

AN EMBEDDED REAL-TIME CONTROL SYSTEMS LABORATORY ACTIVITY

Pau Martí¹, Manel Velasco¹ and Giorgio Buttazzo²

¹ Automatic Control Department, Technical University of Catalonia
Barcelona, Spain

²Retis Lab, Scuola Superiore Sant'Anna
Pisa, Italy

Research Report: ESAII-RR-08-03

August 2008

Abstract

This report presents a prototype laboratory activity to be integrated in the education of embedded control systems engineers. The laboratory experiment includes the control by means of hard real-time software task of an electronic circuit. The selection of the plant, as well as the selection of the real-time kernel and hardware platform where the control task executes is discussed. The report describes details of carrying out the experiment, and establishes a tentative work plan.

Keywords: Embedded systems, real-time systems, control systems, education.

Contents

1	INTRODUCTION	4
2	CONTROLLED PLANT AND PROCESSING PLATFORM	4
2.1	PLANT	4
2.2	PROCESSING PLATFORM	6
2.3	FLEX BOARD	6
2.4	ERIKA KERNEL	8
3	EXAMPLE OF THE LAB ACTIVITY	8
3.1	PLANT ELECTRONIC COMPONENTS	8
3.2	CONTROLLER DESIGN	9
3.3	IMPLEMENTATION 1	11
3.4	IMPLEMENTATION 2	13
4	TENTATIVE WORK PLAN	18
5	CONCLUSIONS	19

1 INTRODUCTION

Two of the challenges faced when teaching networked and embedded control systems at the graduate and under-graduate level can be addressed by appropriate laboratory activities. First, the background of the target audience can drastically vary within the same classroom. Hence, lab activities can be used to balance background differences between students. And second, the theoretical curriculum is often “too theoretical”. Therefore, lab activities must help putting the theory into the real world.

Taking into account such challenges, this report presents a prototype laboratory activity to be integrated in the education of embedded control systems engineers (for full curriculums, see e.g. [5], [6], [8] or [12]). The activity consists on implementing a real-time control application, that is, to control a physical plant by a controller implemented as a software task executing on top of a real-time operating system.

The report is organized as follows. Section 2 introduces the plant and the processing platform where the controller will be implemented. Section 3 discusses details of carrying out the activity. Section 4 presents an outline of a activity tentative work plan. Finally, Section 5 concludes the report.

2 CONTROLLED PLANT AND PROCESSING PLATFORM

2.1 PLANT

Control theory deals with shaping the behavior of physical plants. The chosen plant for the activity is a voltage stabilizer in the form of an electronic *RCRC* circuit, as illustrated in Figure 1.

The selection of this plant obeys several reasons. First, it is a linear time-invariant (LTI) plant, thus permitting to apply basic control techniques for LTI systems. Second, it can be directly plugged into a micro-controller,

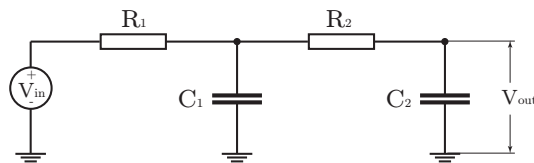


Figure 1: Electronic circuit scheme

meaning that not specific intermediate electronic circuits for signal conditioning (such as amplification, filtering, or converting) are required. That is, the TTL level signals provided by the micro-controller can be enough to carry out the control. Note that this is not the case, for example, of many mechanical systems.

This simplification in terms of the required hardware also helps in the plant modeling and control design stage. The modeling effort reduces to study the plant and no models for actuators or sensors are required. For example, using state-space formalism [2], the circuit that can be modeled in terms of the currents \dot{q}_i at each R_i by equations

$$\begin{aligned} \dot{q}_1 R_1 + (q_1 - q_2) \frac{1}{C_1} &= V_{in} \\ \dot{q}_2 R_2 + (q_2 - q_1) \frac{1}{C_1} + q_2 \frac{1}{C_2} &= 0 \\ q_2 \frac{1}{C_2} &= V_{out}, \end{aligned} \quad (1)$$

gives the state-space form

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{aligned} \quad (2)$$

with

$$\begin{aligned} A &= \begin{bmatrix} 0 & 1 \\ \frac{-1}{R_1 R_2 C_1 C_2} & -\frac{R_1 C_1 + R_2 C_2 + R_1 C_2}{R_1 R_2 C_1 C_2} \end{bmatrix} \\ B &= \begin{bmatrix} 0 \\ \frac{1}{R_1 R_2 C_1 C_2} \end{bmatrix} \\ C &= \begin{bmatrix} 1 & 0 \end{bmatrix} \end{aligned} \quad (3)$$

where $u(t)$ is the control signal, $y(t)$ is the plant output, and $x(t) = [x_1 \ x_2]$ is the state vector, where x_1 corresponds to the voltage in C_2 , and x_2 is \dot{q}_2/C_2 .

The modeling of the plant could have been also done in terms of a transfer function (see [11] for the analysis). Adopting the state space formalism adds another benefit to such control experiment. Since only x_1 , the output voltage, can be measured, the control algorithm demands the use of observers for predicting x_2 . This opens the door to experiment with several type of observers.

Depending on the electronic components and the designed controller, the circuit can be also made sensitive to jitters [1]. That is, interferences from other tasks on the control task deteriorate the control. This permits to study

advance results found in the literature that deal with this problem, such as [10]. Hence, the control set-up is also of interest for graduate students.

Finally, additional benefits of proposing this plant is that it can be easily build, it is cheap, and it can be easily transported.

Note that for this type of plant, the usual control objective would be to track a reference signal or to obtain a constant value at the output, which mandates to use tracking structures. However, the plant can also be used for regulation if the desired output voltage is zero.

2.2 PROCESSING PLATFORM

Networked and embedded control systems often require a processing platform enabled with real-time technology [3]. Such demand suggests that the processing platform should be a processor equipped with a real-time operating system (or real-time kernel) or a network infrastructure with real-time capabilities.

Since the first option seems to have a broader scope of application, the network is discarded. However, it has to be pointed out that there are available network platforms for implementing networked control systems such as the Controller Area Network kit by CCS Inc, [4], where a single board contains four CAN nodes.

Focusing on the processor and operating system, several options are available. First of all, many well-known real-time operating systems such as real-time linux [13] target processors that may be too powerful for embedded applications. But more important, they internal structure is often very complex for non computer scientists.

Taking into account the last observation, the chosen processing platform is the full FLEX board equipped with the ERIKA real-time kernel, both developed by Evidence srl [7] .

2.3 FLEX BOARD

The full FLEX board, illustrated in Figure 2, was born as a development board where to develop and test real-time applications. The main components of the board are:

- the Microchip dsPIC® DSC micro-controller dsPIC33FJ256MC710;
- the socket for the 100 pin Plug-In Module (PIM) available from Microchip;

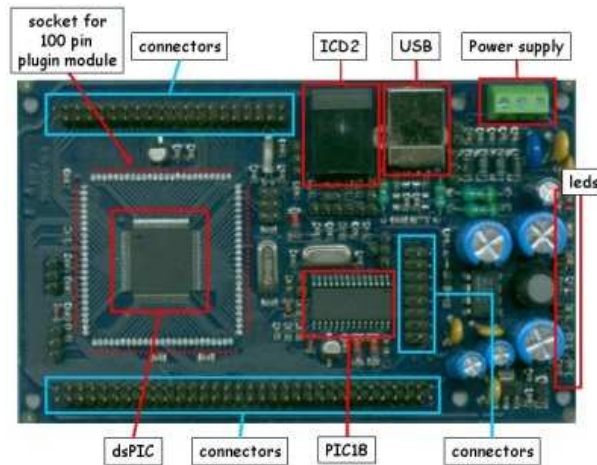


Figure 2: Full FLEX board

- the ICD2 programmer connector;
- the USB connector for direct programming;
- the power supply connectors;
- a set of leds for monitoring the board functioning status;
- the on-board Microchip PIC18® PIC18F2550 micro-controller for integrated programming;
- the set of connectors for daughter boards piggybacking.

The board design has key benefits. First, it is a robust electronic design, which is an important feature when employed by non-highly-skilled users. Second, it has a modular architecture. By exporting almost all the pins of the dsPIC micro-controller, the connection of components is very easy. The usage of custom, home-made daughter boards is also very easy. In particular, a set of general purpose daughter boards is also provided, such as a multi-bus board equipped with CAN, Ethernet, and other communication protocols.

The compact and bare design allows the use of the board not only for development purposes, but also makes the processing platform a suitable solution for direct deployment into the working environment.

Finally, it is important to stress that the board provides full support of the ERIKA Enterprise real-time kernel.

2.4 ERIKA KERNEL

ERIKA Enterprise, available under GPL license, is a real-time operating system for small micro-controllers based on an API similar to those proposed by the OSEK/VDX Consortium. Note the kernel has not been certified yet by the consortium.

The kernel gives support for preemptive and non-preemptive multitasking, and implements several scheduling algorithms for such as Fixed Priority with preemption thresholds, Stack Resource Policy (SRP), and Earliest Deadline First (EDF), which can be used to schedule tasks with real-time requirements. All of them use stack sharing for RAM optimization.

The API provides support for tasks, events, alarms, resources, application modes, semaphores, and error handling. It is important to note that its modular design ends producing a very small footprint. For example, a minimal installation of the kernel consumes from 800 to 2000 bytes of code, for the implementation of a fixed priority scheduling kernel with stack sharing and binary mutexes.

The development environment for ERIKA Enterprise is based in cross-compilation. Evidence srl provides RT-Druid (based on Eclipse) as default development platform. It implements an OIL language compiler, which is able from an OIL specification to generate the kernel configuration. Additional schedulability analysis tools are also available, to perform an estimation of the response time of the application tasks.

3 EXAMPLE OF THE LAB ACTIVITY

3.1 PLANT ELECTRONIC COMPONENTS

The design of the plant, and more specifically, the selection of the electronic components is very important for several reasons. The circuit impedance must be low enough to properly connect to the ADC converter or to external instrumentation such as an oscilloscope.

For example, given the initial components $R_1 = R_2 = 330 \text{ K}\Omega$ and $C_1 = C_2 = 100 \text{ nF}$, it is easy to see that the equivalent circuit impedance may be too high. By lowering R_i to $1 \text{ K}\Omega$ we obtain a manageable circuit impedance.

However, if we inject a square wave as input to the circuit with these components (open loop), we obtain as an output the same square wave. This indicates that the circuit dynamics are very fast, which may be difficult to appreciate the effectiveness of the control. If we increase C_i to $33 \text{ }\mu\text{F}$, the

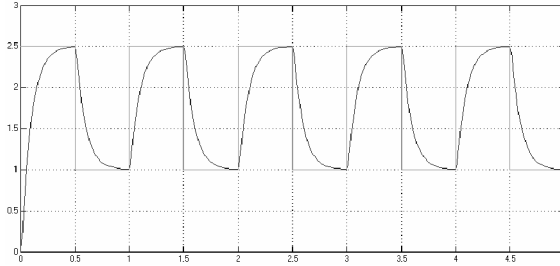


Figure 3: Matlab open loop *RCRC* response

dynamics become slower, as illustrated in Figure 3.

With this components, the state space model is

$$\begin{aligned} \dot{x}(t) &= \begin{bmatrix} 0 & 1 \\ -976.56 & -93.75 \end{bmatrix} x(t) + \\ &\begin{bmatrix} 0 \\ 976.56 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 1 & 0 \end{bmatrix} x(t). \end{aligned} \quad (4)$$

3.2 CONTROLLER DESIGN

For the chosen plant, the goal of the control will be to accelerate the *RCRC* response.

We select as a control objective to track a square wave. In addition, the design of the controller implies selecting the sampling period and the controller itself. Both decisions have an impact on control performance, but also they have to be selected and designed taking into account again the processing platform.

If the assumption is to avoid intermediate electronics between the plant and the dsPIC, the control signal will be generated by the PWM (by adjusting the duty cycle) and the measured variable will be obtained through the ADC. Therefore, the reference together with the controller must be chosen and designed so that the control signal voltage levels lie within the voltage levels provided by the PWM. Also, care must be taken with peak current levels.

Taking this into account, through an iterative design stage, and according to standard rules-of-thumb, the period is set to $h = 0.01$ s, the reference signal is specified to oscillate between 1 V and 2.5 V, and the discrete state feedback controller is specified to set the continuous closed loop poles at

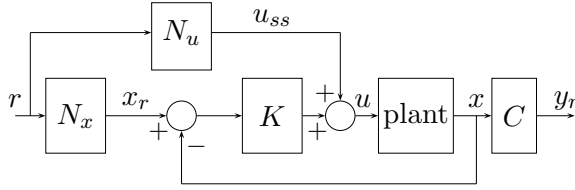


Figure 4: Tracking scheme

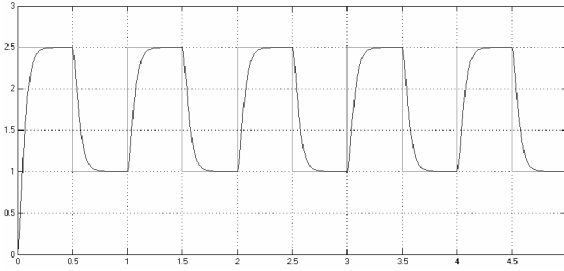


Figure 5: Matlab closed-loop *RCRC* response

$p_{1,2} = -30$. A shorter period, faster closed loop poles, or a higher amplitude for the reference signal may provoke values for the control signal out of the allowed physical ranges.

For the simulation, the standard tracking structure illustrated in Figure 4 was adopted, where N_u is the matrix for the feedforward signal to eliminate the steady-state error, and N_x is the matrix that transforms the reference r into a reference state, with $N_u = [1]$ and $N_x = [1 \ 0]$, and with controller gain $K = [0.0685 \ -0.0249]$. The simulated response in closed loop is shown in Figure 5.

For the simulation, the two state variables are available. However, in the real experiment, an observer must be included. Hence, complete and reduced observers can be analyzed and designed. For simplicity in the control task coding, a reduced observer has been chosen

$$\hat{x}_{k+1}^b = \Phi^{bb}\hat{x}_k^b + \Phi^{ba}x_k^a + \Gamma^b u_k + K_r \left[x_{k+1}^a - \Phi^{aa}x_k^a - \Gamma^a u_k - \Phi^{ab}\hat{x}_k^b \right]$$

where the sub-matrices are the partition of the discrete form of (4) with

```

CPU mySystem {
  OS myOs {
    EE_OPT = "DEBUG";
    CPU_DATA = PIC30 { APP_SRC = "code.c";
                      MULTI_STACK = FALSE;
                      ICD2 = TRUE;};
    MCU_DATA = PIC30 { MODEL = PIC33FJ256MC710;};
    BOARD_DATA = EE_FLEX { USELEDS = TRUE;};
    KERNEL_TYPE = EDF { NESTED_IRQ = TRUE;
                      TICK_TIME = "25ns";};
  };
  TASK myTask {
    REL_DEADLINE = "10ms";
    PRIORITY = 1;
    STACK = SHARED;
    SCHEDULE = FULL;
  };
  COUNTER myCounter;
  ALARM myAlarm {
    COUNTER = "myCounter";
    ACTION = ACTIVATETASK { TASK = "myTask"; };
  };
};

```

Figure 6: Kernel configuration file

$h = 0.01$ s as

$$\begin{bmatrix} x_{k+1}^a \\ x_{k+1}^b \end{bmatrix} = \begin{bmatrix} \Phi^{aa} & \Phi^{ab} \\ \Phi^{ba} & \Phi^{bb} \end{bmatrix} \begin{bmatrix} x_k^a \\ x_k^b \end{bmatrix} + \begin{bmatrix} \Gamma^a \\ \Gamma^b \end{bmatrix} u_k$$

$$y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_k^a \\ x_k^b \end{bmatrix}$$

The observer discrete gain is $K_r = -37.81$, which places the observer continuous closed loop pole at $p_{ob} = -40$.

3.3 IMPLEMENTATION 1

The first implementation involves coding the controller in a periodic task that will execute in isolation on top of ERIKA. The main pseudo-codes are illustrated in Figures 6, 7 and 8.

Figure 6 is the *conf.oil* file that specifies the kernel configuration with EDF and a periodic task that will be used to implement the controller. The basics of the main code are illustrated in Figure 7. First, the timer T1 is initialized, and together with *SetRelAlarm* will produce the periodic activation of the control task every 10 ms (the processor speed was configured at

```

int main(void)
{
    T1_program();
    EE_time_init();           // EDF initialization
    ADC_init();              // ADC1 configuration
    PWM_config();           // PWM1 configuration
    SetRelAlarm(myAlarm, 1, 10);
    for (;;)
    {
        Delay(100000);      // reference signal generation
        reference = 2.5;
        Delay(100000);
        reference = 1;
    }
    return 0;
}

```

Figure 7: Main code

```

TASK(myTask)
{
    x_1 = read_adc();
    r = referece;
    x_2 = Phi_bb*x_2old + Phi_ba*x_1old + Gama_b*u +
        Kr*(x_1 { Phi_aa*x_1old - Gama_a*u - Phi_ab*x_2old);
    u = r*NU + K_1*(r*NX_1 - x_1) + k_2*(r*NX_2 - x_2);
    write_pwm(u);
    x_1old = x_1;
    x_2old = x_2;
}

```

Figure 8: Control task code

40 MIPS). Figure 8 shows the control task code including the observer and the tracking structure.

Figures 9 and 10 show the experimental setup that includes an oscilloscope for displaying the circuit output voltage, in order to show the open loop response, and the closed loop response. The oscilloscope screen-shoots corroborate that the implementation achieves the control goal: the system output performs the desired fast tracking.

It is also interesting to monitor whether the control task executes when specified. A dsPIC pin is set to 0 and 1 when each controller job starts and finishes. By acquiring this information using also the oscilloscope, we can confirm, as illustrated in figure 11, that the activation is every 10 ms.

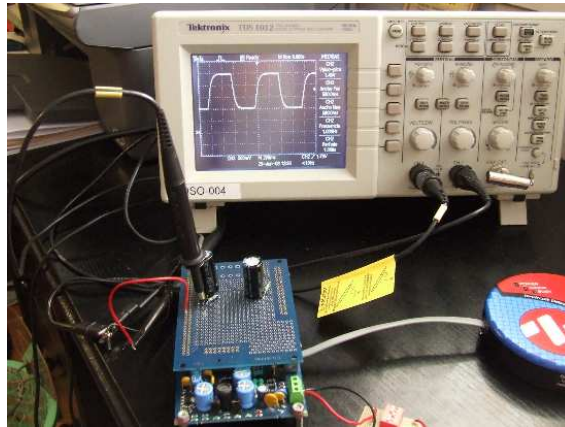


Figure 9: Experiment

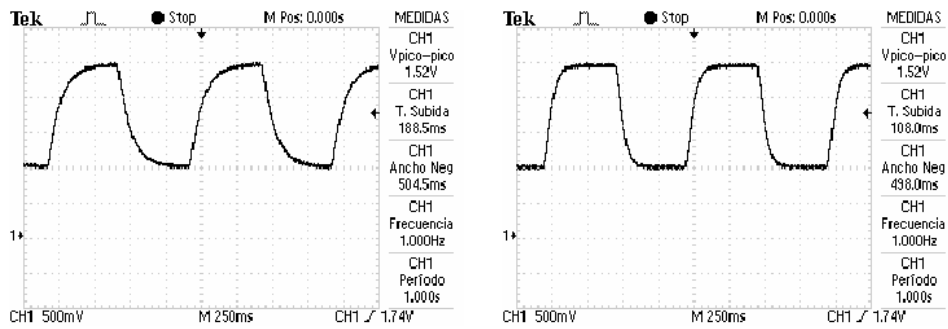


Figure 10: Open loop (left) and closed loop (right) *RCRC* response

3.4 IMPLEMENTATION 2

The second implementation is to illustrate more advanced concepts in the area of real-time and control systems co-design.

In the previous implementation we had a control task executing in isolation. However, in many high-tech systems, the processor is used both for the control computation and for many other software tasks such as interrupt and error handling, monitoring tasks, etc.

Hence, the second experiment involves a multitasking system and its first goal is to study what effects have the execution of other tasks in the control performance. Standard discrete time controllers have periodic timing demands. Sampling and actuation (S/A) have to be performed periodically,

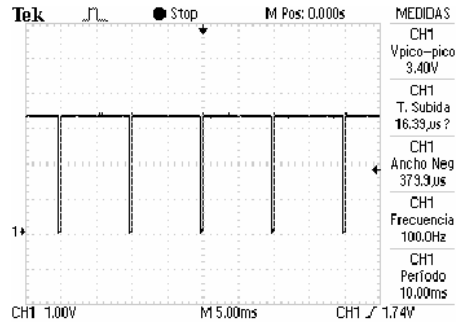


Figure 11: Control task activation

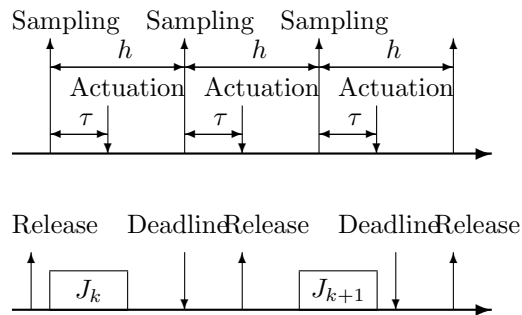


Figure 12: Control timing demands (top) and jobs of a hard real-time periodic task (bottom)

and may include a time delay τ within sampling and actuation, as illustrated in Figure 12 (top).

On the other hand, real-time theory mandates that control jobs J_k for a periodic task execute within each period and before the relative deadline, as illustrated in Figure 12 (bottom). Looking at control tasks, this restriction may not be enough to ensure perfect periodicity. If the control task deadline is equal to the period, and sampling and actuation operations are performed at the start and finishing time of each job (as coded for example in our control tasks, Figure 8), S/A will not be periodic. This phenomena is called *jitters*. And it is well known that jitters in controller jobs degrade control performance [1].

Hence, an starting experiment is to inject a new task in our dsPIC to see whether these degrading effects are visible. The new task has a period and relative deadline of 11 ms. The artificial execution time of the task, 9

```

CPU mySystem {
  ...
  TASK myTask {
    REL_DEADLINE = "10ms";
    PRIORITY = 1;
    STACK = SHARED;
    SCHEDULE = FULL;
  };
  TASK myTask2 {
    REL_DEADLINE = "11ms";
    PRIORITY = 1;
    STACK = SHARED;
    SCHEDULE = FULL;
  };
  COUNTER myCounter;
  ALARM myAlarm {
    COUNTER = "myCounter";
    ACTION = ACTIVATETASK { TASK = "myTask"; };
  };
  ALARM myAlarm2 {
    COUNTER = "myCounter";
    ACTION = ACTIVATETASK { TASK = "myTask2"; };
  };
};

```

Figure 13: Modifications in the kernel configuration file

```

TASK(myTask2)
{
  Delay(9ms);
}

```

Figure 14: Noisy task code

ms, is achieved by placing a delay in the new task code, that we name noisy task.

Including the new task requires modifying the kernel *oil* file, and adding the noisy task, as illustrated in the pseudo-code of Figures 13 and 14. Note also that the main code has been modified to configure the new alarm by a *SetRelAlarm(myAlarm2,1,11)*.

It can be verified that conflicts appear if we overlap the execution of each task, as illustrated in Figure 15. Putting them together under EDF will result in timing variability (jitters) for the control task, as illustrated in the schedule of Figure 16, where the bottom curve is the control task execution (low level means no-execution, middle level means ready to execute but blocked - jitters-, and high level means execution), and the top curve

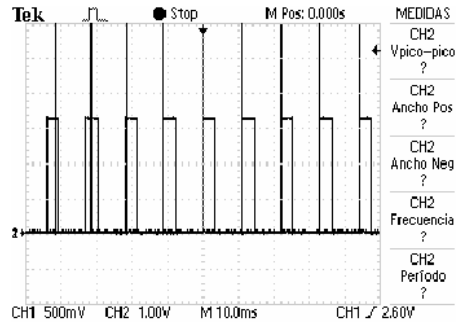


Figure 15: Overlap of the two tasks

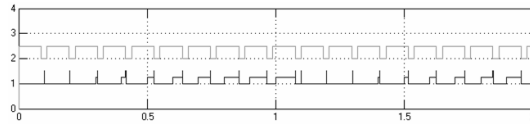


Figure 16: Partial schedule

is the noisy task execution. Note that the control task has a measured execution time of 0.12 ms, which is much less than the noisy task.

Surprisingly, after monitoring the plant response, no control performance degradation was appreciated. This fact indicates that the current control design is robust to scheduling induced jitters.

However, it would be interesting to have a closed-loop system (with the same plant) sensitive to jitters, in order to both show the problem and leave the activity ready for experimenting advanced results that target the jitter problem. With this purpose, the same system was implemented in the Matlab/Simulink TrueTime simulator [9], which facilitates the co-simulation of control tasks execution in real-time kernels, and continuous plant dynamics.

Two strategies were envisioned in order to obtain such a sensitive closed-loop system. The first is to increase the sampling period. This, together with enlarging the noisy task, implies to generate longer jitters (larger variations between sampling operation, and longer time delays between S/A operations). In the second, rather than specifying real poles for the closed loop system, the approach was to select conjugate poles in order to have an oscillatory response. This would permit to better appreciate degradation in the circuit response.

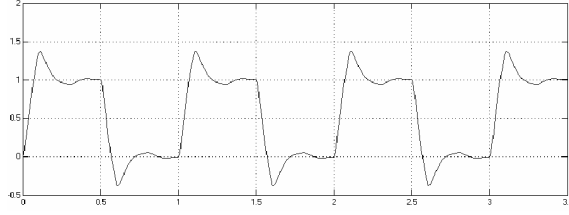


Figure 17: New dynamics

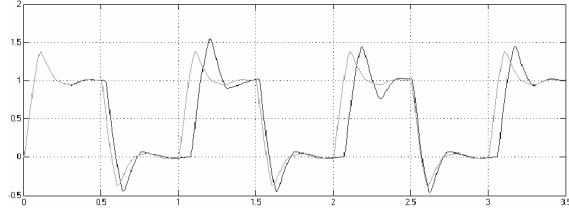


Figure 18: Simulation of the jitters effects

After performing several iterations with the simulator, the continuous-time closed loop poles were specified at $p_{1,2} = -10 \pm 20i$, with a new period for the control task of $h = 100$ ms. Note that the control task execution time remains the same. For this new setting, the control parameters are

$$\begin{aligned}
 K &= [1.0691 \quad -0.0189] \\
 Kr &= -13.4266 \\
 \Phi_{aa} &= 0.3548 \\
 \Phi_{ab} &= 0.0043 \\
 \Phi_{ba} &= -4.2322 \\
 \Phi_{bb} &= -0.0515 \\
 \Gamma_a &= 0.6452 \\
 \Gamma_b &= 4.2322
 \end{aligned}$$

Figure 17 shows the new circuit dynamics if only the control task is executing. Figure 18 shows the previous response (light grey) as well as the circuit response (darker curve) when the control task is sharing the processor with a noisy task with period $h = 110$ ms and execution time 90 ms. The degradation is evident.

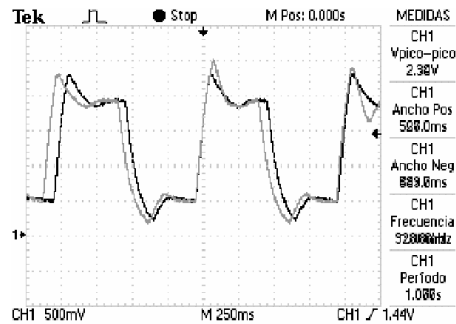


Figure 19: Jitters effects

To verify that in the experimental circuit this effects are also visible, we modified the implementation to include the new parameters. Note also that the reference signal amplitude, for better illustrations, was set to range from 0.5 V to 2 V.

Figure 19 shows the same information as Figure 18 but obtained from the real experiment through the oscilloscope. The dark curve is the system response when the control task is executing in isolation. And the grey curve is the system response when the control task shares the processor with the noisy task. As it can be seen, both figures are very similar, which confirms that jitters are “a problem” depending on the control set-up.

4 TENTATIVE WORK PLAN

The previous section has detailed some of the steps required to successfully carry out the lab activity. In this section we summarize them in a tentative work plan. The work plan is divided into several sessions, assuming that each session is a two-hour lab. It is reserved for future work to detail this program in terms of ECTS (european credit transfer system), program that will be different because ECTS is a student centered system based on the student workload required to achieve the objectives of a program.

S1 - Introduction: Introduction to the activity, and simulation of the open-loop response after obtaining the state-space form of the *RCRC* circuit from the circuit differential equations (1) (consider random values for *R* and *C*). Here we assume that state-space notation is chosen.

S2 - Problem specification (a): This session should be used to specify the problem in terms of the levels for the tracking signal and discrete control

design which includes selecting the sampling period and closed loop pole locations if pole placement is used (other control approaches like optimal control could be used).

S3 - Problem specification (b): To complement the previous session, observers should also be designed and simulated. The outcome of this session should be the complete simulation setup.

S4 - Basic implementation (a): Build the *RCRC* circuit and corroborate its dynamics in open-loop. Start the controller implementation in a periodic hard real-time task in the processing platform.

S5 - Basic implementation (b): Finish the implementation of the controller and corroborate its correctness.

S6 - Multitasking (a): Incorporate a noisy task in the simulation setup to evaluate the jitters effect. This would require to use, for example, the TrueTime simulator.

S7 - Multitasking (b): Incorporate the noisy task in the implementation and validate the results.

S8 - Advanced implementation (a): If degradation in control performance is detected in the previous session, simulate advanced control or real-time techniques that solve the jitter problem.

S9 - Advanced implementation (b): Implement the previous solutions and validate them.

Note that the program layout and the set of covered topics should be adapted to particular needs/background of the target audience or to the goals of the specific curriculum.

5 CONCLUSIONS

This report has presented a laboratory activity to be integrated in the education curriculum of embedded control systems engineers. The activity is a real-time control of a voltage stabilizer. The selection of the plant and processing platform has been discussed. Extensive details of an example have been presented. And a tentative work plan for carrying out the activity has been provided.

We believe that the proposed activity poses several *real* challenges to the students that can be met by putting together interdisciplinary skills (real-time systems, control systems) towards a single goal: building the system.

References

- [1] K.-E. Årzén, K.-E., Cervin, A., Eker, J., and Sha, L. (2000), “An introduction to control and scheduling co-design,” 39th IEEE Conference on Decision and Control.
- [2] Åström, K.J. and B. Wittenmark (1997), *Computer controlled systems*, Prentice Hall.
- [3] Buttazzo, G. (2006), “Research trends in real-time computing for embedded systems”, *ACM SIGBED Review*, Vol. 3, No. 3.
- [4] Custom Computer Services, Inc., <https://www.ccsinfo.com>
- [5] Caspi, P., San Giovanni-Vincentelli, A., et al. (2005), “Guidelines for a graduate curriculum on embedded software and systems”, *ACM Transactions on Embedded Computing Systems*, Vol. 4, N. 3, August, pp. 587 - 611.
- [6] Crespo, A., Vila, F., Blanes, I. Ripoll, (1998) Real-Time Education in a Control Engineering Curriculum, Third IEEE Real-Time Systems Education Workshop.
- [7] Evidence srl., <http://www.evidence.eu.com>
- [8] Halang, W.A. (1990) “A curriculum for real-time computer and control systems engineering”, *IEEE Transactions on Education*, Vol. 33, N. 2, May, pp. 171-178.
- [9] Henriksson, D., Cervin, A. and Årzén, K.-E. (2002), “TrueTime: Simulation of control loops under shared computer resources,” 15th IFAC World Congress.
- [10] Lozoya, C., Velasco, M. and Martí, P. (2008), “The One-Shot Task Model for Robust Real-Time Embedded Control Systems”, *IEEE Transactions on Industrial Informatics*, Vol. 4, N. 3, August.
- [11] Ogata, K. (2001), *Modern Control Engineering*, 4th Edition, Prentice Hall.
- [12] Ricks, K. G., Jackson, D. J., Stapleton, W. A. (2008), “An Embedded Systems Curriculum Based on the IEEE/ACM Model Curriculum”, *IEEE Transactions on Education*, Vol. 51, N. 2.
- [13] Real-time Linux Foundation, Inc., <http://www.realtimelinuxfoundation.org/>