FREERTOS IMPLEMENTATION FOR HIL SIMULATION USING AURIX MULTICORE

Matúš Kozovský
Doctoral Degree Programme (2), FEEC BUT
E-mail: xkozov00@stud.feec.vutbr.cz

Supervised by: Petr Blaha
E-mail: blahap@feec.vutbr.cz

Abstract: This article focuses on the Real-time Operating System (RTOS) implementation for Hardware-in-the-loop (HIL) simulation. Three-core AURIX microcontroller TC277 was chosen for testing. This work describes the distribution of software to individual core to reach required functionality. Basic features of HIL simulation and its usage along with RTOS features are discused.

Keywords: RTOS, AURIX, TRICORE, HIL simulation

1 INTRODUCTION

Nowadays, the reliability and security are key features of any device. Functionality requirements are still growing. Operating systems are more commonly used in embedded systems for this reason. Large amount of different tasks often run in single device. Operating system solution is suitable especially in these applications. Operating system must be predictable and must ensure execution of tasks before their deadlines for these purposes.

Hardware-in-the-Loop (HIL) simulation is also often used for final product reliability testing and simplifies the development of control algorithms. HIL simulation can be used in the final product to monitor deviations from the controlled physical device. For example faults may be not only detected but in some cases they can be predicted using these methods.

2 HIL SIMULATION

HIL simulation is a type of real-time simulation that is used for the development and testing of control systems, control algorithms or for similar purpose. HIL simulation replaces the physical parts of device by a simulation. This method allows you to control simulated device instead of real one. In fact, this approach using HIL simulation shows you how your control algorithm responds, in real time, to realistic virtual stimuli.

Model of real physical part need to by as close to reality as possible for accurate verification of the control algorithm. However, model that would accurately reflect the reality and reacts to all possible environmental effects is not possible to create. The more accurate model requires more computing power. Especially fast continuous states are extremely computionaly demanding.

Proper connection between control platform and hardware for HIL simulation must be realised to create the whole system. Hardware of control algorithm and hardware of HIL simulation can be separated and interconnected by control signals in the same way as they are connected in physical device [1]. Real part and HIL simulation hardware can be switched easily in this case. Another option is to implement control algorithms and simulation into one hardware. Control algorithm needs to be transferred to target platform after testing and verification process in this case. If HIL simulation is implemented in hardware with the control algorithms, it can be for example used to detect faults in the physical part.

HIL simulation can be implemented in various hardware platforms. Simulations requiring more computing power can be simulated using PC with additional extended cards (Field Programmable Gate Array (FPGA) processing card, Graphic Processing Unit (GPU) for calculation, etc.). These methods are generally used [2]. Suitable platform can be determined according to specific requirements [3].

Simulations that require less processing power can be realised using microcontroller. If the microcontroller has sufficient computing power, HIL simulation can be also used for fault detection in final implementation, as mentioned before.

Another advantage of this solution is that control algorithms faults during testing and development can not cause damage of physical system, because controlled part is replaced by HIL simulation. The whole system can be restarted in a case of failure. It is also possible to analyze the fault and modify control algorithm to avoid same faults within real system.

In order to simulate some systems it is appropriate to divide the whole simulation into several separated problems. For example, model of motor can be divided into several parts. Electric part has faster dynamics, mechanical part is slower and thermal part of a motor model is the slowest part. Individual parts of the model can be calculated with different sampling periods. Microcontroller can also perform diagnostics task during HIL simulation and calculate control algorithm. Operating system is suitable for managing individual tasks.

3 OPERATING SYSTEM AND FUNCTIONALITY

Nowadays, great emphasis is placed on the reliability and security of devices. Control system must allow not only the proper control of target physical device, but also self diagnosis and safety monitoring are also necessary for this reason. These demands require not only higher computing power but also proper planning of individual tasks in the CPU. Therefore, control system must be able to perform multiple tasks in one time. In reality, each processor core can only be running a single task at any given point in time. Operating system provides proper switching between individual tasks. Scheduler, one of the most important parts of operating system, is responsible for switching and planing tasks. The scheduler of Real Time Operating System (RTOS) is designed to be deterministic. It mean that task switching and planning is predictable according to precise mechanism and external conditions.

RTOS can be optimized for a classical PC, for a server, as well as for embedded system, which is most commonly used for control applications. Classic Control algorithm or HIL simulation must respond to a certain event within a strictly defined time (the deadline). This can be guaranteed only using predictable scheduler [4].

Priorities of each task, which is executed by operating system, must be properly configured during writing program. The most important and critical tasks have the highest priority. Lower priority tasks are executed at a time when all other higher priorities tasks are in waiting state. Individual tasks can wait for a external event (interrupt) or timeout for periodical tasks [5]. An example of switching individual tasks with different priorities is shown in Figure 1.

FreeRTOS was chosen for implementation testing and practical tests in the target hardware. This version of RTOS is suitable for microcontrollers and can be found for target microcontroller.

4 IMPLEMENTATION

Tri-core microcontroller AURIX TC277 was chosen for testing and final implementation of solution. One core is reserved for the monitoring and diagnosis of running. This core supervises to another

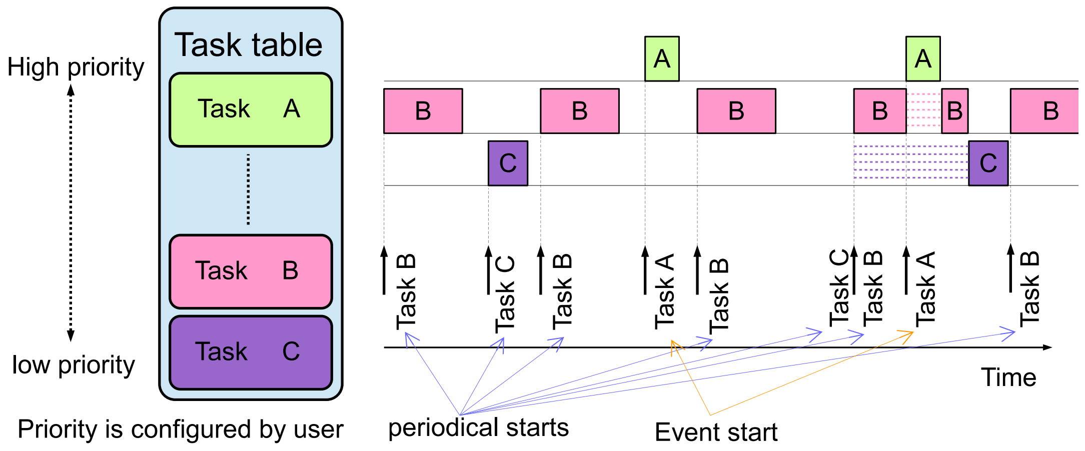

Figure 1: RTOS Task management.

cores for safety reason. Second core is dedicated for HIL simulation. Two periodical tasks run on this core, one simulates electrical and mechanical part and second one simulates thermal behavior of the motor. Third core is dedicated for communications and control algorithm. Control algorithm of the motor should run periodically. Control algorithm is calculated once per PWM period. Task for communication is asynchronous. Task execution depends on the master system requirements using communication line. Distribution of microcontroller tasks is shown in Figure 2.

4.1 FAST PERIODICAL TASK

Nowadays, motors speed is mostly controlled using electric inverters. Switching frequency of these inverters is around 10-20 kHz. HIL simulation to replace real motor should compute states of motor for every period of PWM signal. Task to compute electrical part of HIL simulation should be calculated at least every 100 μs. Mechanical part is calculated using the same period. Thermal part is calculated using ten times longer period.

Periodical task down to 1 ms can be realized using RTOS features. However, 100us periodic task is too fast for FreeRTOS general timer. Suitable solution of this issue is using microcontroller hardware timer to generate periodic interrupt every 100 μs. This interrupt subsequently uses semaphore to allow execution of another period of electrical and mechanical part of HIL simulation task. Control algorithm task of Core 1 and HIL simulation task of Core 0 are synchronized using the same method. Simulated variables such as motor currents, speed and position are forwarded co control algorithms using shared memory. The same method is used for PWM signals from control algorithm to HIL simulation.

5 CONCLUSION

The final application which was created based on the above mentioned theory was successfully created and tested. Measurement confirmed that individual tasks are executed in expected time intervals. Execution sequence of tasks is also due to prediction.

Software for HIL simulation and for control algorithms was generated by MATLAB automatic code

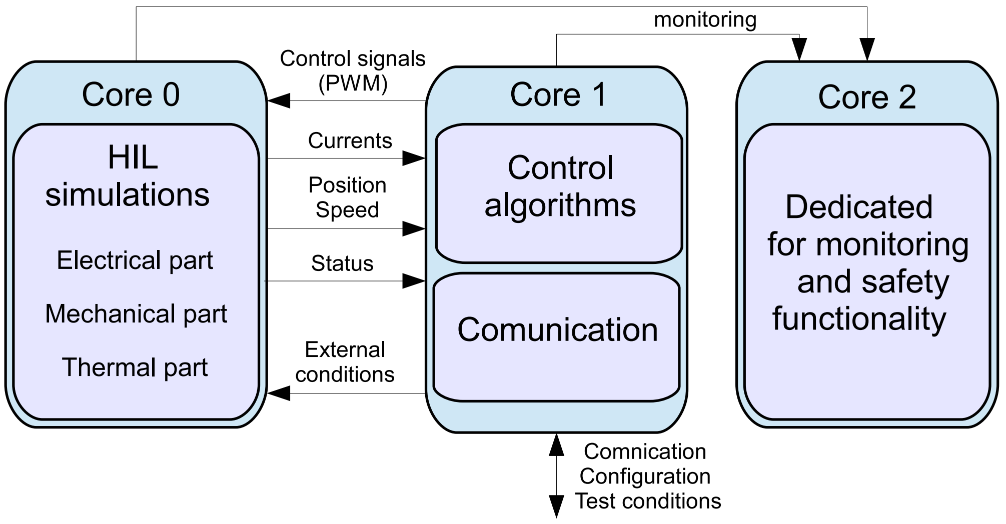

Figure 2: Function distribution into microcontroller cores

generation. The results of HIL simulation and control algorithm implemented to embedded system were consistent with the simulations. The real motor tests require connection of control signals to voltage source inverter.

ACKNOWLEDGEMENT

This work was supported by ECSEL JU under the project 621429 EMC2.

The completion of this paper was made possible by the grant No. FEKT-S-17-4234 - „Industry 4.0 in automation and cybernetics" financially supported by the Internal science fund of Brno University of Technology.

REFERENCES

[1] S. Abourida, J. Belanger, and C. Dufour, "Real-time HIL simulation of a complete PMSM drive at 10 /spl mu/s time step," in 2005 European Conference on Power Electronics and Applications, 2005, p. 9 pp.-pp.P.9.

[2] J. J. Poon, M. A. Kinsy, N. A. Pallo, S. Devadas, and I. L. Celanovic, "Hardware-in-the-loop testing for electric vehicle drive applications," in 2012 Twenty-Seventh Annual IEEE Applied Power Electronics Conference and Exposition (APEC), 2012, pp. 2576–2582.

[3] C. Dufour, S. Cense, V. Jalili-Marandi, and J. Belanger, "Review of state-of-the-art solver solutions for HIL simulation of power systems, power electronic and motor drives," in 2013 15th European Conference on Power Electronics and Applications (EPE), 2013, pp. 1–12.

[4] F. E. Paez, J. M. Urriza, R. Cayssials, and J. D. Orozco, "FreeRTOS user mode scheduler for mixed critical systems," in 2015 Sixth Argentine Conference on Embedded Systems (CASE), 2015, pp. 37–42.

[5] C. S. Stangaciu, M. V. Micea, and V. I. Cretu, "Hard real-time execution environment extension for FreeRTOS," in 2014 IEEE International Symposium on Robotic and Sensors Environments (ROSE) Proceedings, 2014, pp. 124–129.