Abstract Clock-based Design of a JPEG Encoder

Adolf Abdallah, Abdoulaye Gamatié, Rabie Ben Atitallah and Jean-Luc Dekeyser

***Abstract*—This paper presents the design and analysis of multimedia applications such as the JPEG encoder on multiprocessor architectures. Abstract clocks are considered to deal with the correctness of system behaviors and to find the most suitable execution platform configurations regarding performance and energy consumption. We claim that our approach offers a rapid and reliable design space analysis, which is crucial when implementing complex systems.**

***Index Terms*—JPEG, abstract clock, SoC, design, analysis.**

I. Introduction

Multimedia embedded systems implemented on system-on-chip (SoCs), e.g., found in digital cameras and cellular phones, are omnipresent in our daily life. They become increasingly sophisticated and resource demanding to meet the quality of service. Their developers must guarantee their correctness and satisfactory execution performances. In addition, energy consumption is another major issue when such systems are embedded in portable devices depending strongly on batteries. All these aspects make the efficient design of multimedia SoCs very challenging. Existing commercial tools provide RTL (register transfer level) simulation and emulation environments for low-level system prototyping [1]. Unfortunately, RTL level tools cannot adequately support the complexity of multi-processor SoCs required for multimedia applications because they are very slow for a meaningful execution of application software. In order to reduce simulation time, many research efforts have been conducted towards the use of Cycle-Accurate (CA) simulators. At a higher abstraction level, an Instruction Set Simulator (ISS) sequentially executes instructions without any notion of concurrency of a micro-architecture.

In this paper, we propose an approach [2] for correct and efficient design of a JPEG encoder at a high abstraction level so as to reduce the design space exploration efforts in a codesign framework [3]. We consider abstract clocks inspired by the synchronous approach [4] to assist a designer in the choice of system configurations offering a good compromise about correctness, performance and energy consumption. Our approach is typically very useful for platform-based design [5]. Contrarily to RTL and CA based techniques, our approach does not require any coding to run and analyze a system. Thus, it eliminates the related tedious debugging efforts.

II. Clock-based system modeling

In this study, we consider a JPEG encoding of a finite sequence of images. Such an encoding algorithm consists of five tasks (or components) [2]: `rgb`, `dct`, `qu`, `hu` and `re` representing respectively color transformation, discrete cosine transformation, quantization, huffman coding and image reconstruction. Activation rate relations between these components are defined by using an abstract clock notion. Each component is associated with a *finite* abstract clock clk, defined as a sequence of logical instant values: an occurrence of 1 means the component is active and is *simultaneously*[1] *consuming, executing and producing data* (i.e. synchrony hypothesis [4]), while 0 means no activation.

clk_{rgb}:	1	1	1	1	1	...	0	0	0	0
clk_{dct}:	0	1	1	1	1	...	1	0	0	0
clk_{hu}:	0	0	1	1	1	...	1	1	0	0
clk_{qu}:	0	0	0	1	1	...	1	1	1	0
clk_{re}:	0	0	0	0	1	...	1	1	1	1

Fig. 1. Trace of functional clocks associated with JPEG tasks.

Fig. 1 describes a pipelined execution of the JPEG encoder algorithm, where the *functional* clocks clk_{rgb}, clk_{dct}, clk_{hu}, clk_{qu} and clk_{re} are associated respectively with `rgb`, `dct`, `hu`, `qu` and `re`. The trace reflects the precedence relations about components activations. Note that more complex execution models can be easily captured by such traces [6], [7]. In the rest of the paper, we refer to such relations as *functional clock properties* of the encoder. The trace shown in Fig. 1 captures a multi-clock behavior that is typically specified with *polychronous* formalisms like Signal [8] or CCSL [9].

We consider a *parallel random access machine* [10] with shared memory to execute the JPEG encoder. Processor frequencies are assumed to be modifiable, at least within a range of values as in PowerPC or ARM Cortex-A8 processors. Let us consider five processors P_1, P_2, P_3, P_4 and P_5 with the initial frequencies $f_1 = 25MHz, f_2 = 200MHz, f_3 = 100MHz, f_4 = 50MHz$ and $f_5 = 40MHz$ respectively. We use their period values $1/f_i$ to define their activation instants. For synchronization purpose, we consider a reference (or ideal) clock κ providing a common time base. We define the period of κ as $1/LCM(f_1, ..., f_5) = 0.005\mu s$, where LCM is the Least Common Multiple. Fig. 2 depicts the periodic activations of all processors according to their periods and κ. We refer to these activations as *physical* clocks κ_j of processors.

To capture mapping and scheduling choices with abstract clocks, we define a projection of functional clocks on physical clocks. The number c of processor cycles corresponding to each task activation is pre-profiled and known statically according to a target processor.

a) Clock projection of clk_i on κ_j: Assuming the number of instants in a functional clock clk_i, i.e. its length, is smaller than that of a physical clock κ_j, their projection is as follows:

A. Abdallah, A. Gamatié and J.-L. Dekeyser are with LIFL/CNRS, Inria, 40 avenue Halley, 59650 Villeneuve d'Ascq - France.

R. Ben Atitallah is with University of Valenciennes, Le Mont Houy, 59313 Valenciennes - France.

[1]Actually, an active logical instant corresponds to several processor cycles, i.e., it is not "instantaneous". In our approach, this number is determined during the mapping of functionality on a platform.

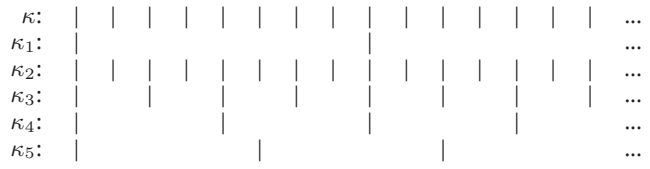

Fig. 2. Trace of physical clocks associated with processors.

1) for each $k \in [1, length(clk_i)]$, the status (i.e., 0 or 1) of the k^{th} instant of clk_i is mapped onto the j^{th} instant of κ_j, where $j = (nb\_occ(clk_i, k, 0)+(nb\_occ(clk_i, k, 1)\times c) + 1)$. The expression $nb\_occ(clk_i, k, 0)$ returns the number of instants with the status 0 in clk_i that have occurred before the k^{th} instant whereas $nb\_occ(clk_i, k, 1)$ concerns instants with the status 1. The constant c is the number of processor cycles performed by an activation. Let $L = length(clk_i)$, every k^{th} instant of κ_j s.t. $(k > nb\_occ(clk_i, L, 0) + nb\_occ(clk_i, L, 0) \times c)$ is associated with the status value 0. Let us call clk'_i the clock resulting from this step;
2) in clk'_i, the value -1 is inserted at all empty positions where the reference clock κ has an instant occurrence while clk'_i has not any instant occurrence. Also, $c - 1$ values of -1 are inserted after each occurrence of a 1 to delimit a whole activation. After this step, clk'_i and κ have the same length.

From the above clock projection, the occurrence of 1 at an instant in a clock clk'_i indicates the processor is executing. The value 0 indicates that the processor is in the *Nop* state. The meaning of -1 is rather contextual: when a sequence of such a value is immediately preceded by 1, then it denotes *potentially active at those instants*; otherwise, it denotes *idle*. In this projection, the delay for inter-task communications or context switches is neglected for simplicity reasons even though in modern architectures, it can be higher than computation time. A way to take it into account is to consider a uniform distribution of an overhead over all task activations by incrementing all c's by some given number of cycles. Another way that is better consists in introducing communication actors in our specifications, which would consume some delay.

b) Scheduling of tasks on processors: We distinguish several task mapping scenarios on an execution platform:

- *Mono-task scheduling on each processor.* Each processor executes one task (itself executed only on this processor). For each task associated with a functional clock clk_i, its scheduling on a processor associated with a physical clock κ_j is captured by a clock projection according to a number of cycles c corresponding to each task activation. An illustration is given in Fig. 3(a) where we map `rgb`, `dct`, `qu`, `hu` and `re` onto processors P_1, P_2, P_3, P_4 and P_5 respectively. For the sake of simplicity, the very simple values $c_{rgb} = 1$, $c_{dct} = 4$, $c_{qu} = 2$, $c_{hu} = 3$ and $c_{re} = 3$ are associated with `rgb`, `dct`, `qu`, `hu` and `re` respectively. The arrows represent activation precedences specified in the functional part after scheduling.
- *Multi-task scheduling on processors.* Here, a processor can execute several tasks. To define such a scheduling, we apply a periodic time slicing to the functional clocks clk_i of all tasks to be executed on a processor with a physical clock κ_j [2]. Such a scheduling is also used in time-triggered protocols [11] for automotive systems.

Beyond the above two schedulings, one may also need to *schedule either multiple tasks on multiple processors, or one task on several processors*. These last cases are solved in a similar way as above. The overall scheduled behavior is periodic as for regular static scheduling techniques applied in SDFs [12] or loop scheduling for software pipelining [13].

III. Clock-based system analysis

We assess the design choices resulting from the previous section, w.r.t. the correctness of functional clock properties, performance and energy consumption.

c) Analysis of functional clock properties: It is important to preserve the functional clock relations when synthesizing a scheduling of a system after a mapping. Typically, a data consuming component must not be executed without having received its required data. In Fig. 3 (a), the constraint between clk'_{rgb} and clk'_{dct} is violated since the second (resp. the third) activation instant in clk'_{rgb} is preceded by the second (resp. the third) activation instant in clk'_{rgb}.

To solve this issue, a first solution consists in delaying the activation of the fastest clock so as to postpone the execution of its associated processor. In Fig. 3 (b), by inserting 8 logical instants of the reference clock in clk'_{dct} and 6 instants in clk'_{qu}, the activation precedences become correct. The black and gray triangles respectively indicate deadline and specific response times. Another solution consists in modifying the frequency in order to satisfy the functional clock properties. Let us reduce the frequency of P_2 from $200MHz$ to $100MHz$. We obtain the trace in Fig. 3 (c) where the functional clock properties are satisfied between clocks clk'_{rgb} and clk'_{dct}.

d) Evaluation of execution time: To compute the total execution time of the system for performance evaluation, we have to determine the amount of computational workload for each processor by using the execution clock traces resulting from task scheduling. The number C of activation cycles executed by a processor during a complete execution of an application is determined by the length of its associated clock trace until the last active instant. If this execution is performed at a frequency f, the corresponding execution time is given by: $\frac{C}{f}$. In Fig. 3 (c), it takes 9 cycles in clk'_{dct} to execute the first two activations. Hence, the corresponding execution time by processor P_2 at $100MHz$ is $\frac{9}{100} = 0.09\mu s$.

e) Minimizing energy consumption: During an execution, we define the *slack time* of a task as the difference between its completion time and its associated deadline. In Fig. 3(b), the deadline value is pointed out by the black triangle on the 26^{th} instant of the reference clock. The response time of `rgb` task is pointed out by the gray triangle on the 21^{th} instant. In Fig. 3(c), this response time occurs at the 25^{th} instant after a reduction of the associated processor frequency. As a result, the slack time is shorter, which also reduces the energy consumption, while the functional properties are still verified. The consumed energy E can be estimated quantitatively by

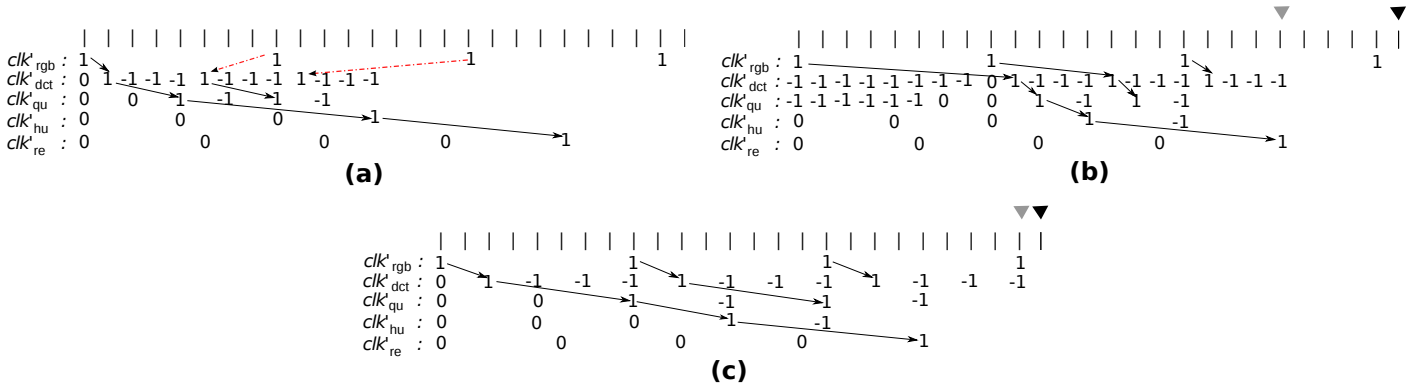

Fig. 3. Trace sketch of clocks clk'_{rgb}, clk'_{dct}, clk'_{qu}, clk'_{hu} and clk'_{re} resulting from a mono-task scheduling on each processor (cycles per task activation: $c_{rgb} = 1$, $c_{dct} = 4$, $c_{qu} = 2$, $c_{hu} = 3$ and $c_{re} = 3$), with: (a) $f_1 = 25MHz, f_2 = 200MHz, f_3 = 100MHz, f_4 = 50MHz$ and $f_5 = 40MHz$ (precedence constraints violated); (b) $f_1 = 25MHz, f_2 = 200MHz, f_3 = 100MHz, f_4 = 50MHz$ and $f_5 = 40MHz$ with a delay of eight logical instants of the ideal clock for clk'_{dct} and six logical instants for clk'_{qu} (precedence constraints satisfied) and (c) $f_1 = 25MHz, f_2 = 100MHz, f_3 = 50MHz, f_4 = 50MHz$ and $f_5 = 40MHz$ (precedence constraints satisfied).

computing the product of the execution time T calculated previously with power consumption W information, obtained from a profiling on experimental platforms: $E = T \times W$.

Another way to minimize energy consumption is to switch off processors temporarily when their assigned tasks are finished. In particular, it is very interesting when the slack time is very long. However, switching on a processor has some cost in terms of delay and energy, which is necessary to bring the processor in a stable state for a new execution. This can increase the overall cost if it happens very frequently. Since our clock analysis is applied on macro-periods (per image) for repetitive algorithms, we choose the slack time reduction.

IV. Validation on JPEG encoder

To validate our clock-based design and analysis, we compare our results with those obtained with the cycle-accurate (CA) simulation in SystemC. CA simulation is largely used in industry and is provides good performance accuracy. However, it is time consuming. We use the SoCLib library [14], which provides an MPSoC simulation environment at CA level.

Configurations ID	Number of Proc.	Allocation type
1	1	task/processor
2	2	task/processor
3	3	task/processor
4	4	task/processor
5	5	task/processor
6	2	sub-image/processor
7	3	sub-image/processor
8	4	sub-image/processor
9	5	sub-image/processor

TABLE I
Mapping configurations for the JPEG encoder.

We consider nine mapping configurations of the JPEG encoder, summarized in Table I. In configurations 1, 2, 3, 4 and 5, the JPEG tasks are distributed according to the number of processors corresponding to a pipelined execution. For configurations 6, 7, 8, and 9, processors carry out the same algorithm on different image blocs corresponding to single program multiple data (SPMD) execution model.

Task	# Cycles	Power (mW)
rgb	7534	3741
dct	7214	3776
qu	6920	3858
hu	5504	3675
re	4594	3653

TABLE II
Cycles and power consumption per task activation in JPEG.

Table II gives input pre-profiling data for the JPEG, used in our clock-based approach. It shows measures of processor cycles and power consumption per task activation for an image, on a RISC processor of type PowerPC 405 CPU Core, at 300MHz (using the Power Analyzer N6705A of Agilent). The measured execution time and power are average values obtained from several real-board experiments using 100 images of bmp type and of different sizes.

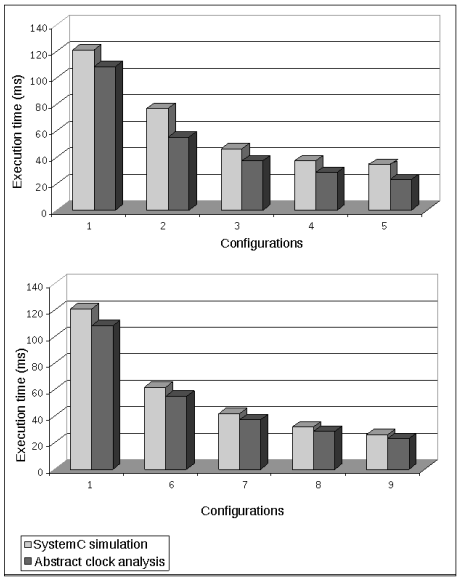

Fig. 4. Comparison of execution time according to the two approaches.

Fig. 4 shows the experimental results of an evaluation of

execution time throughout the clock based analysis and CA simulations in SystemC. The results obtained from the clock-based analysis keep the same tendency as those observed in SystemC. This provides a designer with a consistent basis to assess the different configurations.

We consider a rate of 15 frames per second for the encoding. Then, a duration of $1/15 \approx 66ms$ is obtained for encoding each image. Let us consider this duration as an image processing deadline. We can determine the minimal frequencies based on the clock-based analysis so as to reduce the slack time as much as possible (see [2]). We can also estimate quantitatively the energy consumption. For that purpose, given the execution time evaluated for each task with the clock-based approach, we compute the energy dissipation in configurations 1, 6, 7, 8 and 9 as shown in Fig. 5. This is achieved by accumulating the energy consumption of all tasks via the product of their corresponding pre-determined power consumption (see Table II) and execution times. We observe again that our approach leads to the same tendency as the considered CA simulation.

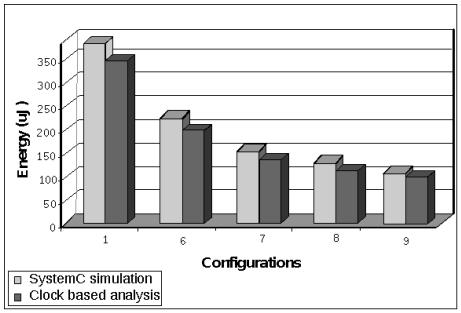

Fig. 5. Comparison of energy consumption to the two approaches.

Concerning the accuracy of our clock-based approach compared to CA simulation (which is more precise), the maximum estimation error is of 28% for execution time (Fig. 4) and of 11.6% for energy consumption (Fig. 5). Indeed, this difference is explained by the fact that the clock-based analysis does not fully take into account synchronizations overhead in multiprocessor system as well as additional activities that are intrinsic to parallel processing such as shared data communication overheads which are accurately evaluated with SystemC simulator. On the other hand, since the SystemC implementation and CA simulation requires a significant time (about one week to implement and achieve all experiments) compared to the clock-based technique (half a day), the latter approach is therefore preferable for a preliminary rapid exploration of large design spaces. Finally, our clock-based analysis is applicable modularly to any periodic clock trace independently from the number of clocks in the trace. Since multimedia applications have generally regular (repetitive) behaviors, they can be characterized with periodic clock traces. In that sense, our proposition is scalable.

The clock analysis exposed in this paper has been achieved with a preliminary *ad hoc* implementation combining the TimeSquare [9] clock simulator, and a library of Ocaml functions ($\approx$ 200 lines) allowing a user to address: execution clock trace construction, functional clock properties, execution time and slack time estimation. More details about these functions are given in [7]. Timesquare is considered for its user-friendly visual interface to simulate and display the periodic clock traces analyzed with Ocaml functions.

V. Concluding remarks

This paper illustrated the design and analysis of a JPEG encoder on a multiprocessor hardware architecture. Starting from a high-level modeling, we proposed an analysis approach by considering abstract clocks inspired by the synchronous approach [4], for a fast reasoning about functional correctness and best design choices regarding temporal performance and energy consumption. We addressed the energy consumption by reasoning on clock traces expressing correct execution of system functionality on given architecture configurations. We estimated the energy consumption for some software/hardware mapping configurations. While the clock-based reasoning is less accurate than a cycle accurate simulation in SystemC, it provides very similar observations in a faster way.

We have started an improvement of the clock-based approach by taking into account synchronization and communication overheads. The aim is to make the analysis more accurate and reduce the precision gap with lower level simulations. A new complete and seamless tool is under development in place of the previous implementation of our approach.

References

[1] B. Bailey, G. Martin, and A. Piziali, *ESL Design and Verification: A Prescription for Electronic System Level Methodology*. Elsevier, 2007.

[2] A. Abdallah, A. Gamatié, R. Ben-Atitallah, and J.-L. Dekeyser, "Correct and Energy-Efficient Design of a Multimedia Application on SoCs," INRIA, Report 7715, 2011, http://hal.inria.fr/inria-00616223_v1.

[3] A. Gamatié, S. L. Beux, E. Piel, A. Etien, R. Ben-Atitallah, P. Marquet, and J.-L. Dekeyser, "A model driven design framework for high performance embedded systems," INRIA, Research Report 6614, 2008, http://hal.inria.fr/inria-00311115/en.

[4] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, "The synchronous languages twelve years later," *Proc. of IEEE*, vol. 91, no. 1, pp. 64–83, Jan. 2003.

[5] A. Sangiovanni-Vincentelli, L. Carloni, F. De Bernardinis, and M. Sgroi, "Benefits and challenges for platform-based design," in *41st annual Design Automation Conference (DAC'04)*, 2004.

[6] X. An, E. Rutten, and A. Gamatié, "Safe design of dynamically reconfigurable embedded systems," in *Workshop on Model Based Engineering for Embedded Systems Design (M-BED'2011)*. ECSI, 2011.

[7] A. Abdallah, "Conception de SoC à Base d'Horloges Abstraites : Vers l'Exploration d'Architectures en MARTE," THESE, Université des Sciences et Technologie de Lille - Lille I, Mar. 2011. [Online]. Available: http://tel.archives-ouvertes.fr/tel-00597031/en/

[8] P. Le Guernic, J.-P. Talpin, and J.-C. Le Lann, "Polychrony for system design," *J. of Circuits, Systems, and Computers*, vol. 12, no. 3, 2003.

[9] C. André and F. Mallet, "Specification and verification of time requirements with CCSL and Esterel," in *Proceedings of LCTES 2009*. ACM, 2009, pp. 167–176.

[10] M. Forsell, "On the performance and cost of some PRAM models on CMP hardware," in *International Parallel and Distributed Processing Symp. (IPDPS'08)*, Miami, Florida, USA, 2008, pp. 1–8.

[11] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[12] S. S. Bhattacharyya, "Compiling dataflow programs for digital signal processing," Ph.D. dissertation, EECS Department, University of California, Berkeley, 1994. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/1994/2589.html

[13] V. Van Dongen, G. Gao, and Q. Ning, "A polynomial time method for optimal software pipelining," in *Parallel Processing: CONPAR 92-VAPP V*, ser. LNCS, L. Bougé, M. Cosnard, Y. Robert, and D. Trystram, Eds. Springer Berlin / Heidelberg, 1992, vol. 634, pp. 613–624.

[14] "The SoCLib project: An open modelling and simulation platform for system on chip design," http://soclib.lip6.fr/.