

Evaluating the real-time performance of the RLC1 processor in control applications: a case study using a DC motor speed control

R. Cayssials, O. Alimenti, E. Ferro and J. Orozco
Department of Electrical Engineering - Universidad Nacional del Sur
Bahía Blanca - Argentina
iecayss@criba.edu.ar

Abstract

In this paper we evaluate the control performance of the RLC1¹ processor. We compare the results with the control performance of a real-time system based on a real-time operating system. The speed control of a DC motor is proposed as a case study. RLC1 shows a control performance suitable for low-cost, low-power embedded real-time control applications.

Keywords real-time processor, control application, VHDL model.

1 Introduction

RLC1 is a processor designed to be applied in embedded real-time applications. Previous real-time processors implement either real-time operating system (RTOS) functions in hardware ([1, 3]) or predictable runtime behaviour ([4]) instead of design a real-time architecture to implement real-time systems. Hence, those processors are very restrictive on either the priority discipline that they can implement or the number of tasks that the system can support. RLC1 architecture was designed from a real-time point of view and consequently any arbitrary priority discipline can be implemented over a set of up to 65535 tasks.

Real-time systems are used in control applications. However, most of real-time performance evaluations consider scheduling parameters (e.g. tasks' response times, missed deadlines, idle cpu-time, etc.) instead of evaluate the effects on the controlled application.

In this paper we evaluate the performance of the RLC1 processor in a control application. We com-

pare the performance of the RLC1 processor with the one obtained using a RTOS.

Section 2 introduces the main concepts on digital control. Section 3 describes a typical implementation of a RTOS. The architecture of the RLC1 processor is shortly explained in section 4. In section 5 the speed control of a DC motor is proposed as a case study. In section 6 the performance of the RLC1 is compared with the one of a typical RTOS implementation. Results are analysed in section 7. Conclusion are drawn in section 8.

2 Digital Control

The basic operation of a digital control system (Fig.1) is to read information from multiple sensors, calculate the outputs and send the results to actuators.

The control application is modelled by the transfer function $G(s)$. The control design problem is to specify the transfer function $C(z)$ to give to the closed-loop the desired control characteristics (e.g. stability, step response, settling time, steady-state error).

Both A/D and D/A converters are sampled at regular intervals and transform the continuous-time transfer function $G(s)$ into a discrete-time model of the control application. The discrete-time model derived from this transformation is not an *approximation* but an *exact* description of the behaviour of the application at sampling intervals ([2]).

$C(z)$ is implemented as software which is executed by a processor and consequently takes some time to complete. If the computational delay is very small compared with the dynamic of the system, it can be neglected. However, if the delay is not negligible, then it has an impact on the system and should be modelled.

¹R. Cayssials. All rights reserved.

A variable sampling interval during runtime produces a *jitter* that is not modelled by the discrete-time model and consequently it could lead to an undesirable behaviour of the application. Real-time systems should be design in order to reduce/eliminate this jitter.

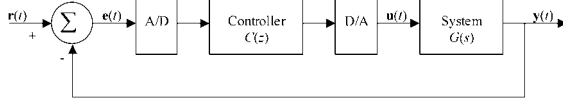


Figure 1: Classical Feedback Discrete-Time Control System. $C(z)$ is the transfer function of the controller and $G(s)$ is the transfer function that models the application.

3 RTOS Functions

A real-time system is modelled as a set Π of n periodic tasks to be executed on a processor. Each task i performs a certain function and it is characterised by its period, T_i , its deadline, D_i , and its execution time, C_i .

A *scheduler* shares the processor time among the real-time tasks of the system. The scheduler is a RTOS task that implements a priority discipline and it is invoked at discrete intervals, denoted T_{timer} . The time in a real-time system is considered slotted and either tasks' periods and tasks' deadlines should be expressed in slots. Therefore, task i will have to be invoked either every $\lfloor \frac{T_i}{T_{timer}} \rfloor$ or every $\lceil \frac{T_i}{T_{timer}} \rceil$ slots. T_{timer} has influence either on the system time resolution as well as the overhead that the RTOS produces.

A real-time task may be executed immediately after its invocation or may be completed just before its deadline. The maximum absolute jitter of task i is equal to D_i and consequently a real-time task i could produce a sampling interval, S_i such that $T_i - D_i \leq S_i \leq T_i + D_i$. This leads to a relative jitter of $\pm \frac{D_i}{T_i}$. Shorter deadlines improve the control performance of the system but require more powerful computational processors to make the system schedulable.

4 The RLC1 Architecture

RLC1 processor was developed under the Chris project. RLC1 architecture consists of two cooperative units (Fig.2): the User Program Processing

unit (UPPU) and the Real-Time unit (RTU).

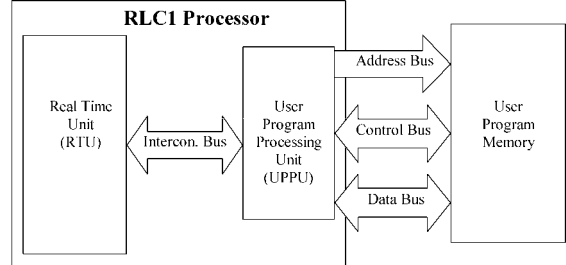


Figure 2: RLC1 processor architecture

On one hand, the UPPU is based on a 8051 processor core. It executes the task code stored in the User Program Memory. The set of instructions was expanded with some special real-time instructions in order to configure the real-time features of the RTU. Further architectures develop under the Chris project will consider different processors cores.

On the other hand, the RTU controls the whole real-time behaviour of the system. It may modify all the registers of the UPU through the interconnection bus. The RTU is based on an innovative design useful for real-time embedded applications. It handles the real-time tasks and priorities of the system.

The RTU assigns the UPU to the highest priority task that requires execution and preserves the state of the task that is preempted. The RTU halts the UPU when no task requires execution and consequently the power consumption of the system is reduced.

The architecture was designed to work with a 10MHz external clock. With this clock frequency, both tasks' period and tasks' deadline can be expressed with a time resolution of 100ns. No timer subroutine is executed and consequently both the overhead as well as the schedulability of the system are improved.

The whole architecture was described in VHDL and synthesised for either Altera, Xilinx and ASIC devices.

5 DC Motor Speed Control

5.1 Description of the application

The speed of a DC motor is controlled varying the energy transferred to the motor using a pulse width

modulation (PWM) technique. The period of the PWM wave, denoted T_{PWM} , depends on both the electrical and the mechanical characteristics of the motor and the load.

The speed is measured indirectly using an optical sensor and a slotted disc attached to the shaft of the motor (Fig.3). The sensor toggles its output every $\frac{2\cdot\pi}{72}rad$. The speed of the motor is then measured counting the number of toggles occurred in a certain interval I_s . The maximum speed of the motor is 2400rpm.

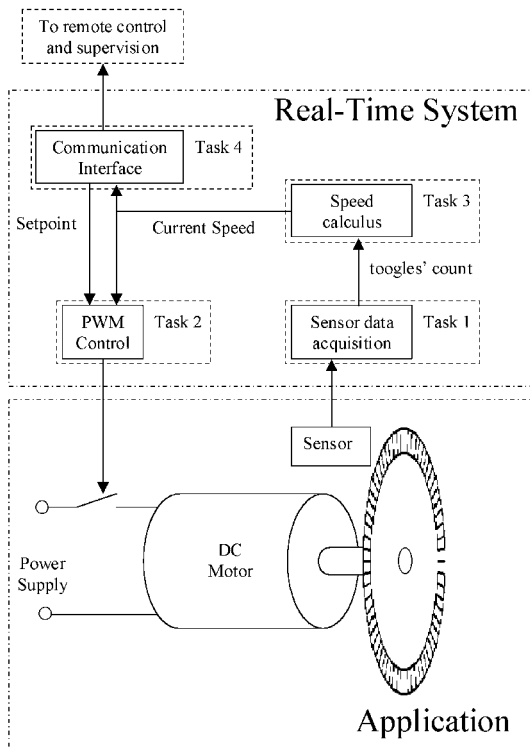


Figure 3: Scheme of the DC motor speed control. The PWM module controls the energy supplied to the motor and the sensor toggles its output every $\frac{2\cdot\pi}{72}rad$.

The speed setpoint is configured by a remote computer through an serial link. In this way, control and supervision of the motor can be done remotely using a standard communication link.

5.2 Real-time system specification

Four well define real-time tasks can be defined from the control functions that the system must perform.

- Task 1: Data Acquisition

This task is in charge of monitoring the sensor and counts the number of toggles that the sensor produces on its output. The task period should be less than $347.22 \mu s$ in order to avoid missing output toggles.

- Task 2: PWM control

Task 2 modifies the duty cycle of the PWM wave according to the difference between the setpoint and the speed of the motor. We set the period of the PWM wave in $T_{PWM} = 20ms$ and the period of task 2 equal to 1ms. So, the resolution on the duty cycle is equal to 5%.

- Task 3: Speed calculus

Task 3 calculates the speed of the motor through the angle elapsed in the interval I_s . The precision of the sensor produces an error on the speed calculus equal to $\frac{2\cdot\pi}{72\cdot I_s}$

If the current angular velocity is w , the error introduced when task 3 is executed every $I_s - D \leq T_3 \leq I_s + D$ instead of $T_3 = I_s$ is equal to $error_{speed} = \pm \frac{w\cdot D}{I_s}$.

A higher speed precision allows us to increase the gain of the PWM control in order to reacts faster to changes of the load conditions. However, errors in the current speed calculus will be amplified by a high gain and it may produce high frequency mechanical vibrations on the shaft of the motor. As a tradeoff between both considerations, we chose an interval equal to 23ms.

- Task 4: Communication interface

This task drives an asynchronous serial link working at 960 bytes/s. In order to avoid missing bytes received through the serial link, task 4 should be invoked at least every $\frac{1}{960}s \approx 1ms$.

6 Evaluation

The four real-time tasks were implemented to be executed on a 8051 microprocessor core equal to the one used in the RLC1 architecture in order to get a fair comparison. Table 1 shows the worst case execution time (in system clock period units), the period and the deadline of each one of the real-time tasks. Highest priorities are represented for lower task indexes.

Task	C [clock period]	T	D
1	401	347.22 μ s	347.22 μ s
2	436	1ms	1ms
3	506	23ms	23ms
4	366	1ms	1ms

Table 1: Worst case execution time, period and deadline of the real-time system tasks.

Timer intervals of the RTOS were selected in order to get tasks periods between 2 and 255 slots. In this way, the overhead of the real-time system is reduce when an 8-bit arithmetic is applied.

Simulations were performed for both systems considering clock frequencies equal to 10, 20, 40, 50, 80 and 100MHz. Figures 4 and 5 show the average jitter of task 1 and 3, respectively.

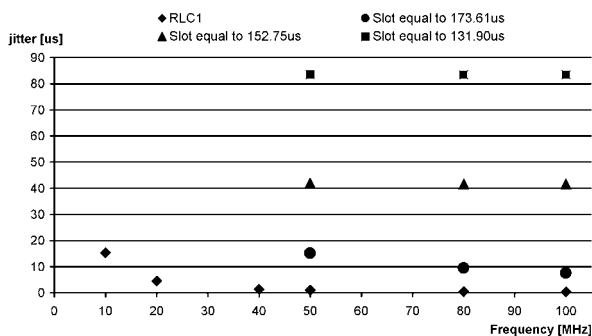


Figure 4: average jitter of task 1

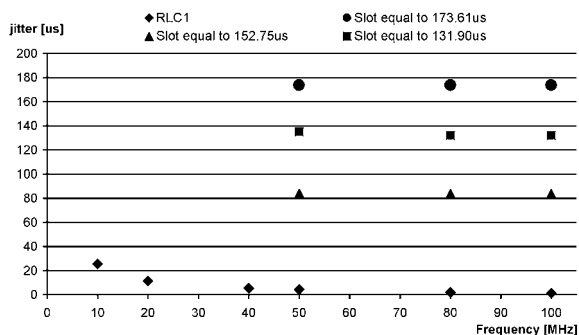


Figure 5: average jitter of task 3

RLC1 schedules the system with a clock of 10MHz whilst the RTOS based system needs at least a clock of 50MHz to schedule it.

7 Result Analysis

From a real-time point of view, the overhead of the system is reduced in a RLC1 based implementation and the system is schedulable at lower clock frequencies. So, the power consumption of the system is improved [5].

On the other hand, from a control theory point of view, the jitter is reduced when a RLC1 processor is utilised. The slot time in a RTOS has a greater influence on the control performance than the clock frequency. However, there may be not an optimal slot time for all tasks: whilst 173.61 μ s is optimal for task 1, a slot time of 152,75 μ s is optimal for task 3.

8 Conclusion

RLC1 processor requires a much less frequency to schedule the real-time system. The control properties of the real-time system are improved because the jitter of the control tasks is reduced. Consequently, the characteristics of RLC1 processor make it suitable to be applied to low-cost, low-power embedded real-time control applications.

References

- [1] Vlado Glavinić, Stjepan Gros, and Matjaz Colnarić. Vhdl-based modeling of a hard real-time task processor. In *IEEE ISIE'99*, pages 49–54, Bled, Slovenia, 1999.
- [2] Vaccaro Richard J. *Digital Control: A State-Space Approach*. McGraw-Hill Series in Electrical and Computer Engineering, 1995.
- [3] W.A. Halang M. Colnaric, D. Verber. Supporting high integrity and behavioural predictability of hard real-time systems. *Informatica, Special Issue on Parallel and Distributed Real-Time Systems*, 19(1):59–69, February 1995.
- [4] Matthew M. Wilding Steven P. Miller, David A. Greve and Mandayan Srivas. Formal verification of the aamp-fv microcode. Report NASA/CR-1999-208992, NASA, February 1999.
- [5] Clive Watts and Ravi Ambatipudi. Dynamic energy management in embedded systems. *IEE Computing and Control Engineering*, pages 36–40, Oct. 2003.