15th Triennial World Congress, Barcelona, Spain

TESTING EMBEDDED CONTROL SYSTEMS USING HARDWARE-IN-THE-LOOP SIMULATION AND TEMPORAL LOGIC

Marco A.A. Sanvido, Vaclav Cechticky
Walter Schaufelberger

Automatic Control Laboratory
Swiss Federal Institute of Technology
Physikstrasse 3
CH-8092 Zürich

Abstract: In this paper a method for testing the implementation of embedded control systems using a hardware-in-the-loop simulator (HILS) and a temporal logic tester is proposed. The goal of the simulator is to replicate a given dynamical process, to be able to generate faults and to automatically analyze the Embedded Control System (ECS) response against a temporal logic specification. The paper explains and demonstrates this technique by using a simple example. The HIL simulation is implemented on an Oberon (Wirth and Gutknecht 1992) platform, the controller used for the example is implemented on the Java real-time platform JBed (Esmertec n.d.).

Keywords: Embedded systems, Simulation, Temporal logic

1. INTRODUCTION

Due to the rapid development of digital-processor technology, ECS control many devices used in daily life. Moreover many of these systems operate in safety-critical situations and therefore call for a rigorous engineering. This crucial point is the reason why so much research effort is put into the development of methodologies for the design, development, implementation and testing of ECS (O'Connor 2001).

Hardware-in-the-loop simulation (HIL simulation or HILS) is a kind of real-time simulation were the input and output signals of the simulator show the same time dependent values as the real process, see (Isermann 1999, Ledin 2000). Such simulators allow to test the *real* embedded control system (ECS) under different *real* working loads and conditions. Other simulation methods do not allow to test the real embedded control system as a complete system. Often the controller part, which is about only 20-30% of the ECS software (Pasetti and Pree 2001), or the software components are tested independently. HILS makes it possible to test the complete ECS system. Moreover the temporal logic tester and the fault generator allow to automatize the testing procedure, which is usually done manually by an operator.

In developing such products and systems, testing and not design is usually the more expensive, time-consuming and difficult activity – cited from the IEEE Spectrum magazine (O'Connor 2001). Therefore not only a tool for helping in the design and implementation plays an important role but also tools which allow the implementation of testing environments.

The paper is subdivided as follow: the first section describes the steps needed to design and test an ECS system. The second section motivates and explains the concepts of this approach. The third section demonstrates in detail how the system works by means of a simple example. Some conclusions are made in the last section.

2. DESIGN OF AN EMBEDDED CONTROL SYSTEM

In Figure 1 a schematic view of the design process of an ECS is shown. As you see, many different steps are needed in the development of a complete ECS system. For many of these steps well know theoretical approaches and methods have been studied and successfully applied to real world problems. There are however some steps were a mature methodology is missing. One of such steps is the testing of ECS, more precisely the on-the-fly testing of the final ECS system under real working conditions. This approach goes in this direction allowing to automatize the testing procedure.

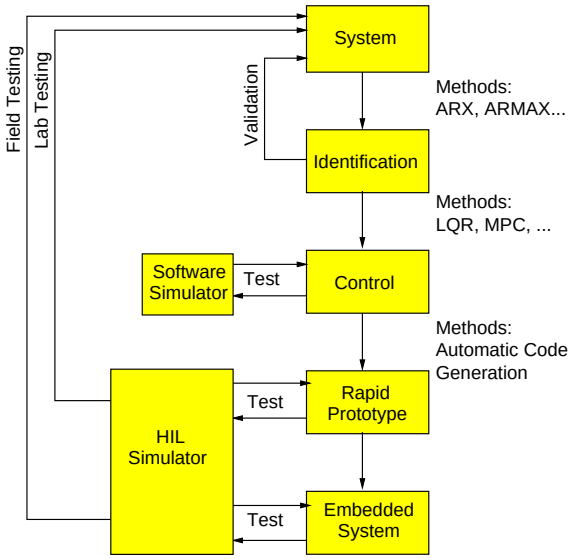

Fig. 1. Control system design

3. CONCEPT

The simulation environment is shown in Fig. 2. The ECS sends its outputs to the process, which is replaced by the simulator, and the simulator sends the sensor data back to the ECS. Since the ECS is working at a given sampling rate the simulator must compute the new values at least at the same rate. Moreover the hardware interface between ECS and process and between ECS and simulator must be somehow implemented. These problems are analyzed more detailed in (Sanvido and Schaufelberger 2001), and they are not a topic of this paper.

The simulator computes the system dynamics from the mathematical model of the process, and is able to generate faults which will be propagated to the ECS. The fault generation is done by using a fault description and specification language, called *FAUSEL*. This language allows to specify how, and when faults are generated during a given simulation run, but in addition it allows to specify the correct reaction of the ECS to the fault. The fault generation is implemented using an object-oriented framework which simplifies the software implementation of the fault generation, whereas the fault specification is done using an extension of LTL – Linear Temporal Logic (Clarke 1999)–, called MTL – Metrical Temporal Logic (Manna and Pnueli 1992). This logic was chosen for two reasons, because boolean expressions are not strong enough to express the temporal behavior of an ECS response, and also because temporal logic is a standard specification language used in formal verification. The MTL specification is transformed by FAUSEL in an Alur-Dill timed automaton (Alur 1999). The automata will be traversed at each simulation step in order to detect a correct/incorrect ECS response.

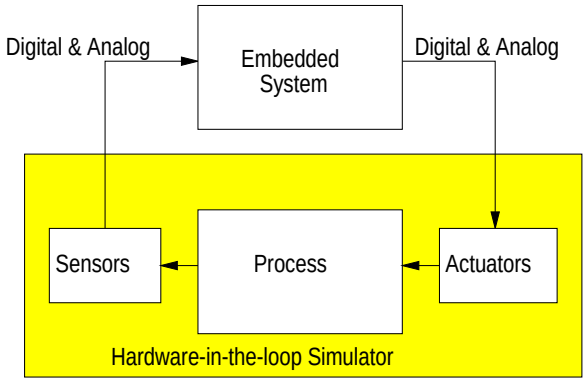

Fig. 2. HIL simulator

4. A SIMPLE EXAMPLE - A BARRAGE SIMULATOR

In this simple example the application of the simulation framework to a real problem is demonstrated. The simulator simulates an hydro-power plant barrage (see Fig. 3), the barrage is composed of two moving parts, the segment and the flag. The segment is moved by the motor α, and moves the barrage upward or downward ($h1$). The water in the reservoir can flow below the barrage from the aperture $h1$. The flag is located on the top of the segment and is moved with motor β, the flag can change the $h2$ height. The water can thus flow over the barrage.

The dynamical model of the barrage is very simple and similar for the motors α and β. In this paper only motor α is discussed . To use a complex barrage dynamical process model and a complex control system is not the goal of the simulation. The goal is to be able o automatically check if erroneous conditions, which can happen on the barrage, are correctly detected and properly handled by the ECS.

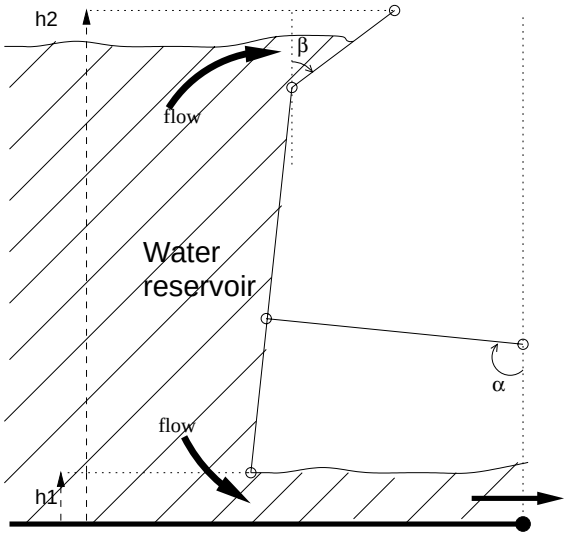

Fig. 3. The barrage

The possible faulty conditions to be tested are:

- **Leak:** an oil leakage from the motor α forces the barrage to close, reducing the high $h1$.
- **Stuck:** the segment cannot be completely closed because something is blocking the segment.

These two faults must be detected and a fault reaction procedure has to be started by the control system. In the leakage case the controller will simply maximize its output power, in order to compensate the position error, and release an alarm. In this way the controller is able at least to stop the segment from closing. In the stuck case the controller will try to open the segment in order to clean the reservoir ground, and thereafter will re-try to close the segment.

The simulator and the ECS are running at a 5Hz frequency, and communicate via a simple serial connection.

```
SIMULATE Barrage;

    FAULT W0 IS Barrages.Stuck;
        START 0.0;
        STOP 50.0;
        PERIOD 0.2;
    SPEC (* open the barrage to clean*)
        F((alpha <= 847) & F(alpha>=945))
    END W0;

    FAULT F1 IS Barrages.Leak;
        START 10.0;
        STOP 30.0;
        PERIOD 0.2;
    SPEC (*detect the alarm *)
        F(G[0,5](u0 > 1))
    END F1;

END Barrage.
```

Fig. 4. FAUSEL Description

After starting the simulation and compiling the FAUSEL description (see the code of the used FAUSEL description in Fig. 4, the generated automata are shown in Fig. 5) the ECS will control the barrage to a given reference position for the motors α and β. The simulator will analyze the ECS response, traversing the automata generated by FAUSEL and will compute the new actual position of the barrage.

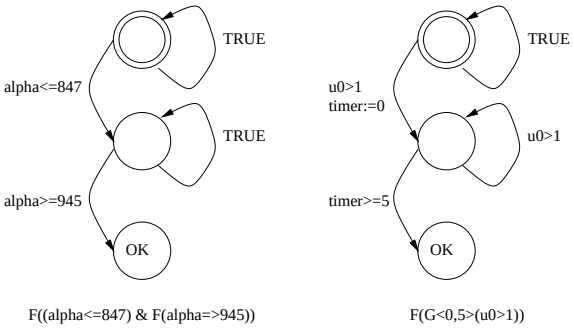

Fig. 5. FAUSEL generated automata

Figure 6 shows the user interface of the simulator. The user interface shows also a simplified representation of the barrage and the actual simulated position of the segment and flag.

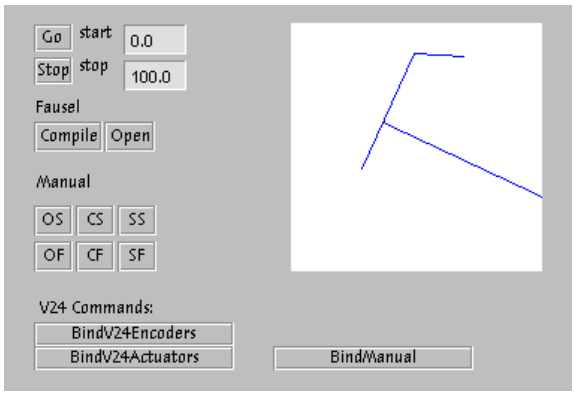

Fig. 6. Barrage simulator interface

4.1 The Controller and Diagnostics Implementation

The control system consists of two interconnected parts, the controller and the diagnostic system. The inputs and outputs of the barrage systems are analyzed by the diagnostic system in order to detect failures and to ask the controller to correct them. The barrage position is controlled by two independent P controllers and the diagnostic system is able to detect two types of faults.

The control system is implemented in the object-oriented language Java (Sun Microsystems n.d.) and is running on the embedded real-time operating system Jbed (Esmertec n.d.). The class diagram in Fig. 7 is a graphical representation of the implemented control system according to UML methodology. The class *SerialConnection* establishes a serial link connection and communication, the class *PController* implements a P-controller, the class *FaultDetection* provides an

algorithm for data analysis, and the class *FifoQueue* is first-in first-out queue. An instantiation of the classes is done in the class *BarrageControlSystem*. The class *BarrageControlSystem* has only one method *main* which implements the control system and is called when the ECS is started.

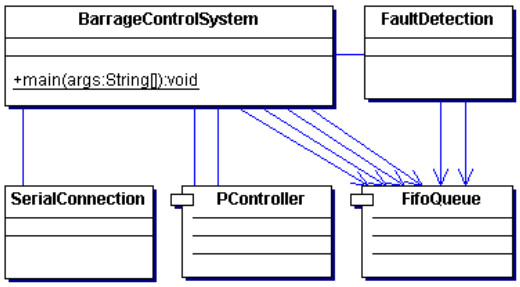

Fig. 7. UML diagram

The control system was developed on the host computer and then compiled into Java bytecode by the Java compiler included in Java2 JDK 1.3 from Sun Microsystems. Subsequently the bytecode is compiled into native machine code, linked with Jbed operating system components by the linker in the Jbed IDE, and downloaded to the target system via a TCP/IP connection. The hardware-in-the-loop simulation of the barrage is running on the host computer as shown in Fig. 8, or on any other machine that allows to run the barrage simulator and supports the serial communication for data exchange with the target system. As target system the RPX Lite (Embedded-Planet n.d.) board based on the Motorola MPC823 intended for industrial control applications (Motorola n.d.) was used.

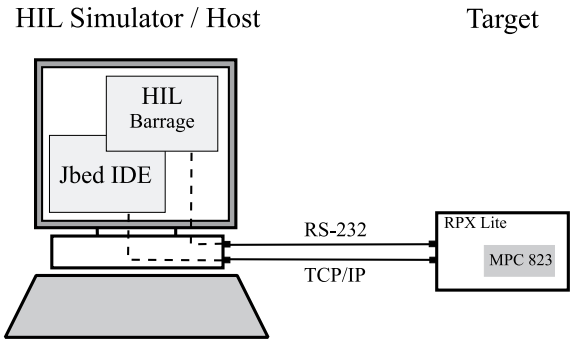

Fig. 8. HIL simulator/Jbed host and target systems

4.2 Results

This section shows the results of two simulation runs, in the first the controller opens the segment to position 1100. After 10 seconds the simulator will start to generate a leakage fault and the correct ECS responses are shown in Fig. 9 and in Fig. 10.

In Figure 11 the ECS response is shown, when no faults are generated. The correct ECS response is specified in FAUSEL as

$$F(G[0,5](u0 > 1))$$

meaning that eventually after the fault generation the motor α input will augment its output over the normal limit of 1 for at least 5 seconds. This specification is automatically verified by the simulator.

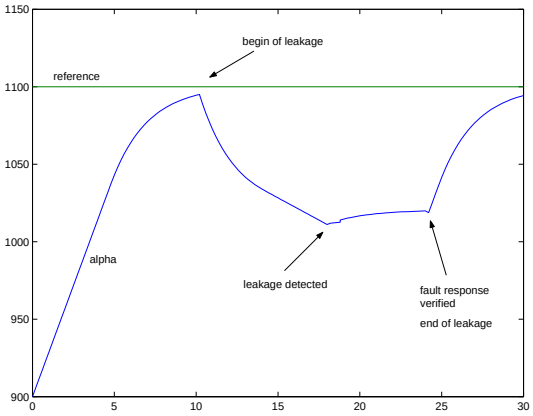

Fig. 9. Leak-fault response

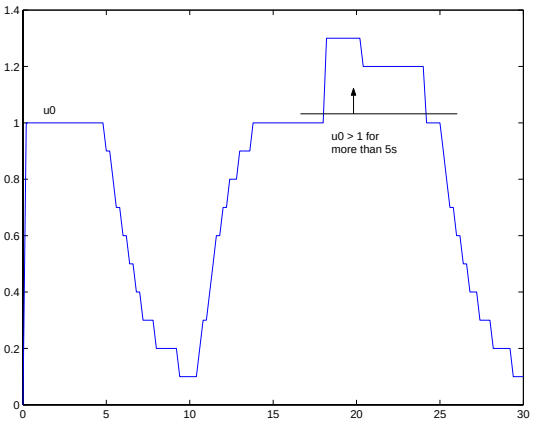

Fig. 10. ECS control output

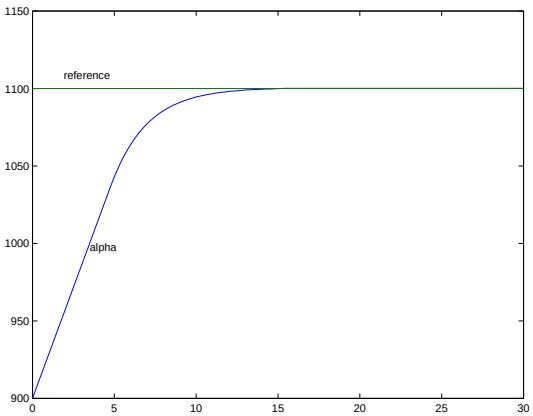

Fig. 11. No-leak response

In the second simulation run, at time 0s the *stuck* fault is started and kept for 50s, or till the specified response is verified. In this case the specification is

$$F((alpha <= 847)\&F(alpha >= 945))$$

meaning that eventually after the fault started the segment will go below 847, i.e. it will touch the obstacle, and after that eventually will go higher then 945. A temporal limitation could also be added to specify a maximal response time of 5s and 0s minimal reaction time. The specification would be

$$F((alpha <= 847)\&F[0,5](alpha >= 945))$$

Figure 12 shows the response of the ECS in presence of a stuck fault, in Fig. 13 the ECS response in the normal case.

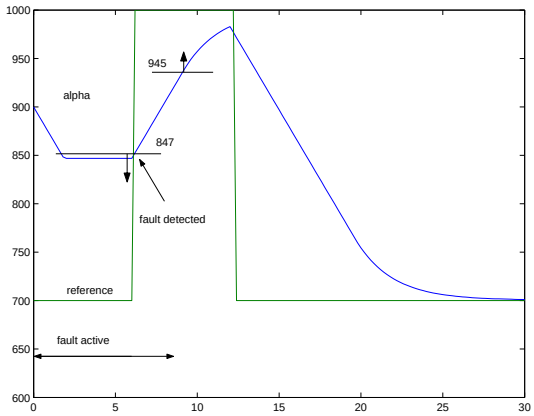

Fig. 12. Stuck-fault response

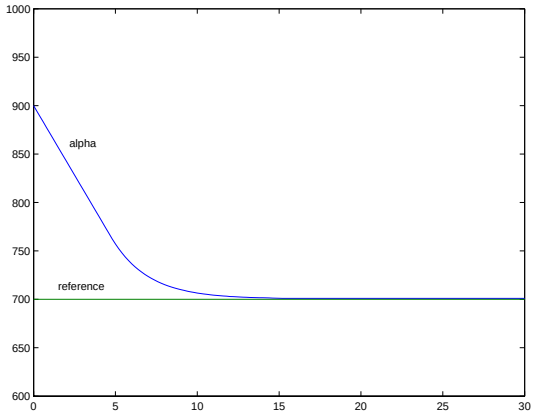

Fig. 13. No-stuck-fault Response

5. CONCLUSIONS

In this paper an approach is demonstrated, which allows a more accurate, precise and automated testing of ECSs. The HIL simulator has also been used to test – only partially – other ECS like the OLGA system (Chapuis 1999), which is an autopilot system for a model helicopter. This approach can be used also for real industrial problems with no changes, the only limitation is the temporal specification complexity, since the translation from MTL to timed automata can produce very large automata.

The barrage example was inspired by a real industrial barrage system but is still far from the complexity of a real application. However we think that such an automatic testing facility would simplify and automatize the testing procedure of an ECS in its final development stage, which is usually done be simple manual checking of the system.

Currently, we are porting the system to a more open Java platform, which will allow other people to use and test the approach. The actual software, implemented on a modified Oberon platform (Wirth and Gutknecht 1992), is not well suited to distribute to a large community. Moreover, we also integrating the HIL Simulator within Matlab by using Matlab's Java interface (Schär and Samedani 2001).

REFERENCES

Alur, R. (1999). Timed automata. In: *Proceedings of the 11th International Computer Aided Verification Conference*. Springer Verlag. pp. 8–22.

Chapuis, J. et al. (1999). Control of helicopters.. In: *Control of Complex Systems*. Springer Verlag. pp. 359–392.

Clarke, E.M. et al. (1999). *Model Checking*. MIT Press.

Embedded-Planet (n.d.). http://www.embeddedplanet.com.

Esmertec (n.d.). http://www.esmertec.com.

Isermann, R. et al. (1999). Hardware-in-the-loop simulation for the design and testing of engine-control systems. *Control Engineering Practice* **7**, 643–653.

Ledin, J.A. (2000). Hardware in the loop simulation. *Emdedded System Conference*.

Manna, Z. and A. Pnueli (1992). *The Temporal Logic of Reactive and Concurrent Systems*. Springer. New York.

Motorola (n.d.). http://www.motorola.com.

O'Connor, P.D.T. (2001). Neglect testing at your peril. *IEEE Spectrum* **38**(7), 18.

Pasetti, A. and W. Pree (2001). Embedded software market transformation through reusable frameworks. In: *Proceedings of the First Embedded Software Workshop* (T. Henzinger and C. Kirsch, Eds.).

Sanvido, M. and W. Schaufelberger (2001). Design of a framework for hardware-in-the-loop simulation and its application to a model helicopter. In: *CD: Proceedings of the 4th International EuroSim 2001 Congress* (A. Heemmink and L. Dekker, Eds.).

Sanvido, M.A.A. (2002). Hardware-in-the-loop Simulation Framework. PhD thesis. ETH Zurich.

Schär, C. and R. Samedani (2001). HIL framework for Java. Master thesis. Automatic Control Laboratory, ETH Zurich.

Sun Microsystems (n.d.). http://java.sun.com.

Wirth, N. and J. Gutknecht (1992). *Projekt Oberon - The Design of an Operating System and Compiler*. ACM Press. New York.