

Multiprocessor scheduling of dataflow models within the Reconfigurable Video Coding framework

Jani Boutellier and Victor Martin Gomez and Olli Silvén
Machine Vision Group
University of Oulu, Finland
{jani.boutellier, victor.martin, olli.silven}@ee.oulu.fi

Christophe Lucarz and Marco Mattavelli
Microelectronic Systems Laboratory
École Polytechnique Fédérale de Lausanne, Switzerland
{christophe.lucarz, marco.mattavelli}@epfl.ch

Abstract

The new Reconfigurable Video Coding (RVC) standard of MPEG marks a transition in the way video coding algorithms are specified. Imperative and monolithic reference software is replaced by a collection of interconnected, concurrent functional units (FUs) that are specified with the actor-oriented CAL language. Different connections between the FUs lead to different decoders: all previous standards (MPEG-2 MP, MPEG-4 SP, AVC, SVC) can be built with RVC FUs.

The RVC standard does not specify a schedule or scheduling heuristic for running the decoder implementations consisting of FUs. Previous work has shown a way to produce efficient quasi-static schedules for CAL actor networks. This paper discusses the mapping of RVC FUs to multiprocessor systems, utilizing quasi-static scheduling. A design space exploration tool has been developed, that maps the FUs to a multiprocessor system in order to maximize the decoder throughput. Depending on the inter-processor communication cost, the tool points out different mappings of FUs to processing elements.

1. Introduction

The effort of designing the Reconfigurable Video Coding (RVC) standard is motivated by the intent to describe already existing video coding standards with a set of common atomic building blocks (e.g., IDCT). Under RVC, existing video coding standards are described as specific configurations of these atomic blocks, also known as functional units (FUs). This greatly simplifies the task of designing future

multi-standard video decoding applications and devices by allowing software and hardware reuse across standards.

The functional units are described in RVC with a dataflow/actor object-oriented language named CAL that allows concise description of signal processing algorithms. For this reason, CAL has been chosen as the language for the reference software of the standard. The CAL Model of Computation (MoC) describes decoders as a set of atomic blocks in a way that exposes parallelism between the computations. However, the abstract and high-level CAL models require a systematic implementation methodology and tools to implement these CAL models into real systems. One of the implementation problems is the assignment of RVC FUs to the processing elements (PEs) available in the underlying system, as well as generating efficient schedules for the FU actions.

In previous work [2], a methodology has been designed for transforming RVC CAL networks into a set of homogeneous synchronous dataflow (HSDF) [6] graphs that enable efficient quasi-static scheduling. The work presented in this paper takes as an input the set of HSDF graphs produced by the previous work, and tries to find an optimal mapping of RVC FUs to the PEs in the system. This paper describes a design space exploration (DSE) tool and as an example, shows the mapping of the RVC MPEG-4 Simple Profile (SP) decoder to a multiprocessor system. The number of processors is not limited, but inter-processor communication costs naturally lead to solutions that only have a couple of processing elements. Finally, the results provided by the DSE tool are discussed.

The paper is organized as follows. Section 2 explains the main concepts in the Reconfigurable Video Coding framework. Section 3 explains the used scheduling approach.

Section 4 illustrates the methodology on a real-life application (MPEG-4 Simple Profile decoder). Section 5 concludes the paper.

2. Concepts of the Reconfigurable Video Coding framework

The MPEG RVC framework aims to offer a more flexible and fast path to innovation of future video coding standards. The RVC framework also provides a high level specification formalism that establishes a starting point model for direct software and hardware synthesis. Moreover, the RVC framework intends to overcome the lack of interoperability between various video codecs that are deployed into the market. Unlike previous standards, RVC does not itself define a new codec. Instead, it provides a framework to allow content providers to define a multitude of different codecs, by combining together FUs from the Video Tool Library (VTL). Such a possibility clearly simplifies the task of designing future multi-standard video decoding applications and devices by allowing software and hardware reuse across video standards.

The main strength of RVC is that unlike the traditional video coding standards, where decoders used to be rigidly specified, a description of the decoder is associated to the encoded data, enabling a reconfiguration and instantiation of the appropriate decoder at the video data receiver. Figure 1 illustrates how the decoders can be constructed within the RVC framework. The MPEG RVC framework defines two standards: MPEG-B which defines the overall framework as well as the standard languages that are used to describe different components of the framework, and MPEG-C, which defines the library of video coding tools employed in existing MPEG standards.

MPEG VTL is normatively specified using RVC-CAL. An appropriate level of granularity for blocks within the standard library is important, to enable efficient reuse within the RVC framework. If the library is too coarse, modules will be too large to allow reuse between different codecs. On the other hand, if the granularity is too fine, the number of modules in the library will be too large for an efficient and practical reconfiguration process, and may obscure the desired high-level description and modeling of the RVC decoder. Prior to RVC, reuse of components across applications has been done, *e.g.*, in multi-mode systems [8].

2.1. The CAL model of computation

CAL is a dataflow and actor oriented language that has been recently specified as a subproject of the Ptolemy project [4] at the University of California, Berkeley. The final CAL language specification has been released in December 2003 [3]. CAL models different algorithms by us-

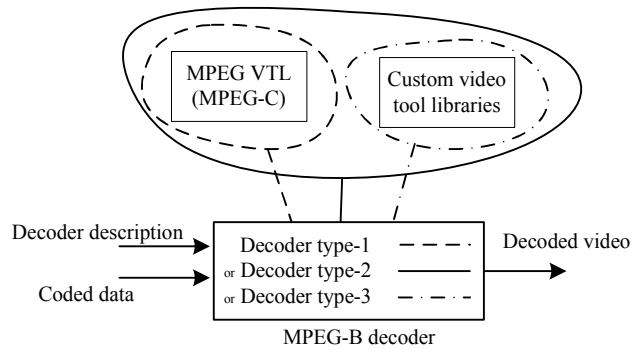


Figure 1. The RVC decoder can be instantiated from standard or custom FUs.

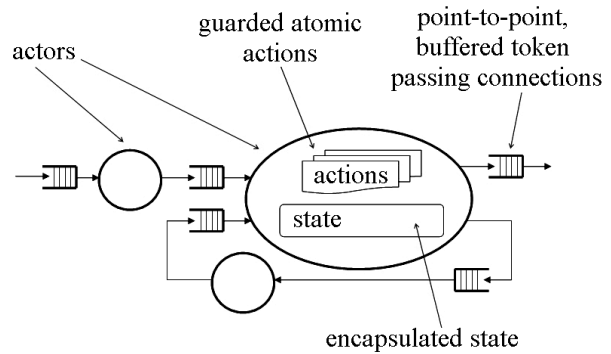


Figure 2. The CAL Model of Computation

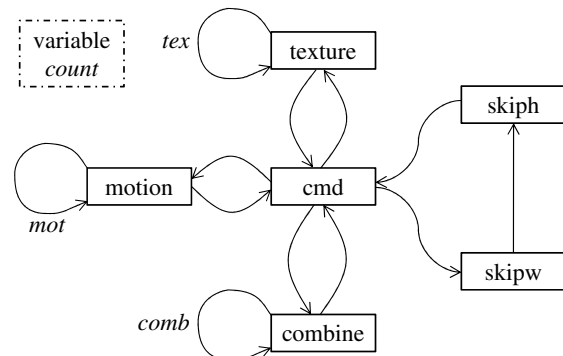


Figure 3. The CAL actor *add* interpreted as an extended finite state machine.

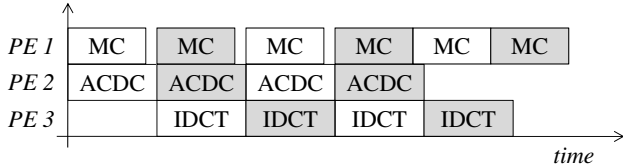


Figure 4. A quasi-static 3-PE macroblock decoding schedule consisting of 6 parts.

ing a set of interconnected dataflow components called actors (see Figure 2).

An actor is a modular component that encapsulates its own state. The state of any actor is not sharable with other actors. Thus, an actor cannot modify the state of another actor. Interactions between actors are only allowed through input and output ports. The behavior of an actor is defined in terms of a set of actions. The operations an action can perform are to 1) consume input tokens, to 2) modify internal state and to 3) produce output tokens. The actors are connected to each other through FIFO channels and the connection network is specified with the Network Language (NL). The action executions within one actor are purely sequential, whereas at the network level, the actors can work concurrently. CAL allows also hierarchical system design: each actor can contain a network of actors.

A CAL actor can also be interpreted as an Extended Finite State Machine (EFSM), and the actions as EFSM state transitions (see Figure 3). The state-space of an EFSM is much greater than that of a regular FSM, because EFSMs (CAL actors) may contain variables. The state transitions in the CAL actors can not take place freely: four types of control mechanisms [7] define which action is going to execute next. These control mechanisms increase the expressiveness of the CAL language, but unfortunately produce an overhead at run-time: actors are constantly checking the status of control mechanisms. The quasi-static scheduling approach used in this work [2] minimizes this run-time overhead by analyzing the CAL network at design-time, leaving only the necessary control mechanisms active at run-time.

3. The scheduling approach

Nowadays, a significant amount of video decoding takes place on mobile devices that have strict power and performance constraints. Thus, also the scheduling used in mobile video decoders must be done efficiently.

In quasi-static scheduling [5], most of the scheduling effort is done off-line and only some infrequent data-dependent scheduling decisions are left to run-time. The off-line determined schedule parts are collected to a repository that is used by the run-time system, which selects en-

tries from the repository on demand and appends them to the ongoing program execution. This approach limits the number of run-time scheduling decisions and improves the efficiency of the system.

Quasi-static scheduling fits very well to the context of video decoding. The video decoding process of hybrid block-based decoders (such as MPEG-4 SP) consists of the decoding of *macroblocks* that consist of several *blocks*. For example, in MPEG-4 SP the blocks are of size 8x8 pixels, and six blocks form a macroblock that produces the information that can be seen on the screen in a 16x16 pixel area. The decoding process varies block-by-block, so the static schedule pieces in quasi-static scheduling should be of a granularity of one block. Furthermore, quasi-static scheduling assumes that the scheduled tasks have been pre-assigned to processing elements at design time, which also improves the scheduling efficiency.

Figure 4 sketches a quasi-static schedule as described above. The decoding of even-numbered blocks is depicted with white tasks in the Gantt chart, whereas the odd-numbered blocks are gray. Blocks 4 and 5 only require Motion Compensation (MC) for decoding, whereas blocks 0 through 3 require also AC/DC prediction and IDCT. The figure simplifies the true computations: in reality each block (MC, ACDC, IDCT) would consist of hundreds of CAL actions. The figure is simplified so far that it only discriminates the tasks on different PEs and different schedule parts. The detailed action schedules that are not shown, are computed at design time and stored for run-time use. The run-time system then selects the appropriate schedule part for decoding each block.

In our previous work [2], we have explained a procedure to transform RVC CAL networks into homogeneous static synchronous data flow (HSDF) graphs that can be quasi-statically scheduled. The quasi-static scheduling algorithm takes the CAL actors and their interconnecting networks as an input, and produces a set of HSDF graphs as an output. The number of produced HSDF graphs depends on the number of *modes* that the CAL network has; the different decoder modes represent the various decoding approaches of 8x8 pixel blocks.

In each produced HSDF graph, one HSDF actor represents an instance of a CAL action. For example, if the RVC *add* (see Figure 3) actor executes the *tex* action 64 times, there will be 64 HSDF actors representing that action, in the HSDF graph. Note, that if a CAL actor is active in several different network *modes* M , the same HSDF actors will appear in each graph that represents those modes M .

The dataflow simulation environment used to run RVC on workstations, is named OpenDF [1]. To verify the functionality of our quasi-static scheduling approach, the OpenDF simulator has been modified so that it forces the simulator to execute the actions in the scheduled order.

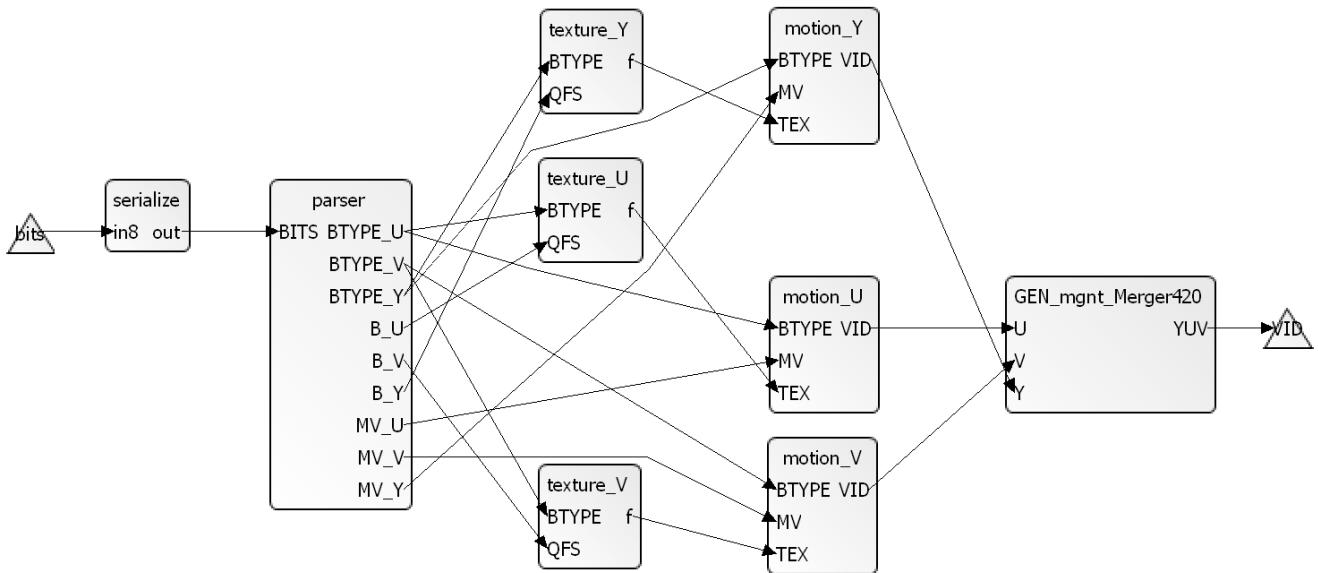


Figure 5. A high-level view of the RVC MPEG-4 SP decoder.

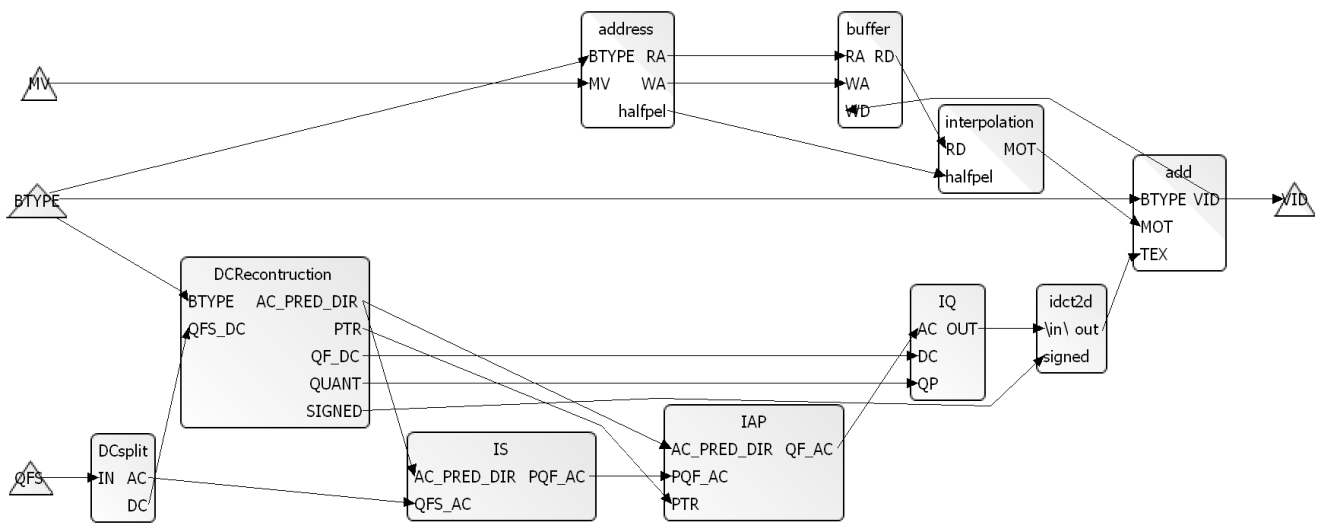


Figure 6. Motion compensation and texture decoding of RVC MPEG-4 SP.

Table 1. Activity of the MPEG-4 SP actors in different operation modes.

FU	New frame	Inter block	ZMV block	Intra block	Hybrid block
Address	x	x	x	x	x
Buffer		x	x	x	x
Interpol.		x	x		x
DCSplit				x	x
DCRec.	x	x	x	x	x
IS	x	x	x	x	x
IAP	x	x	x	x	x
IQ				x	x
IDCT2D				x	x
Add	x	x	x	x	x

Table 2. Number of HSDF actors in the five mode graphs.

New frame	Inter block	ZMV block	Intra block	Hybrid block
6	444	442	345	592

4. Case study: MPEG-4 SP decoder

The behaviour of the MPEG-4 SP decoder (see Figure 5) is controlled to a great extent with the *btype* signal that is created in the *parser* actor and affects the behaviour of the *texture* and *motion* networks that do the main decoding effort. For our work, the *btype* signal was analyzed and it was discovered that it defines five major operations modes for the *texture* and *motion* networks. The combined *motion* and *texture* networks are depicted in Figure 6.

With the information about the different *btype* modes, the MPEG-4 SP decoder was profiled extensively to get the number of CAL actor action executions for each *btype* mode. This profiling produced as a result a list of actor activeness that is depicted in Table 1. The table simplifies the profiling results in the way that any activity in the actor respective to the mode is marked with an *x*, independent of the number of actions executed. The total number of actions executed in each mode is described in Table 2.

4.1. Design space exploration

The focus of this work is to explore the mapping of the MPEG-4 SP actors to a multiprocessor system. The number of mapping alternatives is considerable and requires an automated approach.

We define the task as a design space exploration (DSE)

Table 3. Latencies assigned to non-trivial actions.

Actor	Action	Latency	Freq.
Interpolate	other	2	64
IDCT1D (in IDCT2D)	X	8	16
Transpose (in IDCT2D)	X	8	2

problem that has three parameters: a) mapping of each FU to one of the processing elements and b) determining the priorities between FUs and c) setting the cost of inter-processor communication (IPC). Each FU is constrained to run completely on one PE, but one PE can be responsible for any number of FUs. The number of PEs is not restricted, because the IPC cost naturally limits the feasible number of PEs. The set of PEs is assumed to be fully interconnected.

Pino *et al.* [9] have considered a related problem of *clustering* SDF graphs on multiprocessors. In a clustering problem, an arbitrary graph is given, and the vertices (actors) must be grouped as clusters that are then assigned to processors. The clustering problem is essentially about discovering the sets of vertices that should belong together to one cluster. In our mapping problem the clustering step can be omitted, because the original CAL model defines the clusters: the HSDF actors belonging to one RVC FU form one cluster.

Our design-space exploration software takes as an input the set of HSDF graphs produced by our previous work [2], the number of clock cycles consumed by each CAL actor action, and the IPC cost, which is a simple integer constant. The DSE software searches the combination of FU \Rightarrow PE mappings and FU priorities that produces the minimal combined makespan for all HSDF graphs. The combination of makespans of computed by summing together the makespans produced by scheduling each graph separately.

The priorities between FUs in the design-time scheduling slightly affects the resulting schedule makespans. An optimal solution would be to assign a different priority to each HSDF actor, but that would make the search space unfeasible. Thus, the priorities are assigned to the complete clusters (FUs).

The CAL actions in RVC are generally very fine grained and one action usually only modifies the value of a single pixel. Thus, by default the latency of each CAL action in the MPEG-4 SP network was set to one. A few actions that consume more than one token (pixel) at a time, were assigned longer latencies that are explained in Table 3. At the moment, work is being done to acquire more realistic action latency information that is based on code profiling. In any case, schedules constructed at design-time require the use of worst-case execution times.

The DSE software works in three phases. In the first

phase, the software produces all $FU \Rightarrow PE$ mapping combinations and computes a schedule makespan sum for each mapping. In the second phase, the $FU \Rightarrow PE$ mapping is fixed, and the DSE software generates all FU priority combinations, and computes the schedules makespan sums for each priority combination. In the third phase the $FU \Rightarrow PE$ mapping and FU priorities are both fixed and schedules are generated for visual inspection by Gantt-charts.

It is not guaranteed that this procedure produces absolutely minimal makespans, because only a fraction of the search space is explored. However, we have a reason to assume that the results are fairly close to optimal, because the first phase of the optimization produces considerably larger variations in makespan than the second phase. The makespan variations due to FU priorities are in the magnitude of $\pm 1\%$.

The scheduling of the MPEG-4 SP HSDF graphs is performed by a basic scheduling algorithm that greedily fires HSDF actors as soon as they have enough input tokens to fire. However, the aforementioned FU priorities are observed, so that those actors that belong to a higher-priority FU , are fired before the actors that belong to lower-priority FUs .

The transformation tool that produces the input graphs for our DSE tool, currently generates HSDF graphs of all CAL actors in Figure 6, except IAP (Inverse AC prediction), IS (Inverse scan), DCReconstruction and DCSplit. There are different versions of the IAP actor for luma and chroma block decoding, which is currently not supported by our scheduling tool, although there is no such theoretical restriction. IS, DCReconstruction and DCSplit are not quasi-statically scheduled, because the IS actor executes different actions based on the values of tokens it acquires from the DCReconstruction actor. Thus, quasi-static scheduling of the IS actor (and its predecessors DCSplit and DCReconstruction) would require changing the schedule according to the token values arriving to IS. Whether this is feasible or not, is not sure. Finding this out is a clear direction for future work.

4.2. The results

A separate mapping result was acquired for each IPC cost value that was imposed on the system. When the IPC cost was one, the DSE algorithm mapped the tasks to three processors (A, B and C), such that Address and Buffer actors were of processor A, Interpolation and Add were on processor B and IQ and IDCT2D were on processor C. This can be seen in Table 4 on column one.

When the IPC cost was increased to two, the DSE software found the best optimization result from a 2-PE solution. The functionality that was of PE B with the IPC cost of one, was moved to PE A. Curiously, this mapping reflects

Table 4. $FU \Rightarrow PE$ mappings produced by our DSE tool. A, B and C are PEs.

IPC cost:	1	2	3
Address	A	A	A
Buffer	A	A	A
Interpolate	B	A	A
Add	B	A	A
IQ	C	C	A
IDCT2D	C	C	A
Makespan:	0.56	0.84	1.00

the same motion compensation / texture decoding division that is also present in the original CAL models, but which has disappeared from the HSDF graphs that are used as input to the DSE tool. Finally, when the IPC cost becomes three, a uniprocessor implementation provides the best throughput: everything is computed on PE A.

The last row in Table 4 shows the relative makespan of the mappings, compared to the uniprocessor mapping of the last column. The two-processor solution does not offer a considerable performance advantage over the one-processor solution, which can be explained by the usage frequency of the inverse quantization (IQ) and IDCT operations: only two operation modes out of five invoke actions of these FUs . Figure 7 shows the Gantt-chart of the 2-PE system for the *hybrid block* mode. PE C has a utilization of around 50% in this mode, which is not very good. Other mappings that would balance the 2 PEs better, are made unusable by the high IPC cost of 2 units.

The 3-PE mapping, also visible in Figure 7, reduces the makespan considerably when compared to both 1-PE and 2-PE solutions. However, this mapping is only possible when the IPC cost is low. The results point out that successful mapping of the RVC MPEG-4 SP decoder to a multi-processing system requires low-latency communication between the processing elements in the system.

Figure 7 also shows that the IQ/IDCT functionality in RVC MPEG-4 SP is a bottleneck. PEs A and B have to run idle for a while as they wait for the IQ/IDCT processing element C to finish. Theoretically, the problem lies in the fact that the 2D-IDCT is implemented so that it performs two successive transpose-operations that are fully sequential in the sense that they require all 64 data values to be ready before the computations can start.

5. Conclusion

In this paper we have presented the results of design space exploration for finding the optimal mapping of RVC MPEG-4 Simple Profile decoder functions to a multipro-



Figure 7. 2-PE (top) and 3-PE (bottom) schedules for the *hybrid block* operation mode.

cessor system. Depending on the magnitude of the inter-processor communication cost, the design space exploration software maps the functions to one, two or three processors.

Prior to mapping the functions to processors, the CAL language specification of the RVC MPEG-4 decoder has been transformed to a set of quasi-statically schedulable HSDF graphs. The mapping of functions to processors has been done for these HSDF graphs.

References

- [1] S. S. Bhattacharyya, G. Brebner, J. W. Janneck, J. Eker, C. von Platen, M. Mattavelli, and M. Raulet. OpenDF - a dataflow toolset for reconfigurable hardware and multicore systems. Technical Report BTH-00422, Dept. of Systems and Software Engineering, Blekinge Institute of Technology, 2008.
- [2] J. Boutellier, C. Lucarz, S. Lafond, V. Martin Gomez, and M. Mattavelli. Quasi-static scheduling of CAL actor networks for Reconfigurable Video Coding. *Journal of Signal Processing Systems*, page (accepted with minor revisions), 2009.
- [3] J. Eker and J. W. Janneck. CAL language report. Technical Report UCB/ERL M03/48, UC Berkeley, 2003.
- [4] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, Jan 2003.
- [5] E. A. Lee. *VLSI Signal Processing III: Recurrences, Iteration, and Conditionals in Statically Scheduled Block Diagram Languages*, pages 330–340. IEEE Press, 1988.
- [6] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987.
- [7] C. Lucarz, M. Mattavelli, M. Wipliez, G. Roquier, M. Raulet, J. W. Janneck, I. D. Miller, and D. B. Parlour. Dataflow/actor-oriented language for the design of complex signal processing systems. In *Conference on Design and Architectures for Signal and Image Processing*, pages 168–175, Bruxelles, Belgium, November 2008.
- [8] H. Oh and S. Ha. Hardware-software cosynthesis of multi-mode multi-task embedded systems with real-time constraints. In *CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign*, pages 133–138, 2002.
- [9] J. L. Pino, S. S. Bhattacharyya, and E. A. Lee. A hierarchical multiprocessor scheduling system for DSP applications. In *Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 122–126, 1995.