Automatic generation of synthesizable hardware implementation from high level RVC-cal description

Khaled Jerbi, Mickaël Raulet, Olivier Deforges, Mohamed Abid

AUTOMATIC GENERATION OF SYNTHESIZABLE HARDWARE IMPLEMENTATION FROM HIGH LEVEL RVC-CAL DESCRIPTION

Khaled Jerbi [⋆†] *Mickael Raulet* [†] *Olivier Deforges* [†] *Mohamed Abid* [⋆]

[†]IETR/INSA. UMR CNRS 6164, F-35043 Rennes
email: FirstName.Name@insa-rennes.fr
[⋆]CES Lab. National Engineering school of Sfax
email: FirstName.Name@enis.rnu.tn

ABSTRACT

Data process algorithms are increasing in complexity especially for image and video coding. Therefore, hardware development using directly hardware description languages (HDL) such as VHDL or Verilog is a difficult task. Current research axes in this context are introducing new methodologies to automate the generation of such descriptions. In our work we adopted a high level and target-independent language called CAL (Caltrop Actor Language). This language is associated with a set of tools to easily design dataflow applications and also a hardware compiler to automatically generate the implementation. Before the modifications presented in this paper, the existing CAL hardware back-end did not support some high-level features of the CAL language. The generated HDL language had to be manually transformed to be synthesizable. In this paper, we introduce a general automatic transformation of CAL descriptions to make these structures compliant and synthesizable. This transformation analyses the CAL code, detects the target features and makes the required changes to obtain synthesizable code while keeping the same application behavior. This work resolves the main bottle-neck of the hardware generation flow from CAL designs.

Index Terms— RVC-CAL, Data flow computing, CAL compiler, Automatic transformation

1. INTRODUCTION

Watching and exchanging videos on the Internet or mobile gadgets by billions of people all over the world is the current phenomenon. Moreover, user requirements of high quality are also growing which has caused a noteworthy increase in the complexity of the algorithms of video codecs. To be used, these algorithms have to be implemented on a target architecture that specifies the description language depending on its nature: hardware or software. In this context, CAL Actor Language [1] was introduced in the Ptolemy II project [2] as a dataflow target agnostic language. The MPEG community standardized the RVC-CAL language [3] in the MPEG-RVC (Reconfigurable Video Coding) standard [4]. This standard provides a framework to describe the different functions of a codec as a network of functional blocks developed in RVC-CAL and called actors. Xilinx developed a hardware compiler called OpenForge[1] [5] that generates hardware implementation using an Intermediate Representation (IR) called XLIM. The bottle-neck of hardware generation using OpenForge is the fact that RVC-CAL code generally contains high level structures that are not compatible with this hardware compiler.

On the other hand, an RVC-CAL compiler called *Orcc*[2][6] is in development as a tool that compiles a network of actors and generates several backends: C, LLVM, Java and notably Xlim. Therefore, we proposed to work on the IR of Orcc to add automatic transformations making any RVC-CAL design synthesizable.

This paper presents an automatic transformation of RVC-CAL code to modify the unsupported structures in the IR of Orcc. The transformation tool analyzes the RVC-CAL code and achieves the required transformations to obtain synthesizable code what ever the complexity of the considered actor. In section 2, we explain the main notions of the RVC-CAL language and its functioning structures and mechanisms. The proposed transformation process is detailed in section 3 and finally hardware implementation results of MPEG4 Part2 decoder case study are presented in section 4.

2. BACKGROUND

The execution of an RVC-CAL code is based on the exchange of data tokens between computational entities called *actors*. Each actor is independent from the others since it has its own parameters and finite state machine if needed. Actors are connected to form an application or a design, this connection is insured by FIFO channels. Executing an actor is based on *firing* elementary functions called *actions*. This action firing may change the state of the actor in case of an FSM. An RVC-CAL dataflow model is shown in the network of Figure 1.

[1]http://openforge.sf.net
[2]http://orcc.sf.net

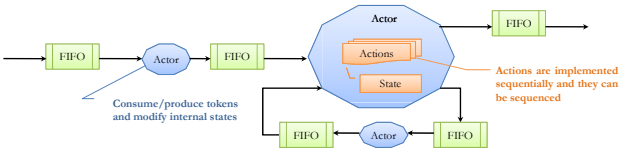

Fig. 1. CAL actor model

The actor execution, so called firing, is based on the Dataflow Process Network (DPN) principle[7] derived from the Kahn Process Network (KPN) [8]. Let Ω be the universe of all tokens values exchanged by the actors and $\mathbb{S}$=Ω* the set of all finite sequences in Ω. We denote the length of a sequence $s \in \mathbb{S}^k$ by $|s|$ and the empty sequence by λ. Considering an actor with m inputs and n outputs, $\mathbb{S}^m$ and $\mathbb{S}^n$ are the set of m-tuples and n-tuples consumed and produced. For example, s_0=[λ,[t_0,t_1,t_2]] and s_1=[[t_0],[t_1]] are sequences of tokens that belong to $\mathbb{S}^2$ and we have $|s_0|$=[0,3] and $|s_1|$=[1,1]. A firing rule is called **multi-token** $iff : \exists e \in |s| : e > 1$ otherwise it is called a **mono-token** rule. The limitation of Openforge is the fact that it does not support multi-token rules which are omnipresent in most actors. Our work consists of automatically transforming the data consumption from multi-token to mono-token while preserving the same actor behavior: firing rules and transitions are detailed below.

2.1. Actor firing

A dataflow actor is defined with a pair $< f, R >$ such as:
* $f : \mathbb{S}^m \rightarrow \mathbb{S}^n$ is the firing function
* $R \subset \mathbb{S}^m$ are the firing rules
* For all $r \in R$, $f(r)$ is finite

An actor may have N firing rules which are finite sequences of m patterns (one for each input port). A pattern is an acceptable sequence of tokens for an input port. It defines the nature and the number of tokens necessary for the execution of at least one action. RVC-CAL also introduces the notion of *guard* as additional conditions on tokens values. An example of firing rule r_j in $\mathbb{S}^2$ is: $\begin{cases} g_{j,k} : [x] | x > 0 \\ r_j = [t_0 \in g_{j,k}, [t_1, t_2, t_3]] \end{cases}$ which means that if there is a positive token in the FIFO of the first input port and 3 tokens in the FIFO of the second input port then the actor will select and execute a fireable action. An action is fireable or schedulable iff :

- The execution is possible in the current state of the FSM (if an FSM exists)
- There are enough tokens in in the input FIFO
- A guard condition returns true

An action may be included in a finite state machine or untagged making it higher priority than FSM actions.

2.2. Actor transition

The FSM transition system of an actor is defined with $< \sigma_0, \Sigma, \tau, \prec >$ where Σ is the set of all the states of the actor, σ_0 is the initial state, $\prec$ is a priority relation and $\tau \subseteq \Sigma \times \mathbb{S}^m \times \mathbb{S}^n \times \Sigma$ is the set of all possible transitions. A transition from a state σ to a state σ' with a consumption of sequence $s \in \mathbb{S}^m$ and a produced sequence $s' \in \mathbb{S}^n$ is defined with (σ, s, s', σ') and denoted:

$$\sigma \xrightarrow[\tau]{s \mapsto s'} \sigma'$$

To solve the problem of the existence of more than one possible transition in the same state, RVC-CAL introduced the notion of priority relation such as for the transitions $t_0, t_1 \in \tau$, t_0 a higher priority than t_1 is written $t_0 \succ t_1$. As explained in [9] a transition $\sigma \xrightarrow[\tau]{s \mapsto s'} \sigma'$ is enabled iff:

$$\neg \exists \sigma \xrightarrow[\tau]{p \mapsto q} \sigma'' \in \tau : p \in \mathbb{S} \wedge \sigma \xrightarrow[\tau]{s \mapsto s'} \sigma'' \succ \sigma \xrightarrow[\tau]{s \mapsto s'} \sigma'$$

This section presented and explained the main RVC-CAL principles. In the next section we present an automatic transformation as a solution to avoid these limitations without changing the overall macro-behavior of the actor.

3. THE PROPOSED METHOD

As shown in Figure 2, our transformation acts on the IR representation of the front-end. The HDL implementation is later generated using the Xlim back-end of Orcc followed by OpenForge.

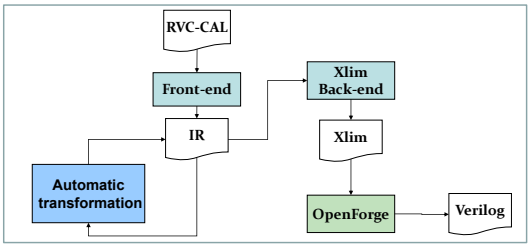

Fig. 2. Automatic transformation localization in Orcc compiling process

3.1. Actor transformation principle

Let us consider an actor with a multi-token firing rule $r \in \mathbb{S}^k$ such as $|r| = [r_0, r_1, .., r_{k-1}]$, this rule fires a multi-token action a realizing the transition $source \xrightarrow[\tau]{a} target$ and $\mathbb{I}$ the

set of all input ports. The transformation creates for every input port an internal buffer with read and write indexes and clips r into a set $\mathbb{R}$ of k firing rules so that :

$$\forall i \in \mathbb{I}, \exists! \rho \in \mathbb{R} : \begin{cases} \rho : \mathbb{S}^1 \rightarrow \mathbb{S}^0 \\ |r| = 1 \\ g_\rho : IdxWrite_i - IdxRead_i \leq sz_i \end{cases}$$

with ρ a mono-token firing rule of an untagged action $untagged_i$, g_ρ is the guard of ρ and sz_i the size of the associated internal buffer defined as the closest power of 2 of r_i. This guard checks that the buffer contains an empty place for the token to read. The multi-token action is consequently removed, and new *read actions* that read one token from the internal buffers are created. While reading tokens another firing rule may be validated and causes the firing of an unwanted action. To avoid the non-determinism of such a case, we use an FSM to put the actor in a reading loop so it can only read tokens. The loop is entered using a $transition$ action realizing the FSM passage $source \xrightarrow[\tau]{transition} read$ and has the same priority order of the deleted multi-token action but has no process. The read actions loops in the read state with the transition $t = read \xrightarrow[\tau]{read} read$. Then the loop is exited when all necessary tokens are read using a *read done* action and a transition to the process state $t' = read \xrightarrow[\tau]{readDone} process \succ t$. The treatment of the multi-token action is put in a *process action* with a transition $process \xrightarrow[\tau]{process} write$. The multi-token outputs are also transformed into a writing loop with *write actions* that store data directly in the output FIFO associated with a transition $w = write \xrightarrow[\tau]{write} write$ and a *write done* action that insures the FSM transition $w' = write \xrightarrow[\tau]{writeDone} target \succ w$.

For example, the actor A of Figure 3 is defined with $f : \mathbb{S}^3 \rightarrow \mathbb{S}^2$ with a multi-token firing rule:

$r \in \mathbb{S}^3 : r = [[t_0, t_1], [t_2, t_3, t_4], [t_5]]$.

```
actor A () int IN1, int IN2, int IN3 ==> int OUT1, int OUT2:
   a: action
   in1:[in1] repeat 2, IN2:[in2] repeat 3, IN3:[in3] ==>
   OUT1:[out1], OUT2:[out2] repeat 2
   do
      {treatment}
   end
end
```

Fig. 3. RVC-CAL code of actor A

Consequently, $|r| = [2, 3, 1]$ which means that there is an action in A that fires if 2 tokens are present in $IN1$ port, 3 tokens are present in $IN2$ and one token is present in $IN3$. The transformation creates the FSM macro-block of Figure 4.

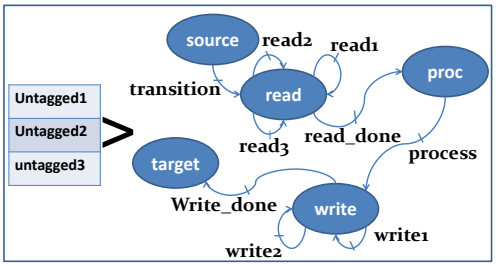

Fig. 4. Created FSM macro-block

3.2. FSM creation cases

We consider an example of an actor defined as $f : \mathbb{S}^3 \rightarrow \mathbb{S}^2$ containing the actions $a1..a5$ such as $a3$ is the only action applying a multi-token firing rule $r \in \mathbb{S}^3$. Creating an FSM only for action $a3$ is not appropriate because $a1, a2, a4, a5$ will be a higher priority which may not be true. The solution is to create an initial state containing all the actions and add the created FSM macro-block of $a3$ (previously presented in Figure 4). The resulting FSM is presented in Figure 5. We

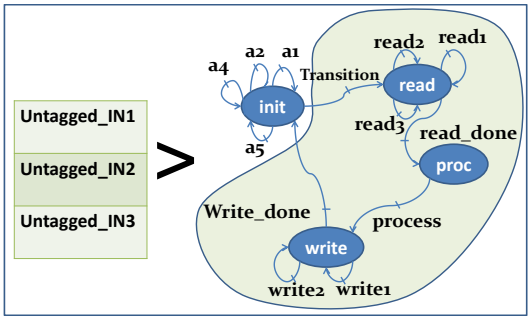

Fig. 5. FSM with created initial state

now suppose the same actor scheduled with an initial FSM as shown in Figure 6. The transition $t = S1 \xrightarrow[\tau]{s \mapsto s'} S2$ is

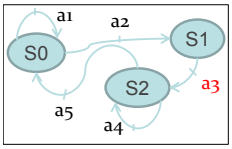

Fig. 6. Initial FSM of an actor

substituted with the macro-block of $a3$ as shown in Figure 7.

4. RESULTS

The achieved automatic transformation was applied on MPEG4 Part2 intra decoder (see design in Orcc Applications [3]) which

[3] http://orc-apps.sf.net

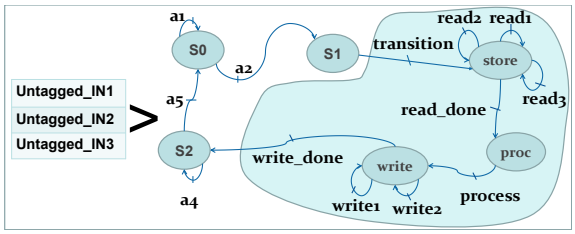

Fig. 7. Resulting FSM transformation

contains 29 actors. The HDL generated code was implemented on a virtex4 (xc4vlx160-12ff1148) and the area consumption results we obtained are presented in table1. After

Criterion	value
Slice Flip Flops	21,624/135,168 (15%)
Occupied Slices	45,574/67,584 (67%)
4 input LUTs	68,962/135,168 (51%)
FIFO16/RAMB16s	14/288 (4%)
Bonded IOBs	107/768 (13%)

Table 1. Design area consumption

the synthesis of the design, we applied a simulation stream of compressed videos. Table 2 below presents the timing results of a 176x144 image size video. The synthesis tool indicates a maximum frequency of 27 MHz. Consequently, we applied it as a frequency of the input data stimulus.

Image size	176x144
Output frequency(MHz)	2
Latency(μs)	328
processing time(ms/image)	11,7

Table 2. Timing results

5. CONCLUSION

This paper presented an automatic transformation of RVC-CAL from high to low level description. This transformation allows avoiding structures that are not understandable by RVC-CAL hardware compilers. We applied this automatic transformation on the 29 actors of MPEG4 part2 video intra decoder and successfully obtained the same behavior of the multi-token design and a synthesizable hardware implementation.

Currently, we are optimizing the transformation to reduce the memory consumption. The transformation process is generalized for all actors. Therefore, improvements are also in progress to customize the transformation depending on the actor complexity analysis. A future work will be the study of the impact of the transformation on the power consumption of the generated implementation.

6. ACKNOWLEDGMENTS

Special thanks for Matthieu Wipliez, Damien De Saint-Jorre and Herve Yviquel for their relevant contribution in the transformation code.

7. REFERENCES

[1] J. Eker and J. Janneck, "CAL Language Report," Tech. Rep. ERL Technical Memo UCB/ERL M03/48, University of California at Berkeley, Dec. 2003.

[2] C. Brooks, E.A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng (eds.), "PtolemyII - heterogeneous concurrent modeling and design in java (volume 1: Introduction to ptolemyII)," Technical Memorandum UCB/ERL M04/27, University of California, Berkeley, CA USA 94720, July 2004.

[3] ISO/IEC FDIS 23001-4: 2009, "Information Technology - MPEG systems technologies - Part 4: Codec Configuration Representation," 2009.

[4] Shuvra Bhattacharyya, Johan Eker, Jrn Janneck, Christophe Lucarz, Marco Mattavelli, and Mickal Raulet, "Overview of the mpeg reconfigurable video coding framework," *Journal of Signal Processing Systems*, vol. 63, pp. 251–263, 2011, 10.1007/s11265-009-0399-3.

[5] Ruirui Gu, Jörn W. Janneck, Shuvra S. Bhattacharyya, Mickaël Raulet, Matthieu Wipliez, and William Plishker, "Exploring the concurrency of an MPEG RVC decoder based on dataflow program analysis," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 19, no. 11, pp. 1646–1657, 11 2009.

[6] Jörn W. Janneck, Marco Mattavelli, Mickael Raulet, and Matthieu Wipliez, "Reconfigurable video coding a stream programming approach to the specification of new video coding standards," in *MMSys '10: Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, New York, NY, USA, 2010, pp. 223–234, ACM.

[7] Edward A. Lee and Thomas M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773–801, 1995.

[8] G. Kahn, "The Semantics of a Simple Language for Parallel Programming," in *Information Processing '74: Proceedings of the IFIP Congress*, J. L. Rosenfeld, Ed., pp. 471–475. North-Holland, New York, NY, 1974.

[9] Johan Eker and Jörn W. Janneck, "A structured description of dataflow actors and its application," Memorandum UCB/ERL M03/13, Electronics Research Laboratory, University of California at Berkeley, May 2003.