IECON'2006 - The 32nd Annual Conference of the IEEE Industrial Electronics Society
Paris, France, 7-10 November 2006

Timing Analysis of Automatically Generated Code by MATLAB/Simulink

Rômulo Silva de Oliveira, Marcos Vinicius Linhares, Ricardo Bacha Borges

Systems and Automation Department - DAS
Federal University of Santa Catarina - UFSC
Campus Universitario - Trindade, Florianopolis, SC, 88040-900, Brazil
{romulo,marcos.rbb}@das.ufsc.br
http://www.das.ufsc.br

Abstract— **In the development of a real-time embedded control system it is necessary several types of professionals. The control engineer's tool is the functional block diagram, since it describes the mathematical models of the controller being designed. Many development tools that use block diagrams also automatically generate code. The objective of this work is to show how it is possible to evaluate, through an algebraic model, the timing behavior of the multitask system generated automatically by Simulink/MatLab, together with its real-time microkernel. This modeling allows the control engineer a better understanding of the timing behavior of the tasks, including the identification of delay and jitter problems.**

I. Introduction

The inclusion of electronic systems in a great variety of products, such as cars, domestic equipments and other, has been causing a huge growth of the electronic industry. Traditional products are now more efficient, with improved quality and lower cost.

Nowadays, equipments from simple domestic machines to complete industrial facilities can be controlled through the use of computer systems. When those systems are firmly fixed into the equipments to the point that, if they are removed, the equipment stops working, then they are called embedded systems [1][2]. Many times these systems have timing requirements. Real-time embedded systems are systems whose correct operation depends on the produced results and on the instant in which those results are produced [3][4][5].

Many real-time embedded systems are designed through empiric techniques and approaches. Several control applications with timing constraints are implemented by manual coding of huge programs in assembly language, programming timers and manipulating low level I/O devices, tasks and interrupt priorities. In spite of the code produced by these techniques to be optimized for an efficient execution, this approach has some disadvantages [6]:

- the assembly programming language is more difficult to understand than others high level programming languages;
- few people get to understand the complete operation of the software produced;
- partial modifications of the program becomes very difficult.

If the timing constraints are not verified previously and the operating system does not include characteristics to manipulate real-time tasks, the system may work correctly for some time but it may reach an error state in specific situations. Without the specific support of tools and methodologies, the analysis for verification of timing constraints become practically impossible. The consequences of the system failure depend on the criticality of the application.

For the design and implementation of a real-time embedded control system it is necessary several types of professionals, each one is a specialist in his/her area that requires its own tools. A simple system would involve a hardware engineer to design the hardware to be used, a control engineer to design the controller and to define the timing constraints and a software engineer to implement the controller.

The interaction between the control engineer and the software engineer should be the most effective possible since one is designing what the other will implement. It is of extreme importance that both exchange information by using the same language. The language of the control engineer is the functional block diagram with the controller's mathematical models. In order to facilitate the communication among them we can map functional blocks to software components.

With the objective of integrating the software engineer with the control engineer, several development tools that use functional block diagrams also include resources for automatic code generation.

MathWorks [7], maker of MATLAB/Simulink, has a tool that generates code in C language corresponding to the system modeled by block diagrams on Simulink. It is called RTW (Real Time Workshop). The MathWorks introduced the Embedded Target as a way to tailor this code to the main makers of embedded control platforms, inserting specific characteristics for each one, like Motorola and Texas Instruments. Simulink allows the design of a multitask system by generating a small real-time microkernel.

The objective of this work is to show how to evaluate, through an algebraic model, the timing behavior of the multitask system automatically generated by MATLAB/Simulink, together with its real-time microkernel. We present the case study of an application for controlling an electric motor implemented on a DSP.

This modeling allows the control engineer to have a bet-

ter understanding of the timing behavior of the tasks, identifying delay problems that are caused by the execution of a multitask system.

Section II of this article describes the application and the hardware platform used in the case study. Section III describes the modeling of timing behavior. Section IV compares the theoretical results with mensurations of the real system. Section V has the final comments.

II. Application

The application was executed on the eZdsp LF2407 from Spectrum Digital, a development board that includes the DSP from Texas Instruments TMS320C2407A. This DSP targets the industrial control market, mainly electric motor control and frequency inverters. Among its peripherals there is an analog-to-digital conversor (ADC) with a sample rate of 2MS/s, a PWM module with vetorial control, digital inputs and outputs, serial and CAN communication interfaces. Each application task is associated with a block diagram. Code was automatically generated by the Real-Time Workshop Embedded Coder.

In order to analyse the timing behavior of the code generated by Simulink an application was implemented by using available blocks. The application consists of six tasks:

- Two feedback control loops for the two CC motor speeds with a PI controller using a PWM output and anti-aliasing digital filter of twentieth order in the feedback (two tasks);
- Receiving of the speed set-point for the two motors through a CAN network;
- Sending data (two motor speeds, PI controller output and set-point) through a CAN network;
- Human-machine interface including a small keypad and a display (those two tasks were simulated by s-functions).

Figure 1 illustrates the tasks specified in MATLAB/Simulink. Each one of the blocks contains the peripheral (CAN, ADC, digital inputs and outputs, PWM and other) necessary for the correct operation of each task, as well as the target platform. In Fig. 2 task "controller" can be seen in details. It can be noticed the presence of hardware elements (PWM, ADC, memory) as well as software elements (buffer, data conversor) and the controller algorithm (PI). Those elements were used from the tools palette of MATLAB/Simulink.

In order to configure Simulink for the application it was first chosen the target platform to be the LF2407 eZdsp, of the Embedded Target of MATLAB/Simulink. TimerClockPrescaler was set to 128, that is the value for which the timer module divides the clock source, that is 40MHz in this case. Therefore, the frequency of the timer that generates the interruption is 312.5KHz; and the FundamentalStepSize (fundamental time) was set to 0.001 seconds, that is the period of the timer interrupt (1ms).

The periods of the tasks were configured (sample time of the blocks) with the appropriate values. These should be a multiple of the fundamental time. After that the code was automatically generated by the tool.

It was not a goal of this work to evaluate the code generated by the tool (its complexity or optimization). Our goal was to model the timing behavior of the application, that is, the operation of the tasks and microkernel generated by the tool, and their mutual timing interference.

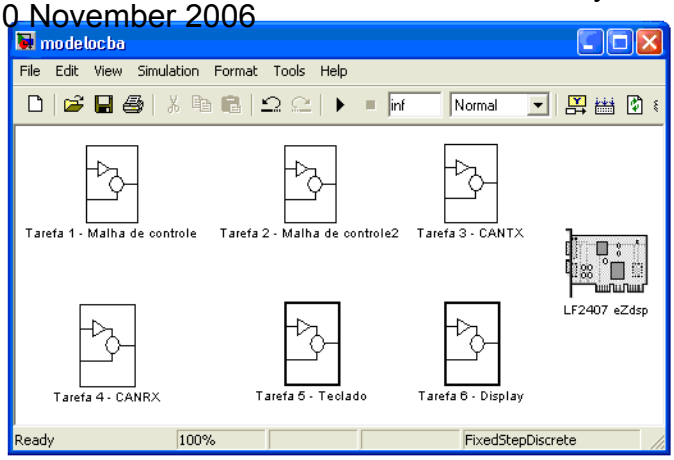

Fig. 1. Aplication

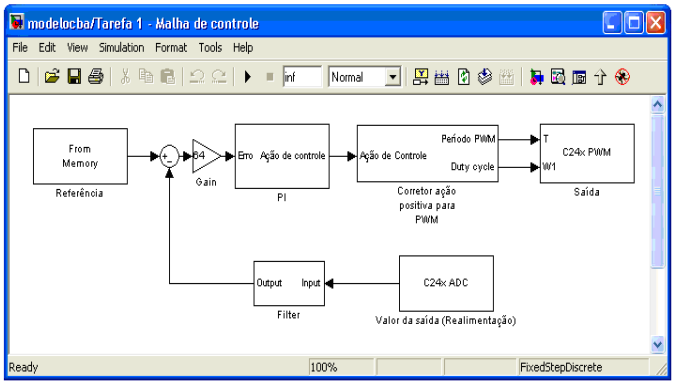

Fig. 2. Controller Task

III. Timing Behavior Analysis

MATLAB/Simulink generates not only the code of the application but also a very small real-time microkernel that is added to the system in order to manage several periodic tasks.

The microkernel has a scheduler based on rate-monotonic, where tasks with a smaller period have a higher priority.

The microkernel handles the timer interrupt which period is configured inside the tool (FundamentalStepSize). This time base is assumed for all tasks of the application (fundamental time), that is, all tasks must have periods that are multiple of that base. Tasks with equal periods (sample time) are merged in a single task when the code is generated.

The microkernel generated automatically by the tool is responsible for managing the several tasks and to guarantee that they execute in their configured periods. The activity diagram of the microkernel can be seen in Fig. 3.

The microkernel uses a scheduler algorithm based on the rate monotonic. It implements a simple rule that assigns priorities in agreement with the periods of the tasks.

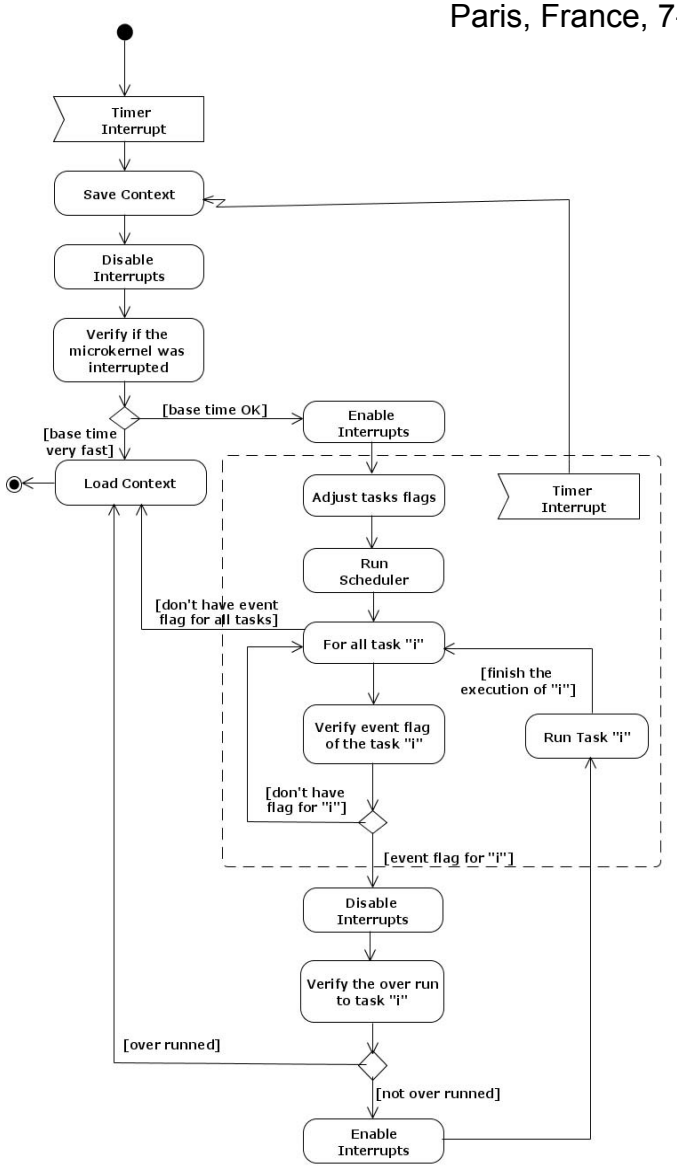

Fig. 3. Activity diagram of the MATLAB/Simulink microkernel.

Specifically, tasks with smaller periods have higher priorities [6].

By analyzing the activity diagram it can be noticed that the microkernel is quite simple and generated specifically for the task set specified by the block diagrams.

In order to prevent a task from begining its execution but never ending (starvation), the microkernel implements an over-run flag. If a task begins its execution but does not complete it until the end of its period then it starts to execute with disabled interrupts. That guarantees the execution of all tasks but, from the point of view of real-time scheduling, it may cause the missing of deadlines of tasks with higher priority in order to provide for the execution of tasks with lower priorities (priority inversion).

Figure 4 shows how the microkernel manages the release of an application task.

It is noticed that the use of the "switch" to select which task will be executed causes interference to tasks of lower

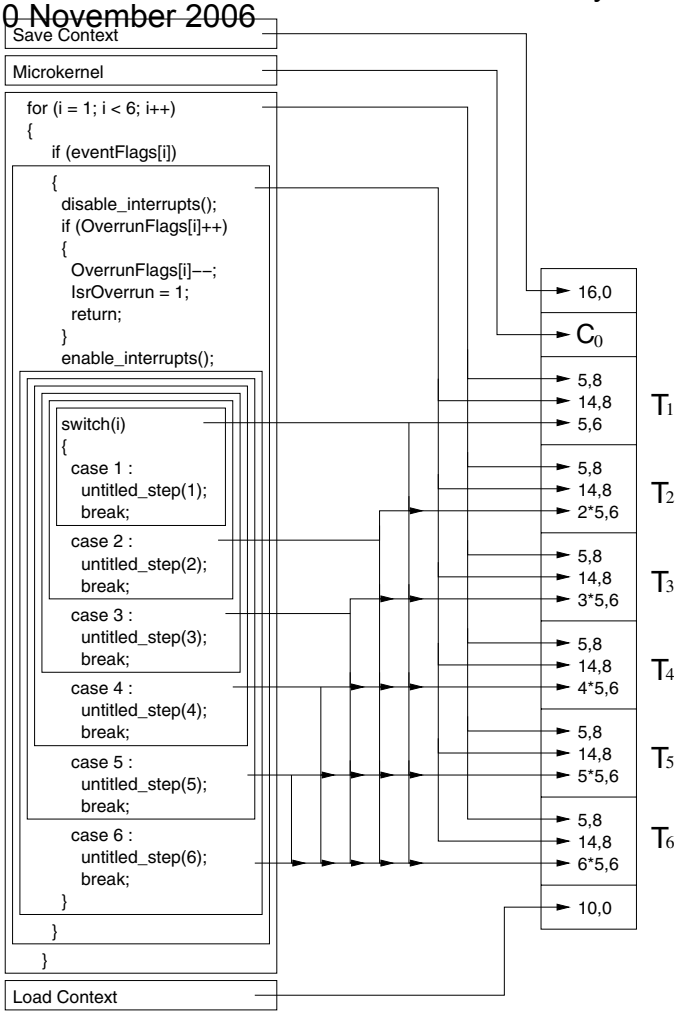

Fig. 4. The time involved in the release of a task.

priority.

A. Mathematical Model

The goal of this timing modeling is to explain how the response time of each task is formed by adding delays from different parts of the software. This model is based on the response time analysis proposed in [8].

We consider in the calculation of the application timing behavior the following parameters:

- the task execution time (C_x) - the time spent by the task, exclusively, in its execution;
- the interference time form the microkernel (I_k) - the time spent with microkernel execution, that is the price to be paid for the real-time multitask facilities.
- the intereference time from higher priority tasks (I_x) - the sum of the execution time of the tasks that have higher priority than the task x during its execution (R_x);
- the task response time (R_x) - represented by the sum of the task execution time, the microkernel interference time and the interference time by the tasks that have higher priority until task x finishes.

Some overheads (see Fig. 4) affect C_x (task execution time):

1. To discover which task needs to execute, that takes 14.8 μs for each task;
2. To select the task that needs to execute, that takes 5.6 μs and it depends directly on the priority of the task.

We can define a C'_x (corrected execution time) composed of some elements (considering that x reflects the priority of the task). Equation (1) defines the corrected execution time.

$$C'_x = C_x + 14.8 + (5.6 \cdot x) \qquad where, \quad x \geq 1 \qquad (1)$$

The interference caused by the microkernel ($I_k(x)$) is represented by the time spent with the timer interrupt handling and its associated overheads. It includes:

- routines for the timer interrupt handling and execution of the scheduler algorithm, represented by the microkernel execution time (C_0). The microkernel has an execution time that varies in agreement with the number of tasks and with optimizations of the compiler used;
- a loop for choosing the next task to be executed, it spends $[n{\cdot}5.8]$ μs (where n is the number of application tasks) when no task executes; or $[5.8 \cdot x]$ μs for each task x, where x reflects the priority of the task;
- routines for context saving and reloading that spend, respectively, 16.0 μs and 10.0 μs.

The microkernel generated by the tool does not execute in an atomic and homogeneous way for all the tasks. Its execution time can be divided in two different parts: C_k, when the timer interrupt happens and no one task is released, and $C'_k(x)$, when the timer interrupt releases task x. So, $I_k(x)$ is defined in (2).

$$I_k(x) = C'_k(x) + \left(\left\lceil \frac{R_x}{P_0} \right\rceil - 1 \right) \cdot C_k \qquad (2)$$

Where C_k and $C'_k(x)$ are shown in (3) and (4).

$$C_k = 16.0 + C_0 + (n \cdot 5.8) + 10.0 \qquad (3)$$

$$C'_k(x) = 16.0 + C_0 + (5.8 \cdot x) \qquad (4)$$

Equation (5) is used to compute the task response time. It is important to know how many time the task x spent since its execution started until its execution ended, considering the intereferences (preemptions) during the task execution.

$$R_x = C'_x + I_k(x) + I_x \qquad (5)$$

Where I_x (interference caused by higher priority tasks) is given by (6).

$$I_x = \sum_{i=1}^{x-1} \left\lceil \frac{R_x}{P_i} \right\rceil \cdot C'_i \qquad (6)$$

In the MathWorks implementation the microkernel has the maximum priority in the system. Even so, if any task is interrupted and does not get to return until the beginning of its next period (over-run) it receives the processor, disabling all interrupts, including timer interrupts. That may cause at some moment the deadline missing of another higher priority task.

It is also important to verify if there will be or not the occurrence of over-run, since it would destabilize the whole system. Therefore, considering the specified periods and priorities, assuming that the deadline of the task (D_x) is equal to its period, and calculating the response times, in (7) is showed an schedulability test that is necessary and sufficent to prevent overruns.

$$\forall x, R_x \leq D_x \qquad where, \quad D_x = P_x \qquad (7)$$

The knowledge of the timing behavior during system design can prevent an undesirable behavior in run-time.

IV. Experiences

Based on the application modeled in MATLAB/Simulink it was put to test the model of the timing behavior. This section describes how some parameters were acquired for the solution of the equations.

Table I describes the period of all tasks. The period of the timer interrupt was defined as 1 ms. The period of each task was attributed to satisfy the needs and requirements of the application.

x	**Task (T_x)**	**Period (P_x)**
0	**Timer interrupt**	1 ms
1	**Feedback control loop 1**	2 ms
2	**Feedback control loop 2**	3 ms
3	**CAN data send**	10 ms
4	**CAN data receive**	15 ms
5	**Keyboard (HMI)**	100 ms
6	**Display (HMI)**	150 ms

TABLE I
Tasks Periods.

In order to accomplish the measurings of the execution time (C_x) some available digital outputs of the DSP were physically connected to an osciloscope. The code was generated automatically by RTW of Simulink and the support to peripherals is given by the Embedded Target for TI C2000 DSP. Each task sets its corresponding output at high level at the beginning of its execution and low level at the end. Initially, the periods of the tasks were configurated so as it was guaranteed that no task would be interrupted by another task during its execution, since that would change the resulting response time. Table II shows the measured execution time (C_x). The table also presents the corrected execution time (C'_x) for the group of tasks, and the execution time of the microkernel when it starts task $x(C'_k(x))$, calculated according to the equations presented before.

The execution time of the microkernel when it does not start any task (C_k) was calculated, for this task set, to be 135.4 μs.

The remaining times were calculated according to the equations presented before. The response time is obtained

Task (T_x)	$C_x(\mu s)$	$C'_x(\mu s)$	$C'_k(x)(\mu s)$
0	74.6	–	–
1	541.2	561.6	96.4
2	540.8	566.8	102.2
3	80.7	112.3	108.0
4	24.4	61.6	113.8
5	1616.0	1658.8	119.4
6	10400.0	10448.4	125.4

TABLE II
TASKS EXECUTION TIME.

by an iteration that diverges ($R_x > P_x$) or converges in a finite number of steps ($R_x(m+1) = R_x(m) = R_x$).

Since it is an iterative process, the initial value of R_x defines the number of iterations necessary for the value to converge. A greater initial value will reduce the number of necessary iterations. The initial value ($R_x(0)$) is given by (8), since the response time of a task of lower priority will suffer interference no less than the response time of the previous task and so on.

$$R_x(0) = C'_x + R_{(x-1)} \tag{8}$$

Equation (9) shows the response time solution for the task $x = 1$. It can be noticed that it is not interfered by C_k only for C'_x, that is because its response time is less than the period of the timer interrupt ($R_x < P_0$). It receives interference only from C'_k before starting its execution. The other tasks ($x = 2; 6$) have a response time that is greater than the period of the timer interrupt ($R_x > P_k$) where C_k increases the response time.

$$R_1(0) = \overbrace{C'_1}^{C_1+14.8+(5.6\cdot 1)} + \overbrace{C'_k(1)}^{16+74.0+(5.8\cdot 1)} = 658.0\ \mu s \tag{9}$$

Equations (10), (11), (12), (13) and (14) show the response time solution of the tasks 2 to 6, respectively.

$$\begin{aligned} R_2(0) &= C'_2 + R_1 = 1224.8\mu s \\ R_2(1) &= \overbrace{566.8}^{C'_2=C_2+14.8+(5.6\cdot 2)} + \overbrace{102.2}^{C'_k(2)=16+74.0+(5.8\cdot 2)} + \\ &\quad + \left(\left\lceil \frac{1224.8}{1000} \right\rceil - 1\right) \cdot 135.4 + \\ &\quad + \left\lceil \frac{1224.8}{2000} \right\rceil \cdot 561.6 = 1366.0\ \mu s \end{aligned} \tag{10}$$

$$\begin{aligned} R_3(0) &= C'_3 + R_2 = 1478.3\mu s \\ R_3(1) &= \overbrace{112.3}^{C'_3=C_3+14.8+(5.6\cdot 3)} + \overbrace{108.0}^{C'_k(3)=16+74.0+(5.8\cdot 3)} + \\ &\quad + \left(\left\lceil \frac{1478.3}{1000} \right\rceil - 1\right) \cdot 135.4 + \\ &\quad + \left\lceil \frac{1478.3}{2000} \right\rceil \cdot 561.6 + \\ &\quad + \left\lceil \frac{1478.3}{3000} \right\rceil \cdot 566.8 = 1484.1\ \mu s \end{aligned} \tag{11}$$

$$\begin{aligned} R_4(0) &= C'_4 + R_3 = 1545.7\mu s \\ R_4(1) &= \overbrace{61.6}^{C'_4=C_4+14.8+(5.6\cdot 4)} + \overbrace{113.8}^{C'_k(4)=16+74.0+(5.8\cdot 4)} + \\ &\quad + \left(\left\lceil \frac{1478.3}{1000} \right\rceil - 1\right) \cdot 135.4 + \\ &\quad + \left\lceil \frac{1545.7}{2000} \right\rceil \cdot 561.6 + \\ &\quad + \left\lceil \frac{1545.7}{3000} \right\rceil \cdot 566.8 + \\ &\quad + \left\lceil \frac{1545.7}{10000} \right\rceil \cdot 112.3 = 1551.5\ \mu s \end{aligned} \tag{12}$$

$$\begin{aligned} R_5(0) &= C'_5 + R_4 = 3210.3\mu s \\ R_5(1) &= \overbrace{1658.8}^{C'_5=C_5+14.8+(5.6\cdot 5)} + \overbrace{119.4}^{C'_k(5)=16+74.0+(5.8\cdot 5)} + \\ &\quad + \left(\left\lceil \frac{3210.3}{1000} \right\rceil - 1\right) \cdot 135.4 + \\ &\quad + \left\lceil \frac{3210.3}{2000} \right\rceil \cdot 561.6 + \\ &\quad + \left\lceil \frac{3210.3}{3000} \right\rceil \cdot 566.8 + \\ &\quad + \left\lceil \frac{3210.3}{10000} \right\rceil \cdot 112.3 + \\ &\quad + \left\lceil \frac{3210.3}{15000} \right\rceil \cdot 61.6 = 4615.3\ \mu s \\ R_5(2) &= \ldots = 5312.3\ \mu s \\ R_5(3) &= \ldots = 5447.7\ \mu s \end{aligned} \tag{13}$$

$$
\begin{aligned}
R_6(0) &= C'_6 + R_5 = 15896.1\mu s \\
R_6(1) &= \overbrace{10448.4}^{C'_6=C_6+14.8+(5.6\cdot 6)} + \overbrace{125.4}^{C'_k(6)=16+74.0+(5.8\cdot 6)} + \\
&+ \left(\left\lceil \frac{15895.9}{1000} \right\rceil - 1 \right) \cdot 135.4 + \\
&+ \left\lceil \frac{15896.1}{2000} \right\rceil \cdot 561.6 + \\
&+ \left\lceil \frac{15896.1}{3000} \right\rceil \cdot 566.8 + \\
&+ \left\lceil \frac{15896.1}{10000} \right\rceil \cdot 112.3 + \\
&+ \left\lceil \frac{15896.1}{15000} \right\rceil \cdot 61.6 + \\
&+ \left\lceil \frac{15896.1}{100000} \right\rceil \cdot 1658.8 = 22505.0\ \mu s \\
R_6(2) &= \ldots = 26945.1\ \mu s \\
R_6(3) &= \ldots = 29176.7\ \mu s \\
R_6(4) &= \ldots = 30711.3\ \mu s \\
R_6(5) &= \ldots = 32149.0\ \mu s \\
R_6(6) &= \ldots = 32981.4\ \mu s
\end{aligned} \tag{14}
$$

Table III shows the timing behavior of the application developed and implemented on the selected platform. The response time (R_x) and the interference time of higher priority tasks (I_x) and the interference of the microkernel ($I_k(x)$) were calculated according to the equations presented before.

Task (T_x)	$R_x(\mu s)$	$C'_x(\mu s)$	$I_x(\mu s)$	$I_k(x)(\mu s)$
1	658.0	561.6	–	96.4
2	1366.0	566.8	561.6	237.6
3	1484.1	112.3	1128.4	243.4
4	1551.5	61.6	1240.7	249.2
5	5447.7	1658.8	2992.3	796.6
6	32981.4	10448.4	18074.8	4458.2

TABLE III
RESPONSE TIME.

By observing table IV it is possible to notice that the proposed model correctly represents the worst-case timing behavior. By directly measuring the response time of all tasks and calculating it from the model there is a maximum error (calculated/measured) of at most 2.00% for the application studied.

The error occurs mainly in the tasks which response time is smaller than the period of the timer. In the slower tasks this error is relatively smaller when compared to the task period.

Task (T_x)	$R_x(\mu s)$ $[Calculated]$	$R_x(\mu s)$ $[Measured]$	$Erro(\%)$
1	658.0	652.6	0.83
2	1366.0	1357.0	0.66
3	1484.1	1467.0	1.16
4	1551.5	1521.0	2.00
5	5447.7	5400.0	0.88
6	32981.4	32866.0	0.35

TABLE IV
RESPONSE TIME ERROR.

V. FINAL CONSIDERATIONS

In this paper it was shown how it is possible to evaluate, through an algebraic model, the timing behavior of the code automatically generated from functional block diagrams. It was used as case study a multitask system generated automatically by MATLAB/Simulink, together with its real-time microkernel. The application was the control of electric motors, implemented on a small DSP. Implementation details of the microkernel needed to be modeled.

The worst-case response time calculated was compared with measurings of the actual system. The calculated times were slightly above the measured ones, as it was expected. The most difficult aspect continues to be how to obtain the worst-case execution time of the individual functions of the application. In the specific case of the studied scenario this aspect was facilitated by the simplicity of the hardware and by the determinism of the execution flow of the application tasks.

Because of the modeling, the designer of the real-time embedded system has a better understanding of the timing behavior of the final system. That improves the development process of systems with embedded controllers. For example, the sources of delays or undesirable variations of the response time can be more easily identified.

This work contributes to the improvement of the software development process of embedded systems in the context of applications which main component is a control strategy.

REFERENCES

[1] A. S. Berger, *Embedded Systems Design: An Introduction to Process, Tools and Techniques*. CMP Books, 2002.
[2] D. E. Simon, *An Embedded Software Primer*. Addison-Wesley, 1999.
[3] H. Kopetz, *Real-time systems: design principles for distributed embedded applications*. Kluwer Academic Publishers, 1997.
[4] Q. Li and C. Yao, *Real-time concepts for embedded systems*. CMP Books, 2003.
[5] J. Stankovic and K. Ramamrithan, Eds., *Tutorial on Hard Real-Time Systems*. IEEE Computer Society Press, 1988.
[6] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*, 4th ed., ser. The Kluwer international series in engineering and computer science. Real-time systems. Kluwer Academic Publishers, 2002.
[7] MathWorks, 2006. [Online]. Available: http://www.mathworks.com
[8] A. N. Audsley, A. Burns, M. Richardson, and K. Tindell, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, pp. 284–292, 1993.