A timing constraints control technique for embedded real time systems

Mouna Ben Saïd, Kaïs Loukil, Nader Ben Amor, Mohamed Abid
CES Laboratory
ENIS National School of Engineers
Sfax, Tunisia
mouna.bensaid@ieee.org

Jean Philippe Diguet
Lab STICC
UBS University
Lorient, FRANCE

Abstract— the real-time applications have a growing complexity and size which have to be well controlled. They can be viewed as a set of synchronized tasks, communicating and sharing critical resources. One of the main difficulties in the real-time application design is time constraints meeting. All tasks have to be running before their predefined deadlines. At this level, the integration of real time operating systems (RTOS) in the real-time systems design flow is necessary to enable scheduling tasks and managing the competition between them with respect of timing constraints. One of the problems encountered here is that one task may have different execution times. It may exceed its predefined WCET and then its deadline for many reasons. The problem is that one deadline exceeding may cause subsequent constraints violations which may disrupt the functioning of the system. This paper deals with this particular issue. It presents a new technique that permits the monitoring of tasks under execution. It controls their timing constraints by means of watchdog concept and detects deadline missing. That information is used to tune the target application parameters in order to satisfy timing constraints for the further computation iterations. We have implemented this technique in the RTOS MicroC/OS-II using the EDF scheduling policy. This technique has been validated using an Altera FPGA prototyping platform and the 3D rendering application.

I. Introduction

The real-time execution is one of the main constraints for embedded systems. In fact these systems are often used in applications where time is a strong constraint and are called real-time (RT) systems. A RT system is not a system that responds as fast as possible to requests, but responds correctly within the time constraints imposed. Then, the information to be processed by a RT system should mainly arise neither too early nor too late so that they can be effectively exploited by the system [1]. A very significant example is the video transmission system. This system must enable the sending and receiving of digitized video at a rate of 25 or 30 images per second for a good quality of service. The video must go through a series of stages including image and sound digitalization, video compression, transmission and reception, and synchronizing sound and image. All these processing steps must be carried out sequentially while meeting frame rate constraint. Here is where we talk about RT constraints whose respect is a major concern in real-time systems design. It is common to differentiate between RT systems by their temporal constraints' level of criticism. There are hard real-time systems, where all timing constraints have to be respected and soft RT systems, which accept a certain rate of temporal errors unless they have catastrophic consequences. This latter class includes, among others, multimedia systems [2], which are subject of the present work.

The real-time applications have a growing complexity and size. In order to facilitate their design, implementation, scalability and conviviality, they are designed in a multitasking context. They can be viewed as a set of synchronized tasks, communicating and sharing critical resources. At this level, the use of a real-time operating system (RTOS) is a necessity. In fact, an RTOS helps control the real-time applications' complexity by managing the chaining and competition between tasks with respect of timing constraints [3]. One of the problems encountered here is the variable execution time of the same task, which may exceed its predefined WCET (Worst Case Execution Time); the largest number of CPU cycles assigned to a task. Many reasons are behind the WCET exceeding such as variable system calls frequency, hardware interrupts, variable types of processed data and inaccurate WCET estimation. The problem is that one deadline exceeding may cause subsequent constraints violations which may disrupt the functioning of the system (with different levels of criticism).

It is important to set up online monitoring techniques to detect such problems and correct them. In [10] a feedback control real-time scheduling is presented. Feedback control real-time scheduling defines error terms for schedules, monitors the error, and continuously adjusts the schedules to maintain stable performance. In this technique, deadline missing problem is resolved using different CPU allocation time and there is no exploitation of the application parameters. In [11], a middleware Control Framework is presented. It enhances the effectiveness of QoS adaptation decisions by dynamic control and reconfiguration of internal parameters and functionalities of distributed multimedia applications. This technique is complex and is not dedicated for embedded systems with insufficient resources.

In this paper, we propose a technique that permits the monitoring of tasks executions within a RT system in order to control their timing constraints meeting. It permits the detection of timing constraints violations to be able to act on the tasks' temporal parameters to solve the problem. We present also its integration in the open source RTOS MicroC/OS-II for which we have added a second real-time scheduler which takes into account timing constraints of real-time tasks. We developed a test application of 3D object rendering under NIOS II IDE. It uses the RTOS MicroC/OS-II and runs on a prototyping platform based on Altera's FPGA. The present paper is organized as follows: in section II, we present an overview of the control technique. Section III is devoted to the RTOS modification and the EDF scheduling algorithm implementation which is necessary to set up our monitoring technique. Section IV presents the experimental results obtained on an FPGA board and using 3D image synthesis application.

II. CONTROL TECHNIQUE OVERVIEW

We start by defining the timing parameters considered in this work that we also call OS parameters. Then we give a description of the proposed technique's structure.

A. *System timing parameters*

In this work, we consider a configuration of periodic tasks. A periodic task [4] launches its instances in regular time intervals called periods. It is characterized by the following timing parameters:

- Period P,
- Deadline D, onto which system timing requirements are usually mapped. It is a task maximum response time,
- Execution time Te, as an average measurement,
- Worst case execution time WCET.

A task may begin execution at different moments and have different execution times, but it must finish before its deadline. The timing parameters must be defined with respect of the following relationship:

$0 \leq Te \leq WCET \leq D \leq P$.

The scheduling of a set of periodic tasks is cyclic and the sequence is repeated similarly every a study period that we call the system hyper-period "H". It is the lowest common multiple of all tasks periods [5]. To meet time constraints, all tasks have to be executed at every period. During the system "H", a task Ti with a period Pi shall be executed ni=H/Pi times. So we must assign each task Ti an execution time Tei so that:

$$\sum_{i=1}^{n} \frac{H}{Pi} * Tei < H \qquad (1)$$

Where n is the number of tasks

In this way we ensure that all tasks normally run within the system "H".

B. *Complete approach overview*

Our control technique runs at both application and OS layers. It is used to supervise the respect of real time constraints while tasks are executing in a real time system. Fig. 1 gives an overview of this control technique.

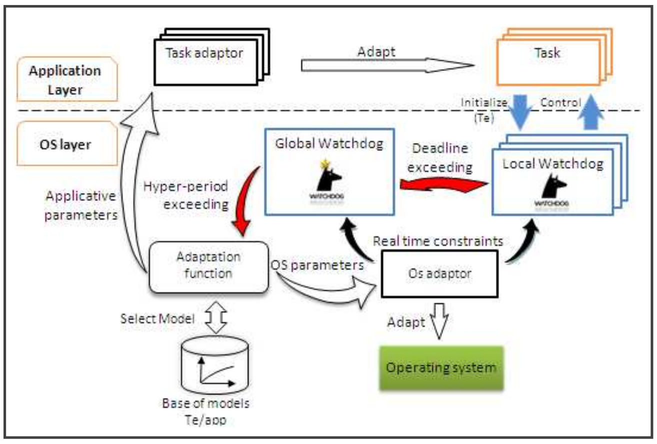


Figure 1. Control technique structure

Our technique is based on two types of 'w': a local 'w' which controls execution at task level, and a global 'w' which controls the execution at system level, over the system "H". We assign a local 'w' to each task in order to supervise its execution and maintain its state for each period (such as number of execution times and end of execution). There is a communication between tasks and their corresponding watchdogs. Each task begins by creating and initializing the watchdog. Once launched, each 'w' is responsible for monitoring its task's execution. A local 'w' also communicates with the global 'w' to submit any deadline exceeding detection alerts. Then the global 'w' saves this event.

When the "H" expires, the global 'w' analyzes the list of events to see whether they affect the overall system execution. If there is no "H" overtaking, the system keeps on with the unchanged set of application parameters. But if it detects a "H" exceeding, it determines its origin, and then it calls an adaptation function. This function makes an adaptation decision according to the overtaking. It chooses a model of Te/application parameters (depend on the target application) using a pre-established model table. It selects the adequate model to solve the problem of overtaking. Then it sends the new parameters values forming the selected model to task and OS adapters so that they modify their current parameters.

C. *Scheduling algorithm selection*

In a conventional multitasking embedded RTOS, tasks are scheduled using a fixed priority based policy where deadlines are not taken into account. Such a policy raises a problem: the highest priority task may deprive less critical tasks from CPU time. Whereas, in order to meet real time constraints all tasks have to finish running before their deadlines. In fact, one task deadline exceeding may cause a delay of all tasks which may disrupts the overall functioning of the system. It is then obvious that such a policy is inappropriate for real-time systems. These systems require using scheduling policy that guarantees good distribution of CPU time among competing tasks and considers the timing constraints of real-time tasks. For this reason, we were brought to seek more adequate scheduling algorithm which ensures that all tasks run during their periods. The scheduling policy that was chosen is EDF, Earliest deadline first [6] (Liu and Layland 1973). It is a preemptive scheduling algorithm used in real-time systems. It belongs to the class of dynamic priority algorithms where priority is assigned according to deadline: task with the nearest deadline is the highest priority task. EDF is also an optimal algorithm; it produces a feasible schedule for all feasible tasks sets. Next section gives a brief explanation of scheduling feasibility.

D. *Schedulability analysis*

Let's consider a set of tasks and their time constraints. The schedulability analysis is the study of tasks' feasibility using the chosen scheduling policy. In other words, it is the fact of checking that every task always meets its deadline. For the EDF policy, we make a schedulability analysis by performing a feasibility test based on the system overall load and following the two theorems below [7]:

Theorem 1 (Lui and Layland, 73):

Any set T of periodic and synchronous tasks (i.e. ∀i, $A_{i,1} = 0$), and whose deadlines are equal to periods (i.e. ∀k, Di = Pi) is feasible under EDF if and only if:

$$U1 = \sum_{Ti \in T} \frac{Tei}{Pi} \leq 1 \qquad (2)$$

Where:

Ai,1 is the arrival date of the 1st instance of task Ti

Di is its deadline

Pi is its period

Tei is its execution time

In case where Di < Pi, (3) is a sufficient but not necessary condition.

$$U2 = \sum_{Ti \in T} \frac{Tei}{Di} \leq 1 \qquad (3)$$

Theorem 2: EDF optimality [Dertouzos, 74]

If there is a policy that leads to a feasible scheduling, then the scheduling is also feasible under EDF.

III. EXPERIMENTATION

The implementation of the technique already described requires two steps: real-time scheduling policy establishment, then control technique implementation.

A. *RTOS selection*

The development environment used in this work is Nios II IDE. It allows complex system design around the NIOS II processor. It serves for the implementation and execution of the system software part whether with simulation or on card. This environment supports the real time operating system MicroC/OS-II allowing the implementation of applications under this RTOS. Since we have chosen to work with the EDF scheduler, two solutions for the choice of RTOS are possible: the first is to add an EDF scheduler to the RTOS MicroC/OS-II since its original scheduler is fixed priority based. The second is to seek another RTOS already with EDF scheduler and then get engaged in the complex modification of the HAL (Hardware Abstraction Layer) in order to configure and port it on our platform. Since the source code of MicroC/OS-II is open and available with all the documentation, we have adopted the first solution.

B. *EDF scheduler implementation in MicroC/OS-II*

To implement the EDF scheduler in the target kernel, we must start by checking the initial conditions which are kernel pre-emption and task periodicity. The first condition is verified by the kernel which is already preemptive. However, concerning the second one, MicroC /OS-II supports only a-periodic tasks. Hence we need to establish periodic tasks management first. Then we move to the implementation of the scheduling policy.

1) *Establishment of task periodicity management*

It's true that MicroC/OS-II doesn't manage tasks periodicity, but it offers time management services (delay a task for some time to expire, resume a delayed task) functioning at the rate of a system clock [8]. This clock triggers a periodic interrupt called "system tick" at a frequency fixed at baseline (usually between 10 and 100 Hz). The ISR associated with the system clock interruption is called "OSTimeTick()". During this routine, MicroC/OS-II decrements the delays of waiting tasks till expiration, then makes them ready to run again.

In order to integrate task periodicity management in the kernel MicroC/OSII, we exploited time management services and the system clock tick interrupt described above. We added a data structure in the user definable Task Control Block (TCB) extension in order to support task period. A timer is associated with each task period and decremented at every clock tick. Once it expires, it is automatically reset and the task is made ready to run.

2) *Scheduler implementation*

Then we extended once more the TCB to support the other tasks temporal parameters (deadline, WCET and execution time). Initially, the deadline and the WCET must be defined offline.

Periodic tasks activation and deadlines update are charged to an internal function called OSTimeTick(). This function is itself triggered periodically by the system ticker ISR. Whenever a scheduling event occurs (ticker ISR, task finishes or is delayed, new task released, etc.) the TCB list will be searched for the task closest to its deadline. This task will then be scheduled for execution next.

To get the highest priority task, we implemented two solutions. The first solution is browsing all deadlines at each switching event to determine the nearest one. The second one consists in arranging the deadline structures in a linked list sorted by ascending order of deadline. The smallest deadline will then be in the top of the list. This solution is more optimized in terms of execution time.

Fig. 2 shows the TCB extended data structure (Task User Data) used for the EDF scheduling following the second solution.

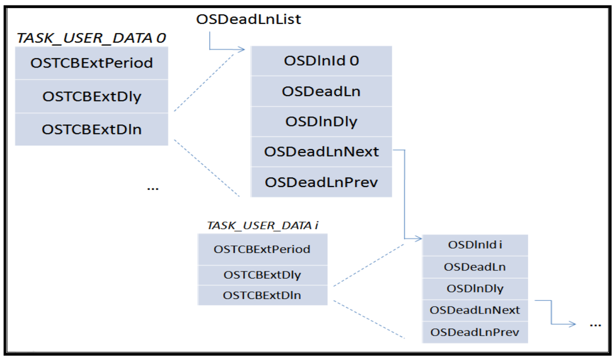


Figure 2. Tasks list structure after scheduling policy establishment

C. 3D test application implementation

We have validated our control technique on a complex multimedia application: the 3D rendering application. This application has a variable execution time due to its different algorithmic parameters. These parameters are the number of polygons representing the 3D object and the type of shading algorithm (flat shading and Gouraud shading). We proceeded first to the reformulation of the source code so that it can handle different 3D objects at once instead of one global object. Then, using MicroC/OSII services, we rewrote the application as a set of tasks cooperating to generate a 3D scene. We chose a simple scenario of four tasks cooperating to generate a 3D rendering of 2 overlapping 3D objects. These tasks are: Init, Anim1, Anim2 and Mixing.

D. Execution time computation

The HAL system clock tick provides low degree of accuracy of time intervals (10-100Hz) which is not useful when tasks are short. That's why we used a higher resolution counter, called "timestamp timer" which can be handled via HAL timing functions described in [9], to obtain time information. However, since the HAL only supports one timestamp timer in the system, and we work in a multitasking context with a preemptive kernel, the use of only this tool is not sufficient in our case. Thus we opted for a method that combines the use of the "timestamp timer" and some of the MicroC/OS-II services. We performed our execution time measurements through the hook function OSTaskSwHook() [8] which is called by the kernel whenever a context switch occurs. When the task is switched in, we start the counter running. Then, when switching to another task we retrieve the current value of the timestamp counter which is the execution time of the task being switched out. We use an accumulator to save the task total execution time.

E. Added system calls

Being a modular system, MicroC/OS-II offers the possibility to select the services we need for a particular application. The selection is done graphically using the NIOS II IDE (the configuration file is generated automatically). That's why we were brought to define some new system calls in order to configure the added services. We tried to limit the number of these calls in order to ease the application developer task. We defined five system calls which are summarized in Table I.

TABLE I. THE ADDED SYSTEM CALLS

System call	Service and location
OSTaskOver()	Indicates the end of one task iteration (a task is defined as an infinite loop) At the end of task before delaying it.
OSTaskInitExt()	Initializes task's timing constraints just before the task's creation.
PHP_Start()	Synchronizes tasks' periods with "H". Called after tasks' creation.
OSInitExt()	Essentially allows enabling the control technique and selecting the scheduler. The options are OS_SCHED_EDF and OS_SCHED_PRIOR_BASED. Joins the call of OSInit() in the main function.
OSInitControl()	Performs nitializations necessary for the new control mechanism. Joins the call of OSInit().

IV. TEST AND VALIDATION

This section is divided into two parts. The first presents the validation of the EDF scheduling policy. The second part is the validation of the control technique using the implemented scheduler. The time values used in the test scenarios are in number of system clock tick (100Hz).

A. Scheduling policy validation

We define in Table II the OS parameters that will be used for the EDF scheduling policy validation.

TABLE II. OS PARAMETERS FOR SCHEDULER VALIDATION

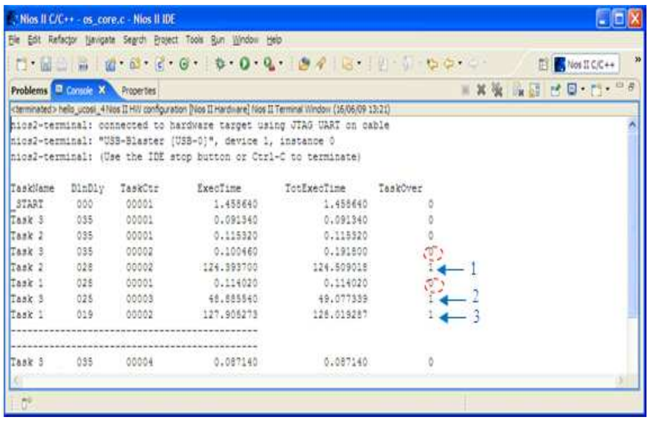
OS Parameters	Task 1	Task 2	Task 3
WCET	13	13	8
Deadline	35	35	30
Period	40	40	40
Nb exe/H	1	1	1

If we apply the feasibility test (2) to this scenario, we find:

$U1 = (13 + 13 + 8) / 40 = 0.85 < 1$

Hence our system is feasible.

Task3 is the highest priority task since it has the earliest deadline. It communicates with Task1 and Task2 via "Message Queue"; it is pending to messages from both of them. Fig. 3 shows the order of tasks' execution under the EDF scheduling policy with an illustrative example of preemption.


Nios II C/C++ - os_core.c - Nios II IDE
File Edit Refactor Navigate Search Project Tools Run Window Help
Nios II C/C++
Problems Console Properties
1
2
3

```
<terminated> hello_ucosii_4 Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (16/06/09 13:21)
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

TaskName   DlnDly   TaskCtr       ExecTime        TotExecTime      TaskOver
_START      000      00001        1.458640          1.458640           0
Task 3      035      00001        0.091340          0.091340           0
Task 2      035      00001        0.115320          0.115320           0
Task 3      035      00002        0.100460          0.191800           0
Task 2      028      00002      124.393700        124.509018           1
Task 1      028      00001        0.114020          0.114020           0
Task 3      025      00003       48.885540         49.077339           1
Task 1      019      00002      127.905273        128.019287           1
-----------------------------------------
-----------------------------------------
Task 3      035      00004        0.087140          0.087140           0
```

Figure 3. EDF scheduler validation

Being the highest priority task, Task3 begins to run first. Then it switches to the "pending" state letting the CPU control to the next highest priority task which is Task2 (because it is placed before task1 in the ready list). The indicator "TaskOver" shows the moments of preemption of Task1 and Task2 (Taskover = 0) by Task3 when this latter intercepts a message. It also illustrates the order of termination of tasks which allows verifying the proper functioning of the scheduler.

MicroC/OS-II's fixed priority based scheduling time is always constant irrespective of the number of created tasks. It is calculated of about 11µs. As for the EDF scheduler presented in this paper, it has an execution time proportional to the number of ready tasks in the system. The overhead is about 7µs per ready task.

B. *Control technique validation*

1) *Complete test case*

The complete test scenario used for the control technique validation is given in Table II. It is composed of 3 tasks: T. Anim1, T. Anim2 and T. Mixing We pass through three steps of testing. The first step presents normal execution of tasks, i.e. with respect to all temporal constraints defined initially. The next step shows the detection of one task deadline violation without influence on the overall functioning of the system. The last step shows the detection of "H" exceeding, which requires a modification of the application and OS parameters, namely an adaptation of the system.

TABLE III. OS PARAMETERS FOR TEST

OS Parameters	T. Anim1	T. Anim2	T. Mixing
WCET	15	15	20
Deadline	50	50	55
Period	60	60	60
Nb exe/H	1	1	1

The system "H" is equal to 60.

If we apply the feasibility test (2) to this scenario, we find:

U1= (15 + 15 + 20) / 60 = 0.83

If we calculate also the sufficient condition for schedulability (3), we find:

U2 = 15/50 + 15/50 + 20/55 = 0.96

According to U1 and U2 calculations, we ensure that there will be no deadline exceeding if the WCET values are respected.

2) *Scenario 2: deadline exceeding*

Following the execution of the previous scenario, the CPU time taken by the task Anim_1 has increased in the third "H" because of the change of processed data (increase of the number of polygons). It exceeds the WCET. The resulting tasks execution times are shown in Table III.

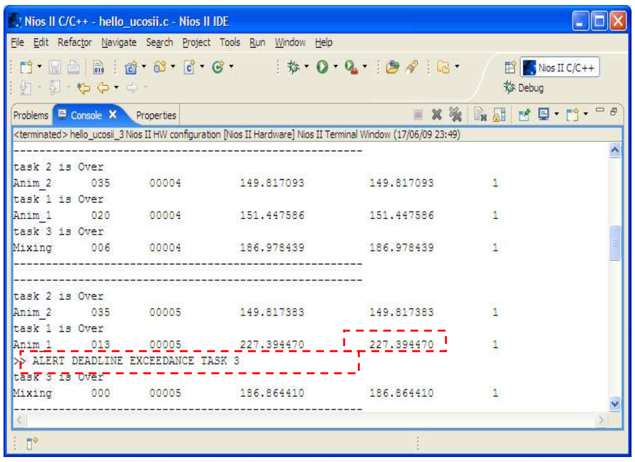
TABLE IV. TASKS' EXECUTION TIME FOR THE FIRST SCENARIO

OS Parameters	T. Anim1	T. Anim2	T. Mixing
Te	23	15	20

U1 = (23 + 15 + 20) / 60 = 0.96

U2 = 15/50 + 23/50 + 20/55 = 1.12

According to these calculations, we find that a running time of 23 for task Anim1 causes a deadline exceeding (U2> 1) without affecting the overall functioning of the system (U1 <1). This theoretical result has been proved practically by the illustration of tasks execution in Fig. 4.


Nios II C/C++ - hello_ucosii.c - Nios II IDE
File Edit Refactor Navigate Search Project Tools Run Window Help
Nios II C/C++
Debug
Problems Console Properties

```
<terminated> hello_ucosii_3 Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (17/06/09 23:49)
------------------------------------------------------
task 2 is Over
Anim_2      035      00004        149.817093         149.817093        1
task 1 is Over
Anim_1      020      00004        151.447586         151.447586        1
task 3 is Over
Mixing      006      00004        186.978439         186.978439        1
------------------------------------------------------
------------------------------------------------------
task 2 is Over
Anim_2      035      00005        149.817383         149.817383        1
task 1 is Over
Anim_1      013      00005        227.394470         227.394470        1
>> ALERT DEADLINE EXCEEDANCE TASK 3
task 3 is Over
Mixing      000      00005        186.864410         186.864410        1
------------------------------------------------------
```

Figure 4. Illustration of deadline exceeding detection

The local 'w' associated with Task 3 detects a deadline overtaking. It triggers a deadline exceeding alert which is then saved in the exceeding event list.

3) *Scenario 2: Hyper-period exceeding*

In this latter scenario, the execution time of task Anim1 increases again to reach about 36 clock ticks (Table IV)

TABLE V. TASKS' EXECUTION TIME FOR THE SECOND SCENARIO

OS parameters	T. Anim1	T. Anim2	T. Mixing
Te	36	15	20

U1 = (36 + 15 + 20) / 60 = 1.18

U2 = 36/50 + 15/50 + 20/55 = 1.38

Both of U1 and U2 exceed 1, causing this time an exceeding of the deadline then the period. Fig. 5 shows the execution graph of the second scenario.

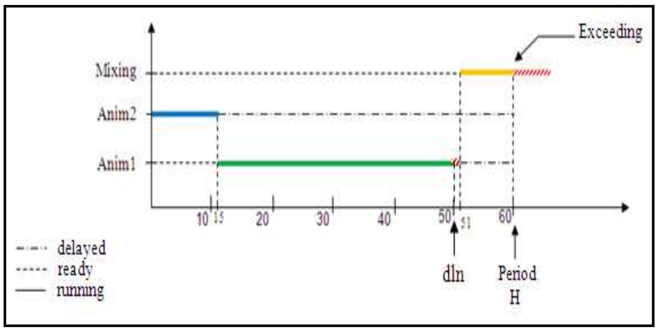
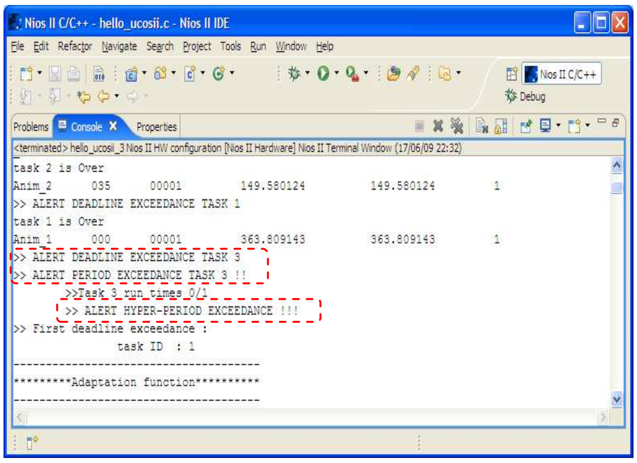


Figure 5. Execution graph of the second scenario

Task Anim1 begins to exceed its deadline by a tick (until 51). Then, the task "Mixing" exceeds its deadline also and continues to run up more than its period. Since it runs once a "H", the task "Mixing" also exceeds the "H".

We illustrate in the fig. 6 that the execution on platform is in conformity with the theoretical result.


Nios II C/C++ - hello_ucosii.c - Nios II IDE
File Edit Refactor Navigate Search Project Tools Run Window Help
Nios II C/C++
Debug
Problems
Console
Properties

```
<terminated> hello_ucosii_3 Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (17/06/09 22:32)
task 2 is Over
Anim_2      035      00001         149.580124          149.580124        1
>> ALERT DEADLINE EXCEEDANCE TASK 1
task 1 is Over
Anim_1      000      00001         363.809143          363.809143        1
>> ALERT DEADLINE EXCEEDANCE TASK 3
>> ALERT PERIOD EXCEEDANCE TASK 3 !!
        >>Task 3 run times 0/1
        >> ALERT HYPER-PERIOD EXCEEDANCE !!!
>> First deadline exceedance :
               task ID  : 1
--------------------------------------
*********Adaptation function**********
--------------------------------------
```

Figure 6. Illustration of "H" exceeding detection

Fig. 6 illustrates the deadline and period violation alerts triggered by the local watchdogs of both tasks Anim1 and Mixing. Then, the global 'w' triggers a "H" exceeding alert. It determines from the exceeding event list already saved the first overtaking that caused the disruption of the tasks execution. Finally, it calls the adaptation function to solve the problem.

V. CONCLUSION

The present paper dresses the issue of timing constraints violation in soft real time applications because of the variability of task's execution time. It proposes a control technique implemented in a real time kernel in order to enable the monitoring of tasks execution. This technique handles the detection of any task deadline or period exceeding using a local 'w' for each task and one global 'w' for the whole application. It identifies the possible influences of these violations on the overall functioning of the system. The resulting work may be seen as a supervising block having the OS parameters (execution time, WCET, deadline, period, and "H") as input and the timing constraints violation alerts as output.

The establishment of the proposed technique has gone through several stages. First, we needed to use a real-time scheduling algorithm. We chose the EDF algorithm and were brought to implement it in the RTOS MicroC/OS-II. We exploited time management services and the system clock tick interrupt provided by this kernel for successful integration of the scheduling policy. The next stage was the implementation of the control mechanism. We defined five new system calls that have to be used by a MicroC/OS-II application developer to benefit from these new services. The work has been tested and validated through the 3D rendering application running on an Altera FPGA prototyping platform. Our technique is used to set up a more general adaptation technique that manages two other constraints: lifetime, battery and output quality [12].

REFERENCES

[1] Audrey M. and Silly-Chetto M., "Simulation and assessment of real time scheduling algorithms under constraints of QoS," Research report, September 2004

[2] Decotigny D., "Bibliographie d'introduction à l'ordonnancement dans les systèmes informatiques temps réel," http://david.decotigny.free.fr/rt/intro-ordo, 1999-2002

[3] A. Tanenbaum, "Modern Operating Systems," Prentice Hall, 1994, ISBN 0130313580

[4] N. Audsley and A. Burns, "Real-time systems scheduling," Department of Computer Science, University of York, UK

[5] J. Delacroix, "Linux Programmation système et réseau," Dunod, 2009, ISBN 13 : 978-2-10-052539-3

[6] N. Navet and J.P. Thomesse, "L'ordonnancement, la clé d'une gestion efficace des ressources," in: J'automatise, september 2002, no 24

[7] N. Navet, "Introduction to the schedulability analysis," academic course available at www.loria.fr/~nnavet/cours/Mines2003-2004/Mines-Ordo-03_04.pdf, November 2003

[8] Labrosse J.J., "MicroC/OS-II, The real time kernel," R&d books editions, 2002, ISBN 1-57820-103-9

[9] Altera corporation, "NIOS II software developer's Handbook," March 2009

[10] C.Lu, J.Stankovic, G.Tao, and S.Son. Feedback control real-time scheduling: Framework, modeling and algorithm. special issue of RT Systems Journal on Control-Theoretic Approaches to Real-Time Computing, 23(1/2):85–126, july/september 2002.

[11] B.Li and K.Nahrstedt. A control-based middleware framework for quality of service adaptation. IEEE Journal on Selected Areas in Communication, September 1999

[12] K. Loukil, N. Ben Amor, M. Abid, "Self adaptive reconfigurable system based on middleware cross layer adaptation model," SSD'09, Djerba, Tunisia, March 2009.