Safe Design of Dynamically Reconfigurable Embedded Systems

Xin An, Abdoulaye Gamatié, Eric Rutten

Safe Design of Dynamically Reconfigurable Embedded Systems

Xin An*, Abdoulaye Gamatié**, Éric Rutten*
* INRIA Rhône-Alpes, Grenoble, France ** LIFL/CNRS - INRIA Lille Nord Europe, Lille, France

***Abstract*—Dynamically reconfigurable embedded systems are more and more attractive with the high need to adapt embedded systems regarding frequent environment changes, better execution performances and lower energy consumption. This paper presents an approach for the safe design of these systems. The UML standard MARTE profile is adopted for the design. The resulting models are transformed into formal models (e.g. synchronous programs), by means of which, two levels of analysis are carried out: the system configuration analysis concerning their functional and nonfunctional properties by applying an abstract clock analysis, and the synthesis of a correct controller for system reconfiguration by using discrete controller synthesis to enforce the desired behaviors, and decide and trigger reconfigurations. At last a case study is provided to illustrate our approach. This study is achieved in a co-design framework, referred to as GASPARD2.**

I. INTRODUCTION

Over the recent decades, there has been an increasing requirement for embedded systems to be able to dynamically adapt to their environment variations. Typically, a surveillance embedded system for street observation must adapt its image analysis algorithms according to the luminosity of the weather. The need for adaptivity may also come from the execution platform to provide a better performance or to reduce energy consumption in a system. For instance, for data-intensive algorithms such as the discrete cosine transform, a hardware accelerator gives a more powerful execution in terms of performance than a processor. Further motivations for adaptivity are the need for coping with different protocols and data-coding standards, e.g., in multimedia embedded applications.

Meanwhile, with more and more new features integrated into embedded systems, their design complexity has escalated. For example, in smart phones, multiple applications are made available for users. This leads to a real challenge about cost-effective and safe design methodologies of dynamically reconfigurable embedded systems. Firstly, design correctness issues must be addressed to ensure system reliability in every possible configuration. Secondly, reconfiguration correctness must also be established to safely control the variation between system configurations.

A. Contribution of the paper

We present a methodology for the safe design of dynamically reconfigurable embedded systems. Safety in the design is approached by using formal methods to verify the design, and to generate (part of) the controller triggering reconfigurations. It goes according to the following steps:

- **Step 1: system design using the MARTE profile.** The considered systems are modeled with the UML standard profile MARTE [8] that provides a rich set of concepts for the design of embedded and real-time systems. Both software application and hardware execution platforms are represented with adequate concepts. Then, different software/hardware allocations can be described. For all these aspects of a system, various configurations can be specified and modelled.
- **Step 2: system analysis for correct and efficient execution.** The system models resulting from the previous step are now analyzed from two main points of view: *analysis of each system configuration* and *synthesis of a reconfiguration controller* that enforces the designer-specified system properties. These analyses are achieved based on a transformation of MARTE models towards formal models. Here, we consider the formal tools and techniques provided by the synchronous technology [2]. System configurations are addressed by applying an abstract clock analysis proposed in [1]. A configuration consists of a complete implementation of a system on a given execution platform. The system reconfiguration is dealt with by using a technique allowing one to automatically synthesize a system controller for a given property to be enforced, as in [7]. The main advantage is that the controller is automatically generated and correct w.r.t. the property, using the BZR synchronous language [10].
- **Step 3: composition of initial, uncontrolled design with synthesized controller.** The results obtained from system analysis are now integrated to initial system models in order to define a correct design with respect to system requirements. This is done by composition of the original uncontrolled system and the synthesized controller.

The above methodology is currently under construction and is expected to be set up within a co-design framework, GASPARD2 [5], defined for high-performance embedded systems. A complementary approach to this work concerning the implementation level integration of the generated controller in specific FPGA platforms is proposed in [9].

B. Outline

This paper is organized as follows: Section II presents the design concepts regarding modeling techniques we have adopted for our methodology. Section III describes the analysis

This work is partly supported by the French ANR project FAMOUS – Reference: ANR-09-SEGI-003.

techniques concerning system configuration and reconfiguration aspects respectively. A case study is presented in Section IV to illustrate our proposal. Finally, we conclude our paper in Section V.

II. Design Concepts

A. Modeling of reconfigurable systems with MARTE

We use the UML standard profile for *Modeling and Analysis of Real-Time Embedded systems* (MARTE) [8] to model embedded systems. MARTE offers a rich set of concepts to describe different features of reconfigurable embedded systems. The *General Component Modeling* (GCM) package is used to define general aspects such as algorithms in the application software part of a system. The *Hardware Resource Modeling* (HRM) package is used to describe hardware architecture, e.g. processors and memories. The *Allocation* package serves to define software/hardware mapping. Furthermore, for data-intensive applications such as image or video processing, data-parallel algorithms and multiprocessor execution platforms are described with the *Repetitive Structure Modeling* (RSM) package. All these packages are useful in the description of each system configuration. Concerning reconfiguration modeling, we also need additional features: *Configurations*, which is used to describe different implementation scenarios, or modes, of a system, and UML *Finite State Machines*, which is used to describe configuration switches. All these concepts are available within the Papyrus tool used for modeling in GASPARD2 environment.

B. Modeling each configuration with a synchronous language

The synchronous approach [2] has been proposed in 80s to provide a rigorous mathematical semantics for the safe design of embedded real-time systems. The synchrony hypothesis means that there is a notion of logical instant, like a cycle or a step, within which computation and communication are made. Since then, synchronous languages, e.g. ESTEREL, LUSTRE and SIGNAL have been developed for and widely applied in the area of embedded systems. In our approach, we mainly consider SIGNAL [4], which adopts a multi-clocked philosophy for modeling: the system behaviors are described using relations between the values of observed events, and the occurrences, also referred to as *abstract clocks*, of these events.

A modeling approach from MARTE models to synchronous equational models has already been proposed in [6] in GASPARD2. The resulting models take into account platform and environment constraints on embedded data-intensive applications specified in MARTE. They are therefore translated towards different synchronous dataflow languages such as LUSTRE and SIGNAL.

C. Modeling reconfigurations with the BZR language

BZR [10] is a mixed imperative and declarative programming language, which expresses system behavior in terms of automata (imperative part) and specifies given properties to be enforced by using *contracts* (declarative part). The compilation of BZR automatically synthesizes a controller enforcing safety properties. This controller is then re-injected automatically into the initial BZR program so that an executable program can be generated (in C or Java) for execution. The automata and contracts in BZR can be generated from the state machine and mode switch components specified in MARTE.

III. Design Analysis Techniques

A. System configurations

The validation of system configurations is addressed by using the synchronous models. Properties of interest include functional properties that are addressed with the SIGNAL compiler: syntax and type analysis, data-dependency analysis, abstract clock analysis, and automatic code generation, exploitable in standard simulation environments such as GtkWave. Among these facilities, abstract clocks are very useful for characterizing components interaction within a configuration. Such an interaction quite depends on environment constraints and software/hardware allocation choices. Then, the compiler analyzes abstract clock constraints to check implementability criteria. This is commonly known as clock synchronizability analysis. Further important analysis techniques are available in SIGNAL, such as the temporal performance evaluation [11] achieved by co-simulating a program P and its associated temporal interpretation $T(P)$, i.e. another program obtained automatically from P. The program $T(P)$ computes an approximate execution time of P according to a given system configuration.

B. Enforcing system reconfiguration correctness

Instead of addressing the correctness of reconfiguration by exploring control automata with model-checking, we rather adopt a more constructive method by enforcing reconfiguration correctness with *discrete controller synthesis* (DCS).

DCS is a constructive and automated method ensuring required properties on a system. It exploits transition system models, and is originally defined in supervisory control of discrete event systems. Inputs are partitioned into uncontrollable and controllable ones. The uncontrollable inputs typically come from the system's environment, while the values of the controllable inputs are given by the synthesized controller. It also requires a specification of a control objective which needs to be enforced by control. DCS produces a controller which gives the constraints on controllable events, w.r.t. the current state and the uncontrollable inputs, such that the resulting controlled system satisfies the given objective. The synthesized controller is maximally permissive, and the most possible behaviors are kept. DCS has been defined and implemented as a tool integrated with the synchronous technology: the SIGALI tool [12] and BZR.

IV. Case Study: CM player

A. Informal description

We consider a simple continuous multimedia (CM) player system (modeled in Figure 1) extracted from [14]. The CM server (or the synchronization component) takes as input the

streams of video and audio (CM) data packets from corresponding CM sources, synchronizes and assembles data from several packets into synchronized playable units, calculates the system time at which frames should be played, and dispatches them to output devices (e.g. the screener to play video and the speaker to play audio). The video stream is composed of a sequence of JPEG frames whereas the audio stream is captured in the form of Sparc audio.

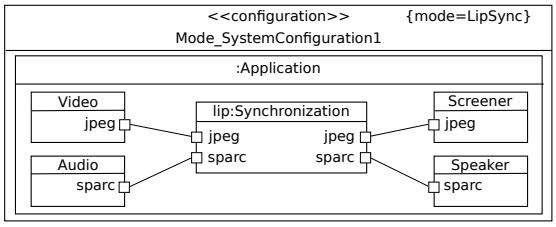

Figure 1. The system configuration: LipSync

Since different temporal relations between media objects can be defined according to application specifications, in this paper, we suppose that the synchronization component has two modes or algorithms dealing with two different temporal relation specifications (taken and adjusted from [3]) respectively as shown in Figures 2 and 3. Both of these specifications are specified by using the reference point synchronization model [3]. Each block of the media is a *logical data unit* (e.g. frames for digital video) defined by the designer. Specially, a LDU is seen as a logical processing unit for media sources.

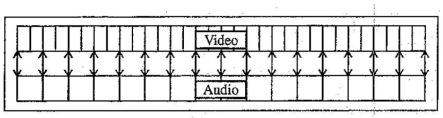

Figure 2. The lip synchronization specification

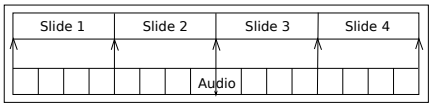

Figure 3. The slide show synchronization specification

As we can see form Figure 2 (resp. 3), the lip synchronization algorithm (resp. slide show synchronization algorithm) each time takes an audio unit and two video units (resp. one slide and four audio units) for processing and assembling a single playable unit.

B. Step 1: Design of the CM Player

1) Modeling of system functionality: With two different execution modes for the synchronization component, we have two functional configurations as modeled using MARTE in Figure 1 and 4 namely *LipSync* and *SlideShowSync*. The stereotype ≪*configuration*≫ is used to represent a specific configuration with the name labeled below, which is associated with the value of the *mode* noted on the top right to indicate when the configuration is active.

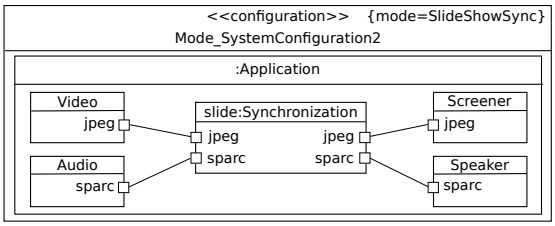

Figure 4. The system configuration: SlideShowSync

2) Modeling of system hardware platform: For the hardware implementation of this simple CM player system, we consider a hardware architecture composed of a processor, a hardware accelerator and memory devices as modeled in [13] (see Figure 5). The CM sources are realized through sensors (e.g. a camera for video data and a microphone for audio data) which have dedicated media conversion units to produce the required data format for further processing. And we assume that each sensor has two different processing frequencies (15 and 45 MHz). A processor and a hardware accelerator, with frequency values 30 and 45 MHz respectively, are provided for the media synchronization processing. And the screener and speaker are realized as actuators, provided with two processing frequencies (15 and 45 MHz) as well.

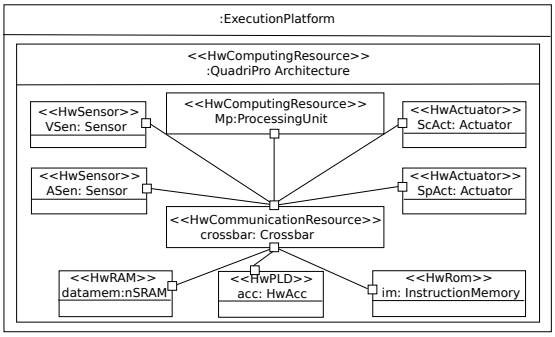

Figure 5. The hardware platform model

The *Hardware Resource Modeling* package of MARTE is used here to describe the architecture. Figure 5 gives the details of our modeling.

3) Modeling of mapping: The mapping of the CM system onto the hardware execution platform consists in allocating each functional component onto the hardware resources. Due to the space limitation, we only give two possible allocation scenarios w.r.t the lip synchronization algorithm, and the *deployment* procedure is also omitted.

Figure 6 depicts the configuration (C1) that the synchronization component is executed on the processor while the media sources/output devices are all mapped to the slower sensors/actuators. Figure 7 shows the configuration (C2) of the mapping scenario with the synchronization component allocated to the hardware accelerator and all the other components allocated to the faster devices. The mappings from software components to hardware resources are noted by dotted lines in the figures.

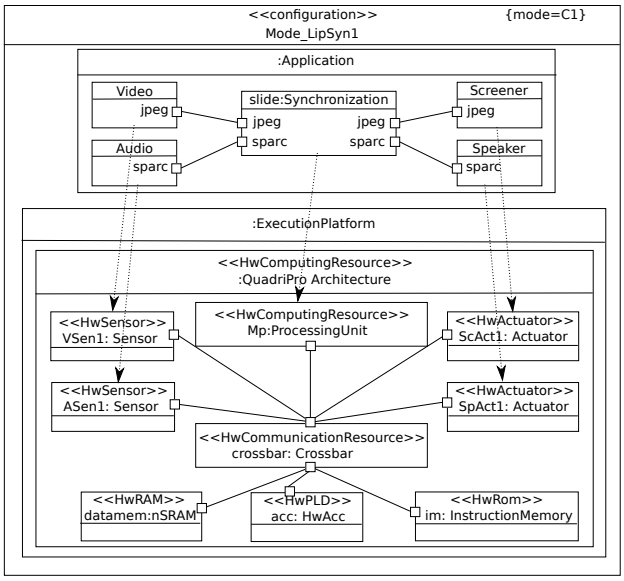

Figure 6. A configuration executed on the processor

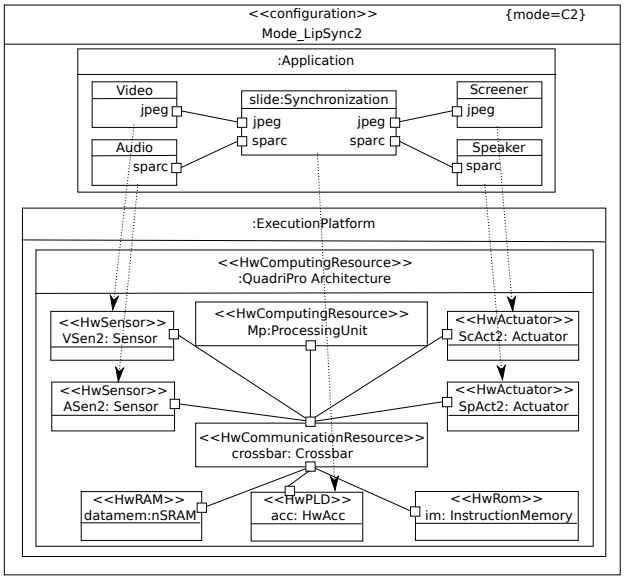

Figure 7. A configuration executed on the hardware accelerator

C. Step 2: Formal Analysis of System Models

1) Configuration Analysis: Due to space limitations, we illustrate only the analysis of configurations w.r.t the lip synchronization mode concerning their correctness, execution time as well as energy consumptions. The method proposed in [1] is employed for the analysis. For simplicity, a single playable unit is considered for the analysis. We firstly identify temporal properties by defining logical clocks for each component, then synthesize a physical clock for each hardware resource, and at last analyze each configuration by mapping logical clocks to physical ones.

Identification of functional temporal properties. The temporal relations between the components are modeled by using a logical binary clock. An occurrence of 1 means the activation of the component whereas 0 means no activation. And at each activation instant (which corresponds to an instruction cycle), one logical data unit is consumed and produced. Figure 8 depicts the activation traces of logical clocks *Vclk, Aclk*, *Sclk*, *SCclk* and *SPclk* corresponding to components video, audio, the lip synchronization, screener and speaker.

Vclk:	1	1	1	1	...	0
Aclk:	1	0	1	0	...	0
Sclk:	0	1	0	1	...	0
SCclk:	0	1	0	1	...	0
SPclk:	0	1	0	1	...	0

Figure 8. The functional temporal property of the system

Synthesis of physical clocks. In order to relate the logical clocks to physical ones, the approach in [1] traces the activation of each hardware resource by computing the period value between two successive processing cycles, e.g. a processor with frequency 30 MHz has the period value equal to approximately 0.033 microseconds, that is 1/(30 MHz). A most frequent clock, called ideal clock, is also defined (by computing the Least Common Multiple) in order to synchronize multi clocks. Figure 9 depicts the the ideal clock and different physical clocks (resp. VSen1Clk, ProClk and HwAClk) associated with the hardware resources (resp. VSen1, processor and hardware accelerator) as well as their relations. The physical clocks ASen1Clk, ScAct1Clk and SpAct1Clk (resp. VSen2Clk, ASen2Clk, ScAct2Clk and SpAct2Clk) associated with ASen1, ScAct1 and SpAct1 (resp. VSen2, ASen2, ScAct2 and SpAct2) have the same clock as VSen1Clk (resp. HwAClk).

Figure 9. Physical clocks of hardware resources w.r.t an ideal clock

Analysis of configurations. We firstly consider the configuration (C1) of Figure 6. We assume the number of cycles executed, at the activation instant of components, by hardware resources is 1. By means of mapping the logical clocks onto the physical ones, we get the result shown in Figure 10. The value -1, whose meaning depends on its nearest preceding value that is not -1, means active when it is 1 and idle when 0. As we can see the logical clock properties are not respected. The constraint between VSen1Clk$'$ and ProClk$'$ is violated, because the first activation of ProClk5$'$ happens earlier than the second activation of VSen1Clk$'$. That is, the video sensor is activated not frequently enough while the processor does not get the data required for its processing.

Then, we consider the configuration (C2) in Figure 7, for which the hardware accelerator is used for the synchronization process whereas both media sensors use the faster frequency value 45 MHz. Figure 11 gives the result of this mapping. This time the temporal properties are respected, and the execution time, which is pointed out by the empty triangle, is 7 ticks (or $(7-1) \times 0.011$ microseconds). However, in consideration of

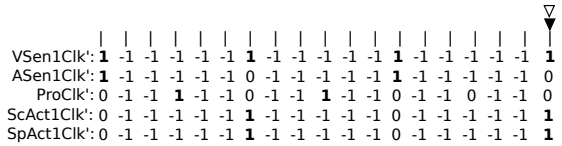

Figure 10. Mapping trace for C1 shown in Figure 6

energy consumption minimization, the slack time, defined by the difference of the task deadline (pointed out by the black triangle) and execution time, is big (12 ticks for each hardware resource). Thus, we consider another configuration (C3) that replaces the accelerator and actuators in C2 with the processor and slower actuators and analyze it in the same way (see Figure 12). As a result, it respects the temporal properties, takes 19 ticks for the execution time, and has a relatively small slack time (12 ticks for two sensors and 9 ticks for the processor).

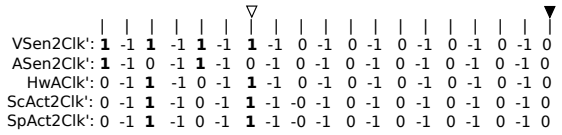

Figure 11. Mapping trace for C2 shown in Figure 7

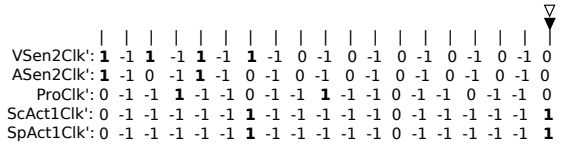

Figure 12. Mapping trace for C3

Similarly, all possible configurations of the system can be analyzed, and meaningful information, e.g. correctness, execution time, can be collected.

2) Reconfiguration Analysis: We take into account the previous three configurations C1, C2 and C3 for the LipSync mode and a correct configuration for the SlideShowSync mode C4 to describe our reconfiguration analysis.

Specification of configuration controller. Having a number of possible configurations/modes for the system, the UML *Finite State Machine (FSM)* is used to model and manage all these configurations as well as their switches. As shown in Figure 13, the FSM specifies how mode values are produced for selecting configurations of the system. It has four states corresponding to the four configurations we have mentioned above. Each state is associated with a specific mode value and the active configuration is the one having the same mode value of the current state of the controller FSM.

We assume that the controller has two inputs: mode switch and the current energy level (**H**igh or **N**ot **H**igh). As we can see, the designed controller controls that the system chooses the correct mode for the synchronization processing, and in each mode, it behaves according to the current energy level. Controllable variables (i.e. ctr1, ..., ctr6) are also defined in

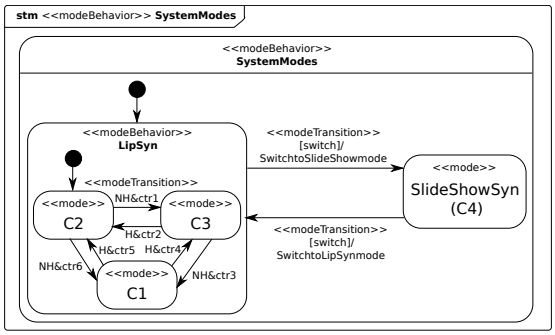

Figure 13. The FSM for system configurations

the controller in order to provide controllable points for the DCS tool to enforce system requirements as shown next.

Enforcing system requirements. We firstly encode the designed configuration controller described in Figure 13 into BZR (see Figure 14). Meanwhile, the analysis results for each configuration, i.e. its correctness, execution time and energy consumption, from Section IV-C1 are also associated with each state. In the program, we simply use the amount of ticks of execution time and slack time as the values of the execution time and energy consumption.

```
automaton
state Lip
do
    automaton
    state C2
    do  correctness = true;
        execution_time = 7;
        energy_consumption = 60;
    until not energy_high & ctr6 then C1
    | not energy_high & ctr1 then C3

    state C1
    do  correctness = false;
        execution_time = 19;
        energy_consumption = 0;
    until energy_high & ctr5 then C2
    | not energy_high & ctr4 then C3
    state C3
    do correctness = true;
    execution_time = 19;
    energy_consumption = 33;
    until energy_high & ctr2 then C2
    | not energy_high & ctr3 then C1
    end

until swtch then Slide

state Slide
do
    correctness = true;
    execution_time = 10;
    energy_consumption = 46;
until swtch then Lip
end;
```

Figure 14. The BZR program for the control automaton

The system constrains are defined by using BZR contracts [10]. Figure 15 gives the code of the contract description of the system requirement that the system always stays away from incorrect configurations, and meanwhile, the execution time constraint, whose limit is defined as a constant (e.g. 19 ticks), must also be respected.

```
contract
var time_constraint:bool;
let
    time_constraint = (execution_time<=19);
tel
assume true
enforce (correctness & time_constraint)
with (ctr1,ctr2,ctr3,ctr4,ctr5,ctr6:bool)
```

Figure 15. The BZR contract enforcing system requirements

At last, by feeding the BZR program and the contract to the BZR compiler, it will synthesize a controller (if it exists) automatically satisfying the system requirement. Figure 16

depicts the simulation results of this case study with respect to different execution time limits (i.e. 19 ticks for the left simulation and 10 ticks for the right one).

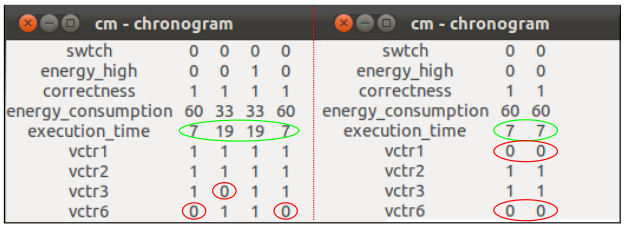

Figure 16. The simulations of controlled system w.r.t the execution time limit being 19 and 10 ticks respectively

Let's firstly look at the simulation on the left, for which all three configurations satisfy the execution time limit 19 ticks. Initially, the system is in state C2, and when the energy level is low (depicted as *energy_high* equal to 0 in the figure), the controller inhibits the system from going to state C1, by setting the value of the controllable variable ctr6 to 0, which is an incorrect configuration. The same happens when the system is in state C3. As a result, the system stays in configurations C2 and C3. When the execution time limit is set to 10 ticks, for which only C2 satisfies the execution time limit. As we can see from the simulation on the right, the controller forbids all transitions by setting corresponding controllable variables to 0 and makes the system stay in configuration C2.

D. Step 3: Composition of synthesized controller

Finally, the synthesized controller generated by BZR is integrated into the initial system model (as shown in Figure 17). It gives the values of the controllable variables w.r.t the current system state and input such that the resulting controlled system satisfies the system requirements.

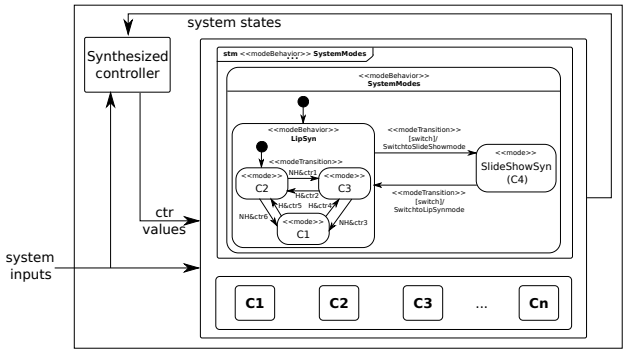

Figure 17. The final CM player model

V. Conclusions

In this paper, we propose a methodology for the safe design of dynamically reconfigurable embedded systems. Such systems are getting an increasing attention due to the crucial needs to address the frequent evolution of embedded systems, regarding requirements from their environments or execution platform in terms of performance and energy consumption. The UML standard profile MARTE has been adopted for the high-level modeling of system configurations (i.e. functionality, execution platform and their allocation). The resulting models are analyzed by using the approach proposed in [1], which employs abstract clocks inspired by the synchronous approach [2], for a fast and qualitative analysis about functional and nonfunctional properties of system configurations via the analysis of scheduling traces resulting from the mapping of logical clocks (capturing functional properties) onto physical ones (derived from hardware processing frequencies). The DCS formal technique is then used for synthesizing a correct controller enforcing reconfiguration correctness. The domain specific language BZR [10] has been adopted for this purpose. At last a CM player case study has been given to illustrate our approach.

References

[1] Adolf Abdallah, Abdoulaye Gamatié, and Jean-Luc Dekeyser. Correct and energy-efficient design of SoCs: the H.264 encoder case study. In *International Symposium on System-on-Chip (SoC'2010)*, Tampere Finland, 2010.

[2] Albert Benveniste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, Robert, and De Simone. The synchronous languages 12 years later. In *Proceedings of The IEEE*, pages 64–83, 2003.

[3] Gerold Blakowski and Ralf Steinmetz. A media synchronization survey: reference model, specification and case studies. *IEEE journal on selected area in communications*, 1996.

[4] Abdoulaye Gamatié. *Designing embedded systems with the* SIGNAL *programming language: synchronous, reactive specification*. Springer New York, 2010.

[5] Abdoulaye Gamatié, Sébastien Le Beux, Éric Piel, Anne Etien, Rabie Ben Atitallah, Philippe Marquet, and Jean-Luc Dekeyser. A model driven design framework for high performance embedded systems. Research Report RR-6614, INRIA, 2008.

[6] Abdoulaye Gamatié, Éric Rutten, Huafeng Yu, Pierre Boulet, and Jean-Luc Dekeyser. Synchronous modeling of data intensive applications. Research Report RR-5876, INRIA, 2006.

[7] Abdoulaye Gamatié, Huafeng YU, Gwenaël Delaval, and Éric Rutten. A case study on controller synthesis for data-intensive embedded systems. In *Embedded Software and Systems*, pages 75–82, 2009.

[8] OMG Group. Modeling and analysis of real-time and embedded systems (marte). www.omgmarte.org/. 2007.

[9] Sébastien Guillet, Florent de Lamotte, Éric Rutten, Guy Gogniat, and Jean-Philippe Diguet. Modeling and formal control of partial dynamic reconfiguration. In *Proceedings of the 6th International Conference on ReConFigurable Computing and FPGAs,* Cancun, Mexico, 2010.

[10] Gwenaël Delaval, Hervé Marchand, and Éric Rutten. Contracts for modular discrete controller synthesis. In *Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems*, pages 57–66, Stockholm Suède, 2010.

[11] Apostolos Kountouris and Paul Le Guernic. Profiling of SIGNAL programs and its application in the timing evaluation of design implementations. In *IEE Colloquium on the Hardware-Software Cosynthesis for Reconfigurable, HP Labs, Bristol, UK*, 1996.

[12] Hervé Marchand, Patricia Bournai, Michel Le Borgne, and Paul Le Guernic. Synthesis of discrete-event controllers based on the SIGNAL environment. *Discrete Event Dynamic Systems*, 10:325–346, 2000.

[13] Imran Rafiq Quadri, Abdoulaye Gamatié, Pierre Boulet, and Jean-Luc Dekeyser. Modeling of configurations for embedded system implementations in MARTE. In *1st workshop on Model Based Engineering for Embedded Systems Design - Design, Automation and Test in Europe (DATE 2010)*, Dresden Germany, 2010.

[14] Lawrence A. Rowe and Brian C. Smith. A continuous media player. In *Network and Operating System Support for Digital Audio and Video*, pages 376–386. Springer Berlin / Heidelberg, 1993.