, the frame buffer and the motion compensation module. These functional units are themselves hierarchical compositions of actor networks. The objective of the design was to support 30 frames of 1080p in the YUV420 format per second, which amounts to a production of 93.3 Mbyte of video output per second. The given target clock rate of 120 MHz implies 1.29 cycles of processing per output sample on average.

The results of the implementation study were encouraging in that the code generated from the MPEG RVC CAL specification did not only outperformed the handwritten reference in VHDL, both in terms of throughput and silicon area, but also allowed for a significantly reduced development effort. Table 1 shows the comparison between CAL specification and the VHDL reference.

It should be emphasized that this counter-intuitive result cannot be attributed to the sophistication of the synthesis tool. On the contrary the tool does not perform a number of potential optimizations, such as for instance optimizations involving more than one actor. Instead, the good results appear to be yield by the implementation and development process itself. The implementation approach was based generating a proprietary implementation of the standard MPEG RVC toolbox composed of FUs of lower level of granularity. Thus the implementation methodology was to substitute the FU of the standard abstract decoder model of the MPEG-4 SP with an equivalent implementation, in terms of behavior. Essentially standard toolbox FU were substituted with networks of FU described as actors of lower granularity. A notable difference of such implementation approach when compared with the classical hand writing of HDL code from a textual or sequential specification (i.e. a C/C++ program for instance) was that the CAL specification of the proprietary implementation toolbox (that can be directly derived from the standard RVC toolbox) could go through significantly more design iterations than the one applicable from a hand-written VHDL reference —in spite of being developed in approximately a quarter of the development time (including the time of developing the standard MPEG RVC toolbox from scratch). Whereas a dominant part of the development of a classical VHDL reference development need to be spent getting the system to work correctly, the effort of the CAL specification could be focused on optimizing system performance to meet the design constraints.

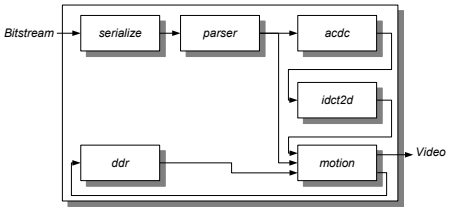

Fig. 15 Top-level dataflow graph of the proprietary implementation of the RVC MPEG-4 decoder.

The initial design cycle of the proprietary RVC library resulted in an implementation that was not only inferior to the VHDL reference, but one that also failed to meet the throughput and area constraints. Subsequent iterations explored several other points in the design space until arriving at a solution that satisfied

the constraints. At least for the considered implementation study, the benefit of short design cycles seem to outweigh the inefficiencies that resulted from high-level synthesis and the reduced control over implementation details.

	Size slices, BRAM	**Speed** kMB/s	**Code size** kSLOC	**Dev. time** MM
CAL	3872, 22	290	4	3
VHDL	4637, 26	180	15	12
Improv. factor	1.2	1.6	3.75	4

kMB/s=kilo macroblocks per second
kSLOC=kilo source lines of code

Table 1 Hardware synthesis results for a proprietary implementation of a MPEG-4 Simple Profile decoder. The numbers are compared with a reference hand written design in VHDL.

In particular, the asynchrony of the programming model and its realization in hardware allowed for convenient experiments with design ideas. Local changes, involving only one or a few actors, do not break the rest of the system in spite of a significantly modified temporal behavior. In contrast, any design methodology that relies on precise specification of timing —such as RTL, where designers specify behavior cycle-by-cycle— would have resulted in changes that propagate through the design.

Table 1 shows the quality of result produced by the RTL synthesis engine of the MPEG-4 Simple Profile video decoder. Note that the code generated from the high-level dataflow RVC description and proprietary implementation of the MPEG toolbox actually outperforms the hand-written VHDL design in terms of both throughput and silicon area for a FPGA implementation.

Another synthesis tool called Cal2C [27,31] validates another implementation methodology of the MPEG-4 Simple Profile dataflow program provided by the RVC standard (Figure 14). Table 2 shows that synthesized C-software is faster than the simulated CAL dataflow program (20 frames/s instead of 0.15 frames/s), and close to real-time decoding for a QCIF format (25 frames/s). However it remains slower than the automatically synthesized hardware description by Cal2HDL [16,15].

MPEG4 SP decoder	**Speed** kMB/s	**Clock speed** GHz	**Code size** kSLOC
CAL **simulator**	0.015	2	3.4
Cal2C	2	2	10.4
Cal2HDL	290	0.12	4

Table 2 MPEG4SP decoder speed and SLOC.

As described above, the MPEG-4 Simple Profile dataflow program is composed of 42 actor instantiations in the flattened dataflow program. The flattened network becomes a C++ file that currently contains a systemC scheduler (sequential fashion scheduler) for both actor scheduling and token consumptions/productions. Each actor becomes a C file containing all its action/processing with its overall action scheduling/-control. Its number of SLOC is shown in Table 3. All of the generated files are successfully compiled by gcc. For instance, the "ParserHeader" actor inside the "Parser" network is the most complex actor with multiple actions. The translated C-file (with actions and state variables) includes 1043 SLOC for actions and 1895 for action scheduling. The original CAL file contains 962 lines of codes as a comparison.

MPEG-4 decoder	**CAL**	**NL**	**C**	**C++**
Number of files	27	9	42	1
Code Size (kSLOC)	2.9	0.5	9.5	0.9

Table 3 Code size and number of files automatically generated for MPEG-4 Simple Profile decoder.

8 Conclusion

This paper describes the essential components of the ISO/IEC MPEG Reconfigurable Video Coding framework based on the dataflow concept and having as core the new specification formalism called RVC-CAL a subset of CAL dataflow language. The RVC MPEG tool library, that covers in modular form all video algorithms from the different MPEG video coding standards, shows that CAL is an appropriate language for supporting design flows aiming at building complex heterogeneous systems from high level system specifications. The MPEG RVC framework is also supported by a simulator, software and hardware code synthesis, and the DIF/TDP analysis tools. CAL dataflow models used by the MPEG RVC standard result also particularly efficient for specifying signal processing systems in a very synthetic form compared to classical imperative languages. Moreover, CAL model libraries can be developed in the form of libraries of proprietary implementations of standard RVC components to describe architectural features of the desired implementation platform, thus enabling the RVC implementer/designer to work at level of abstraction comparable to the one of the RVC video coding algorithms. Hardware and software code generators then provide the low level implementation of the actors and associated network of actors for the different target implementation platforms (multi-core

processors or FPGA). Several works and extension of the tools and implementation methodologies supporting the MPEG RVC framework are currently in development. They include the evolution of the software and hardware code generators in terms the extensions specified by the standard RVC-CAL language, the development of scheduling tools such as the quasi-static scheduling [5] or the DIF/TDP analysis tools [9] for the scheduling/mapping on multicore platforms of SW synthesized from the RVC abstract decoder model and the evolution of the current Open DataFlow environment including tools for more accurate and extended profiling and debugging capabilities.

References

1. Actors FP7 project: URL http://www.actors-project.eu
2. Bhattacharya, B., Bhattacharyya, S.: Parameterized dataflow modeling for DSP systems. Signal Processing, IEEE Transactions on **49**(10), 2408–2421 (2001). DOI 10.1109/78.950795
3. Bhattacharyya, S., Brebner, G., Eker, J., Janneck, J., Mattavelli, M., von Platen, C., Raulet, M.: OpenDF - A Dataflow Toolset for Reconfigurable Hardware and Multicore Systems. In: First Swedish Workshop on Multi-Core Computing (MCC), pp. 43–49 (2008)
4. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J.: Cycle-static dataflow. Signal Processing, IEEE Transactions on **44**(2), 397–408 (1996). DOI 10.1109/78.485935
5. Boutellier, J., Lucarz, C., Lafond, S., Gomez, V., Mattavelli, M.: Quasi-static scheduling of CAL actor networks for reconfigurable video coding. Springer journal of Signal Processing Systems. Special Issue on Reconfigurable Video Coding (2009)
6. Buck, J., Lee, E.: Scheduling dynamic dataflow graphs with bounded memory using the token flow model. In: Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on, vol. 1, pp. 429–432 vol.1 (1993). DOI 10.1109/ICASSP.1993.319147
7. Eker, J., Janneck, J.W.: CAL Language Report Specification of the CAL Actor Language. Tech. Rep. UCB/ERL M03/48, EECS Department, University of California, Berkeley (2003)
8. Graphiti Editor sourceforge: URL http://graphiti-editor.sf.net
9. Gu, R., Janneck, J.W., Raulet, M., Bhattacharyya, S.S.: Exploiting statically schedulable regions in dataflow programs. In: International Conference on Acoustics, Speech and Signal Processing. IEEE conference on (2009)
10. Hind, M.: Pointer analysis: haven't we solved this problem yet? In: Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, pp. 54–61. ACM, Snowbird, Utah, United States (2001). DOI 10.1145/379605.379665
11. Hsu, C.J., Ko, M.Y., Bhattacharyya, S.S.: Software synthesis from the dataflow interchange format. In: Proceedings of the 2005 workshop on Software and compilers for embedded systems, pp. 37–49. ACM, Dallas, Texas (2005). DOI 10.1145/1140389.1140394
12. International Standard ISO/IEC FDIS 23001-5: MPEG systems technologies - Part 5: Bitstream Syntax Description Language (BSDL)
13. ISO/IEC FDIS 23001-4: MPEG systems technologies – Part 4: Codec Configuration Representation (2009)
14. ISO/IEC FDIS 23002-4: MPEG video technologies – Part 4: Video tool library (2009)
15. 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. Springer journal of Signal Processing Systems. Special Issue on Reconfigurable Video Coding (2009)
16. Janneck, J.W., Miller, I.D., Parlour, D.B., Roquier, G., Wipliez, M., Raulet, M.: Synthesizing hardware from dataflow programs: An MPEG-4 simple profile decoder case study. In: Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on, pp. 287–292 (2008). DOI 10.1109/SIPS.2008.4671777
17. Ko, M.Y., Zissulescu, C., Puthenpurayil, S., Bhattacharyya, S., Kienhuis, B., Deprettere, E.: Parameterized Looped Schedules for Compact Representation of Execution Sequences in DSP Hardware and Software Implementation. Signal Processing, IEEE Transactions on **55**(6), 3126–3138 (2007). DOI 10.1109/TSP.2007.893964
18. Lee, E., Messerschmitt, D.: Synchronous data flow. Proceedings of the IEEE **75**(9), 1235–1245 (1987)
19. Lucarz, C., Mattavelli, M., Thomas-Kerr, J., Janneck, J.: Reconfigurable Media Coding: A New Specification Model for Multimedia Coders. In: Signal Processing Systems, 2007 IEEE Workshop on, pp. 481–486 (2007). DOI 10.1109/SIPS.2007.4387595
20. Lucarz, C., Piat, J., Mattavelli, M.: Automatic synthesis of parsers and validation of bitstreams within the MPEG Reconfigurable Video Coding Framework. Springer journal of Signal Processing Systems. Special Issue on Reconfigurable Video Coding (2009)
21. Moses project: URL http://www.tik.ee.ethz.ch/moses/
22. Murthy, P.K., Bhattacharyya, S.S.: Shared memory implementations of synchronous dataflow specifications. In: Proceedings of the conference on Design, automation and test in Europe, pp. 404–410. ACM, Paris, France (2000). DOI 10.1145/343647.343809
23. von Platen, C., Eker, J.: Efficient realization of a cal video decoder on a mobile terminal (position paper). In: Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on, pp. 176–181 (2008). DOI 10.1109/SIPS.2008.4671758
24. Plishker, W., Sane, N., Kiemb, M., Anand, K., Bhattacharyya, S.S.: Functional DIF for Rapid Prototyping. In: Proceedings of the 2008 The 19th IEEE/IFIP International Symposium on Rapid System Prototyping - Volume 00, pp. 17–23. IEEE Computer Society (2008)
25. Ptolemy II: URL http://ptolemy.eecs.berkely.edu
26. Raulet, M., Piat, J., Lucarz, C., Mattavelli, M.: Validation of bitstream syntax and synthesis of parsers in the MPEG Reconfigurable Video Coding framework. In: Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on, pp. 293–298 (2008). DOI 10.1109/SIPS.2008.4671778
27. Roquier, G., Wipliez, M., Raulet, M., Janneck, J.W., Miller, I.D., Parlour, D.B.: Automatic software synthesis of dataflow program: An MPEG-4 simple profile decoder case study. In: Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on, pp. 281–286 (2008). DOI 10.1109/SIPS.2008.4671776
28. Sriram, S., Bhattacharyya, S.S.: Embedded Multiprocessors: Scheduling and Synchronization. Marcel Dekker, Inc. (2000)
29. The DIF Package: URL http://www.ece.umd.edu/DSPCAD/dif
30. The OpenDF dataflow project sourceforge: URL http://opendf.sf.net
31. Wipliez, M., Roquier, G., Nezan, J.: Software code generation for the RVC-CAL language. Springer journal of Signal Processing Systems. Special Issue on Reconfigurable Video Coding (2009)

32. Zima, H., Chapman, B.: Supercompilers for parallel and vector computers. ACM (1991)