Testing of Hybrid Real-time Systems Using FPGA Platform *

Jan Krákora and Zdeněk Hanzálek
Czech Technical University in Prague,
Faculty of Electrical Engineering, Department of Control Engineering
Karlovo náměstí 13, Prague 2, 121 35, Czech Republic
{krakorj, hanzalek}@fel.cvut.cz

July 21, 2006

Abstract

This paper presents a hybrid Hardware-in-the-Loop (HIL) methodology based on both the discrete event system, given by timed automata, and the continuous systems, given by difference equations.

The methodology is implemented using an FPGA platform. It guaranties not only the speed enhancement but also the time accuracy and extensibility with no performance loss. Compared to the operating system based platforms, the FPGA platform is able to achieve much faster sampling frequency.

Methodology FPGA implementation is generated by using Xilinx System Generator, bit exact toolbox for Matlab/Simulink.

Keywords: Hardware-In-the-Loop, FPGA, Real-time system testing, Hybrid Systems, Timed automata, Difference equations

1 Introduction

Hardware-in-the-Loop (HIL) applications are used by design and test engineers to evaluate and validate e.g. vehicle components (electronic control units etc.) during development of new systems. Rather than testing these components in complete system setups, HIL allows the testing of new components and prototypes, so called Implementation Under Test (IUT).

Replacing the rest of the system by a model implemented in a computer (Tester tool) the solution greatly reduces the size and complexity of IUT and increases the flexibility and rate of running many test scenarios. The physical components being tested respond to the simulated signals as if they were operating in a real environment. Therefore they can not distinguish between the signals sent by other physical components and signals provided by models running on a computer.

This paper presents design methodology of a hybrid tester tool. It is based on both the discrete event system theory, given by timed automata [8], and the continuous systems theory, given by difference equations [17, 18].

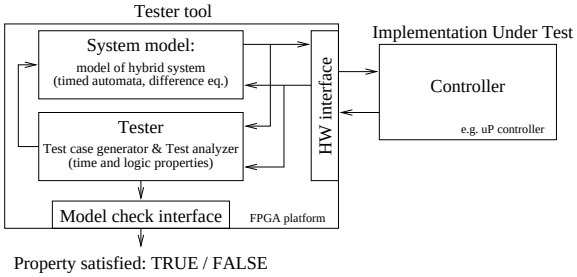


Figure 1: Hardware in the loop conception

Our tester tool is implemented by using FPGA platform that guaranties not only the speed performance but the time accuracy and quite good extensibility with no performance loss as well. Compared to the operating

*This research work has been supported by Ministry of Education of the Czech Republic under project 1M0567 and by Europian Commission under project FRESCOR IST-034026

system based platforms, the FPGA platform is able to achieve much faster sampling frequency. Moreover, the FPGA platform is not affected by rather complex behavior of the operating system services, interrupt handling etc. In contrast to the typical sampling period of 10 μsec, achieved in real-time operating systems like RTLinux [4] or OCERA [3], the sampling period less than 100 $nsec$ can be achieved on FPGA platform. The more important is the fact that, the FPGA platform has zero jitter since it is synchronous HW, and separate parts do not influence each other. On the other hand the operating system based platforms are well supported by widely used development tools.

1.1 Background

The hybrid tester tool combines discrete-event system, in the form of timed automata, and continuous systems in the form of difference equations.

Timed automata [8] are finite state automata, consisting of states (locations) and transitions, among this states. The transitions are extended by clocks that are used to specify the quantitative time. The clocks are variables having non-negative real values, that can be reset. The difference of two clocks can be compared to a constant. In the initial system state, all clock values are null, then all evolve at the same speed, synchronously with time.

The main feature of timed automata is, that it can be verified using temporal logic [12]. It allows user to check if any specification property of model is satisfied or not. It is suitable to inspect model due to e.g. deadlock, logical and timed conditions.

UPPAAL tool [10], used in this article, allows the user to design timed automata and to verify the automata using the mentioned temporal logic. Moreover, it offers a XML API for model export as well.

Timed automata implementation into FPGA is inspired by [7]. The paper shows a way, how to transform a timed automata (TA) into a program. The global time is captured in variable now_clk, the only running clock (i.e. incremented each sampling period). Even though the clock is a real valued (sometimes called dense time) in timed automata, it may be represented by an integer value (thus belonging to systems with sparse time) when assuming periodic sampling. For each clock of timed automaton there is one static variable clk. Such variable is set to variable now_clk whenever the clock is reset. The difference between now_clk and clk represents the clock value. Each channel, used to synchronize the evolution of two timed automata, is replaced by one logic variable which is triggered synchronously with all other channels in the model.

The continuous part [18] of the tester tool is realized using difference equations that are able to represent a continuous system with periodic sampling (called discrete-time systems). The transfer function $G(z)$ of such system determines the physical layout of arithmetic and storage units in FPGA.

The tester tool was implemented in high performance FPGAs, where several arithmetic operations can be executed within the clock cycle close to 10 $nsec$ (Xilinx Virtex II XC2V1000-5FG456C). As presented, a discrete-event system as a set of timed automata is designed in UPPAAL. A continuous system is designed as a transfer function using Matlab control toolbox. Both discrete-event and continuous system are transformed into FPGA using Xilinx System Generator, Matlab/Simulink toolbox (XSG). The toolbox can generate the final VHDL code for a target FPGA platform.

1.2 Related works

Testing can be either off-line or on-line. The off-line testing generates test cases where the complete test scenarios and properties verdict are computed a-priory and before test execution. The off-line test generation is often based on a coverage criterion of the model, on the test purpose of fault-model.

The on-line testing combines test generation and execution. A single test primitive is generated from the model at the time which is then immediately executed on the IUT. Then the output produced by the IUT as well as its time of occurrence is checked against the specification. The test case is produced until it is decided the end of test.

The related tools are briefly described in the following paragraphs. Each tool comparison to our approach is appended.

TTG tool [13] is an off-line test generator. The tool is based on the top of IF modeling language [11] and it is able to generate a set of test cases for an IUT represented by a timed automata. Compare to our approach the tool proposes the off-line tests of discrete event system. Moreover, it can not test using hybrid approach.

The UPPAAL-TRON [14, 16] is a tool for on-line testing based on the discrete event system description using timed automata. The tool both generates test cases to IUT and checks test properties to be verified at

the same time. The tool is executed at a UNIX platform, it can not guarantee the system time resolution of FPGA. Moreover, it is not able to test an IUT using hybrid system approach.

The SoftCom [19] is an on-line industrial HIL application that simulates an industrial environment activities to react with an IUT e.i. programmable logic controller (called PLC). The application is based on discrete event systems based on state machines. However the SofCom is an application developed for Windows OS or non real-time Unix OS. Those system time resolution is approximately 100 *msec*, thus very high. Moreover, it does not support hybrid system testing.

LabView FPGA [1] in HIL application allows an user to test IUT on-line. The user can design its own HIL using LabView environment supporting both discrete event and continuous systems. Such HIL can be directly set into FPGA. However, compared to our approach the tool misses the discrete-event system timed automata representation.

Another tool CarMaker [6] is used for both on-line and off-line testing of a vehicle embedded control systems. The system is used for developing of vehicle embedded control systems interacting with discrete event and continuous system environment as well. Nevertheless, the CarMaker similarly to LabView FPGA does not support timed automata.

1.3 Outline

The paper structure is the following. The section 2 presents our hybrid HIL application methodology. The HIL discrete event system implementation using timed automata and continuous system implementation using difference equations are presented here. The section 3 shows two case studies. The first study describes HIL implementation using discrete event approach only, the second one shows the HIL implementation combining both discrete event systems and continuous systems.

2 Tester tool implementation issues

HIL achieves a highly realistic simulation of equipment in an operational virtual environment. A typical HIL system includes a IUT and an analysis platform, tester tool.

The structure of our HIL system conception is depicted in figure 1. There are two main blocks, a tester tool and an IUT. The tester tool produces an output test signal to influence IUT behavior and it reads an input signal to verify its reaction.

The tester tool consists of four parts.

System model specifies the system to be controlled by the controller IUT. It generates an output signal to IUT (IUT input signals) and it reacts to an input signal from the implementation.

The system model implementation is given either by discrete event system description (e.g. timed automat) or by continuous system representation (e.g. difference equation). The discrete event system part is based on timed automata.

The models are designed using UPPAAL tool [10] and transformed using UPPAAL UTAP library [9]. Continuous system model, designed in Matlab/Simulink, is described by difference equations [15, 18].

Tester block generates test cases and checks the behavior of the IUT. A test case generates a set of signals that are transmitted into a System model, influencing its states. Consequently, the tester analyzes the input and output signals and verifies the properties of IUT given by its IUT specification. Tester results are propagated via the Model check interface.

The test case generator and test analyzer implementation are based on discrete event system representation and on continuous system as well. The test analyzer is able to affect both time and logic properties.

HW interface transmits data signals from System model to the IUT and vice versa. It performs A/D and D/A conversion when appropriate e.g. a pulse width modulation (PWM).

The interface is implemented using common input and output ports of the target platform i.e. FPGA.

Model check interface is a block that interprets a test analyzer result to the user. It signals when a given IUT property is satisfied or not.

The tester tool internal signals are provided using variable data exchange. The external communication between Tester tool and controller is represented by digital or analog signal.

2.1 Tester tool implementation

The tester tool has been implemented on FPGA platform. The FPGA platform allows a designer to create its structure as needed. Moreover, it allows to implement complex real-time structures and implement a parallel architectures. Thanks to the platform, the tester tool is fully configurable and time accurate.

The tester tool was designed using Xilinx System generator (XSG), a FPGA toolbox for the Matlab/Simulink IDE [5]. It gives a possibility to design a FPGA infrastructure in user friendly interface without deep knowledge of a complex low level languages like VHDL [20].

The tester tool implementation issues, like state machine, timed automata or channels using the System generator are presented bellow.

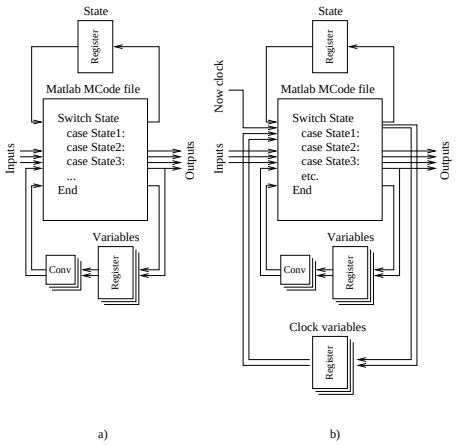


Figure 2: State machine and timed automaton implementation in XSG

2.1.1 State machine implementation

The state machine structure, implementable in XSG, is shown in Figure 2a). The automaton consists of the state register, the Matlab MCode file (MCode) and the set of local variables with their data type converters.

The state register stores the value of actual automaton state. The state value is updated depending on condition set in MCode.

The MCode is a file with a Matlab function with a *Switch-Case* statement in Matlab like notation. The statement structure is given by a structure of an accordant automaton.

Each *Case* statement represents location and each *if* statement in the *Case* represents state to state transition constrain.

The registers are used to store an eventual automaton variable. Because the variable register data type can differ from the required input data type of MCode, such variable have to be converted using data type converter.

A MCode file content example is depicted in listing in figure 3.

2.1.2 Timed automata implementation

Our timed automata implementation is inspired by [7]. The paper shows the transformation of timed automata into a program. The global time is captured in variable now_clk, the only running clock. Even though the clock is a real valued in timed automata, it may be represented by an integer value when assuming periodic sampling. For each clock of timed automaton there is one static variable clk. Such variable is set to variable now_clk whenever the clock is reset. The difference between now_clk and clk represents the clock value. Each channel, used for synchronizing the evolution of two timed automata, is replaced by one logic variable which is triggered synchronously with all the other channels in the model.

The TA implementation into FPGA is similar to state machine implementation described in the section above. The TA implementation consists of a state register and TA, the state machine structure is given by MCode. Moreover, the now_clk generator and one clock register per each timed automaton clock variable were added. Each clock is updated depending on MCode structure.

The clock evaluation example is depicted in figure 4. It presents the now_clk and clk history of variables and it shows an example of periodic reset of variable clk. The clk sampling period is based on now_clk. Its reset time is given by timed automata constrains in MCode.

```
function [state_new, channel_req, clk_new] =
    TA1(state, now_clk , channel_ack, clk)

% Variable initialization
state_new = state;
channel_req = 0;
clk_new = clk;

% Timed automaton state machine
switch state
    case 1 % State1 - the initial state
        if( channel_ack == 1 )
            channel_req = 0;
            clk_new = now_clk;
            state_new = 2;
        else
            channel_req = 1;
        end
    case 2 % State2
        if( (now_clk - clk) == 2 )
            state_new = 1;
        end
    otherwise
        % an error occurred
        state_new = -1;
end
```

Figure 3: MCode listing of timed automata TA1 (see figure 6)

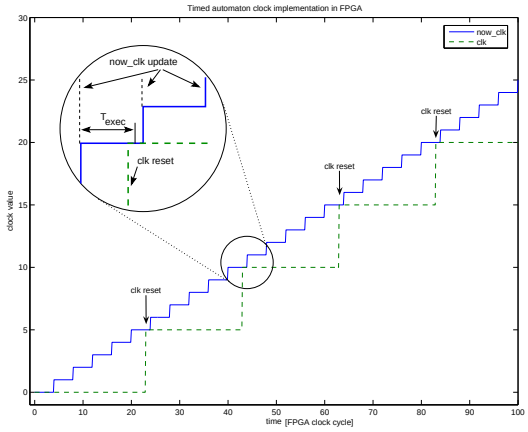


Figure 4: Timed automaton clock implementation

The variable now_clk is incremented each sampling period. The sampling period is given by the length of the FPGA clock cycle and by the number of the FPGA clock cycles needed to evaluate the continuous part and discrete part. The length of the FPGA clock cycle is given namely by the complexity of the arithmetic operations performed in the continuous part (see the chain of the multiplication and addition units in figure 7). The shortest possible FPGA clock cycle is about 10 *nsec*, assuming the continuous system of the third order and 16-bit fixed point logics on Xilinx Virtex II XC2V1000-5FG456C.

The existence of the urgent locations and channels in timed automaton requires repetitive calls of the corresponding timed automaton function (e.g. figure 3). For example, when a sequence of three urgent locations is assumed, then at least four repetitive calls are needed. Therefore the sampling period has to be greater than T_{exec}, the time needed to update all FPGA internal processes. For example the sampling period of the timed automata is set to 4 clock cycles in figure 4.

The implementation of timed automaton location-to-location transition is bounded by a time constrain interval and can be resolved by an approach based on the principle that the only one sample from the interval is chosen. In timed automata, while using invariants and guards, a transition can be taken within given time interval. This non-deterministic feature, related to the system parameter for which only lower and upper bounds are known, can be handled in the tester tool while using stochastic approach. In our example 3 the stochastic

approach can be implemented by using a function randTI() that generates a random value given by a probability distribution function on the time interval. In the case of normal distribution on interval $[2, 4]$ (invariant $clk <= 4$ and guard $clk >= 2$ in figure 6a), the code line 20 should be changed into if((now_clk-clk) == randTI(normal,2,4)).

Channel implementation Channel primitive implementation for the synchronization of state automata is shown in figure 5. The figure presents two automata using one channel to coordinate its activity.

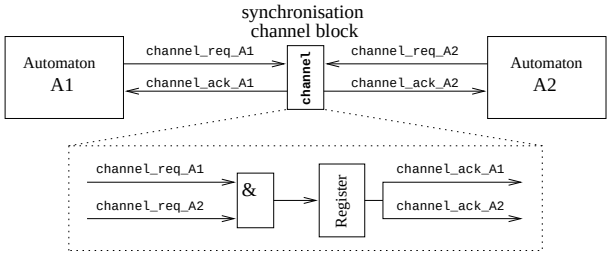


Figure 5: Channel implementation

Each automaton uses two variables. The first variable, channel_req_ax, activates the synchronization process, the second one, channel_ack_ax acknowledges the automaton that synchronization process is done.

The channel structure is depicted in the dotted box in figure 5. The structure uses one logic operand AND and one register to avoid algebraic loop.

Timed automata example An example of timed automata transformation into XSG is presented in the following paragraphs.

Figure 6 shows two timed automata in UPPAAL like notation. Each automaton uses a local clock variable clk. Both automata are synchronized with urgent synchronization channel channel.

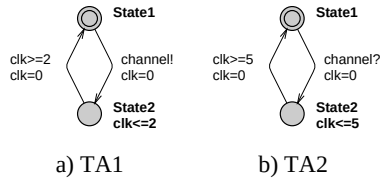


Figure 6: Timed automata example in UPPAAL like notation

The automaton *TA1* has two locations **State1** and **State2**. *TA1* waits for *TA2* in the initial location **State1** using channel. If the synchronization event occurs the clk is reset to zero and *TA1* passes into location **State2**. The automaton wait here for 2 time units. It is given by constrain in invariant and transition **State1** → **State2** guard. After than the clock clk is reset and *TA1* returns to **State1**.

In the similarly way, *TA2* starts in initial location **State1**. After the synchronization it waits in **State2** location for 5 time units and returns back to **State1** location.

The transformation from *TA1* into MCode is listed in figure 3.

The *TA2* transformation into *MCode* is similar to *TA1*. It differs only in code line 20 where (now_clk - clk) is compared to 5 sampling periods.

In order to retrieve a sequence of states leading to given state, a diagnostic trace can be implemented by using a circular buffer that stores a time stamp, state info and variable at each state turn.

2.1.3 Discrete-time system implementation

The continuous system model is implemented as a digital filter [15, 17]. The general form of the filter transfer function between the output $Y(z)$ and the input $X(z)$ is given by

$$G(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{1 + a_1 z^{-1} + \cdots + a_n z^{-n}} \tag{1}$$

where m is maximal order of numerator $y(z)$ and n is maximal order of denominator $x(z)$. Without loss of generality we suppose the system to be pure, $n \geq m$. With respect to the transfer function of z^{-1} that represents a delay of one time unit, the equation can be modified into

$$\begin{aligned} y(k) &= b_0 x(k) + b_1 x(k-1) + \cdots + b_m x(k-m) \\ &\quad -a_1 y(k-1) - \cdots - a_n y(k-n) \end{aligned} \tag{2}$$

The $y(k)$ is the k-th sample of output and $x(k)$ is k-th sample of input. It can be directly implemented into FPGA platform using delay elements, adders and multiplier.

For example, 3^{rd} order transfer function can be implemented by using XSG as shown in figure 7.

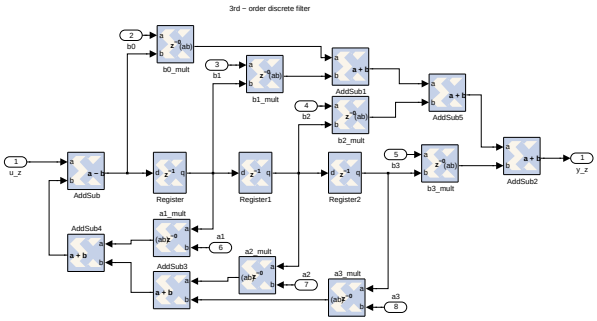


Figure 7: Transfer function implemented using XSG

An example of the transfer function response to input step function is depicted in figure 8. The figure shows the $y_s(k)$ of transfer function implemented in standard Matlab/Simulink, $y_x(k)$ of transfer function implemented in XSG toolbox and $y_f(k)$, the response of transfer function implemented in FPGA. The XSG $y_x(k)$ deviation is given by fix point logic accuracy, preset in toolbox arithmetic. The XSG simulation is bit-exact to the FPGA implementation, that means the simulated system behavior is identical to the system behavior finally implemented in target FPGA platform (signals $y_x(k)$ and $y_f(k)$ are completely identical in figure 7).

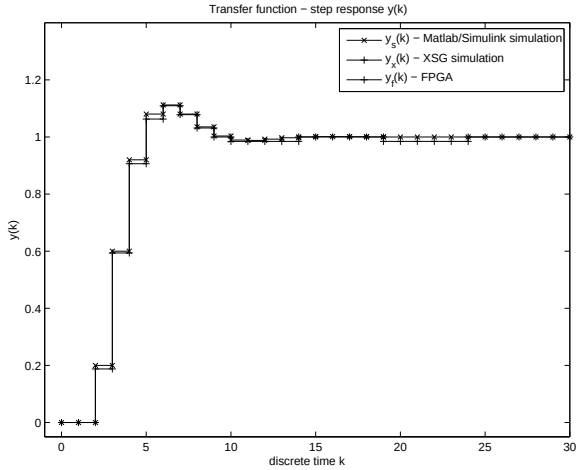


Figure 8: Output function of the system model as a reaction to step function

3 Case studies

The section shows two case studies. The first case is composed using the timed automata. The second case study combines both, timed automata and continuous system implementation.

3.1 Case study - Safety system

This case study deals with the on-line testing of controller that is responsible for safety system control used, for example, to avoid an operator to be injured at a danger device e.g. pressing or cutting machine. The case study is illustrated in figure 9. It consists of a *gate*, a *controller*, and a *door*. The *gate* detects an object that is approaching the press device e.g. an operator body part. The *door* prevents the object to reach the device. The *controller* responsibility is to close the *door* in specified time after the *gate* was activated.

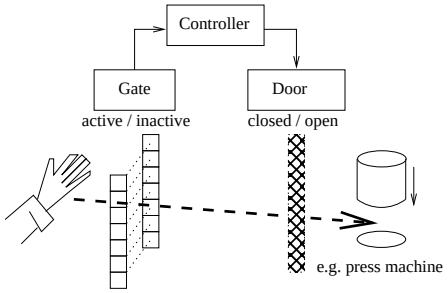


Figure 9: Case study 1: Safety system

Very simple *controller* algorithm, the *gate* functionality and the *door* functionality are presented in figure 12.

The goal is to validate the controller whenever it closes the *door* within specified time after the *gate* is activated.

The case study implementation scheme of HIL is presented in figure 10. It consists of a system model simulating the *gate* and *door* activity, the *controller* under test and tester containing three property blocks and the test case presented bellow.

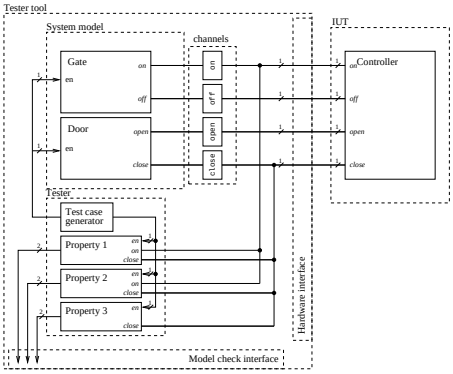


Figure 10: Case study 1: Implementation scheme of the case study HIL

The system model is composed of timed automata representing the *gate* and the *door* (see figure 12a) and c)). Both the *gate* and the *door* timed automata are transformed into XSG using methodology depicted in subsection 2.1 including synchronization channels on, off, open and close.

The tester consists of the test case generator and the set of properties to be verified. The test generator activates the system model block and all property blocks as presented bellow. The tester and the properties are given by timed automata with sampling period 100 *nsec*.

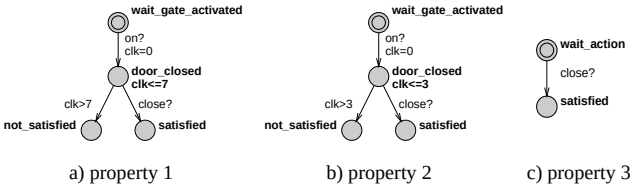
There is just one test case implemented for this case study. The test case activates the system model *gate* singly at 2 μsec. All property blocks are activated at the time.

There are three properties to be verified. Each property is expressed by Computation Tree Logic (CTL) [10] and transformed into timed automata model as explained in subsection 2.1. The list of properties is the following:

Property 1 *"The controller always closes the door within 7 μsec since the moment the gate is activated."*. The accordant CTL formula is $A\Box\ t\_clk <= 7$. For the property timed automaton see figure 11a).

Property 2 *"The controller always closes the door within 3 μsec since the moment the gate is activated."*. CTL formula is $A\Box\ t\_clk <= 3$. For the property timed automaton see figure 11b).

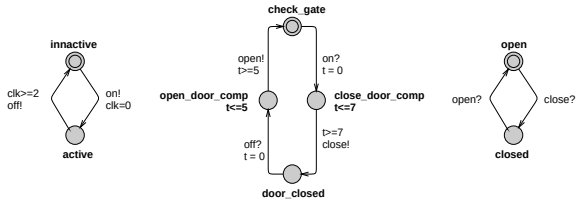
Property 3 *"Does there exist the state in which the door is closed?"*. CTL formula is $E\Diamond$ *door_closed*. The property timed automaton is depicted in figure 11c).



a) property 1 b) property 2 c) property 3

Figure 11: Case study 1: Properties to be verified

The *controller* under test is implemented using timed automata approach (controller timed automata now_clk update time = 1 μsec) . Its structure is depicted in figure 12b). When the *gate* is activated, the *controller* receives the synchronization signal on (consequently, transition **check_gate** → **close_door_comp** in figure 12b) is activated). Further then it computes a reaction and it closes the *door* by signal close within 7 time units (**close_door_comp** → **door_closed**). When the *controller* receives information that the *gate* is deactivated, it opens the *door* with 5 time units execution delay (**door_closed** → **open_door_comp** → **check_gate**).



a) Gate TA b) Controller TA c) Door TA

Figure 12: Case study 1: Gate, controller and door timed automata

Figure 13 shows the state evolution of the *controller* and the *gate* in upper plot and evolution of each verified property in bottom one.

Given by the studies, the *gate* is activated at 2 μsec. The *controller* evolves at the time in **check_gate** → **close_door_comp** and switches to the **door_closed** after 7 μsec (computation time of the controller). Figure 13 bottom plot shows the following results

Property 1 is satisfied. The door is closed within 7 μsec

Property 2 is not satisfied. The door is not closed within 3 μsec.

Property 3 is satisfied. The **door_closed** state occurs at 9 μsec

3.2 Case study 2 - Motor control

This case study deals with testing of controllers given by continuous system representation. It presents combination of continuous system (system model, controller) with discrete event system (property blocks, test generator). The study tests several properties of the servo-system with or without friction to shaft. The system consists of a motor and a controller. The controller operate the motor to a reference angular velocity ω_ref.

The motor behavior can be described by the following equations

$$\begin{aligned} \frac{\partial i}{\partial t} &= -\frac{R}{L}i - \frac{\xi}{L}\omega + \frac{1}{L}u \\ \frac{\partial \omega}{\partial t} &= \frac{\xi}{J}i - \frac{1}{J}m_z \end{aligned} \tag{3}$$

In this system, R is an terminal resistance, L is an terminal inductance, ξ is a torque constant, ω is an angular velocity, u is an applied armature voltage, i is an armature current, J is an equivalent moment of inertia of the motor and load referred to the motor and m_z is a torque load given by e.g. friction to shaft.

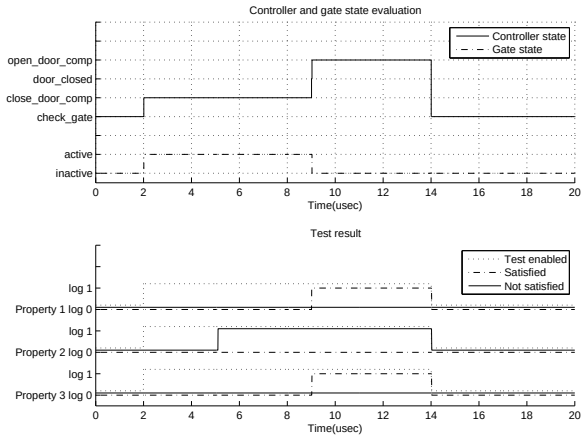


Figure 13: Case study 1: Tester output

Using the motor Maxon A-max 26 [2], the transfer function $\frac{\omega}{u}$, with $m_z = 0$ and $T_s = 1[msec]$, is

$$G(z) = \frac{2.652z + 0.3143}{z^2 - 0.9202z + 0.0001003} \tag{4}$$

When the servo-system has friction to the shaft $m_z = 0.1 * 10^{-3}$[Nm] the transfer function, with $T_s = 1[msec]$, is

$$G_{inh}(z) = \frac{2.559z + 0.2924}{z^2 - 0.8483z + 9.249 * 10^{-5}} \tag{5}$$

There are two controller tested in this study separately. The first controller is given by transfer function

$$G_{C1}(z) = \frac{0.3136}{z + 0.5151} \tag{6}$$

with same T_s. The transfer function of the second controller is

$$G_{C2}(z) = \frac{0.3377z - 0.3056}{z - 1} \tag{7}$$

The goal is to test each controller to be able to operate both systems given by transfer function 4 and 5. Each controller has to be able to satisfy bellow mentioned properties in both operating conditions, with and without friction at the shaft (given by). The properties are the following:

1. The motor angular velocity should be stabilized into interval (9, 11) $Rads^{-1}$within 5 $msec$ when operating condition are changed (i.e. when reference velocity is changed from speed 0 $Rads^{-1}$to a non zero velocity or when friction m_z varies).

2. Along the state of the system the motor angular velocity should not exceed 12 $Rads^{-1}$.

The structure of the HIL is depicted on figure 14. All the tester tool parts except the interfaces have been implemented using XSG. The speed of the continuous part is degraded by the performance of ADC and DAC. The problem with DAC can be partially solved while implementing Pulse Width Modulation (PWM) in FPGA for a specific controlled system (e.g. in the servo-motor case study, the motor behaves like low pass filter with the time constant which determines the period of PWM). The problem with ADC, in the servo-motor case study, can be solved by implementation of the IRC speed detector in FPGA.

The controllers are implemented using Matlab/Simulink discrete transfer function blocks. They are directly connected into the tester tool via XSG in/out interfaces.

The tester tool consists of the parts depicted in chapter 2. All the necessary details of each part are given in the following paragraphs.

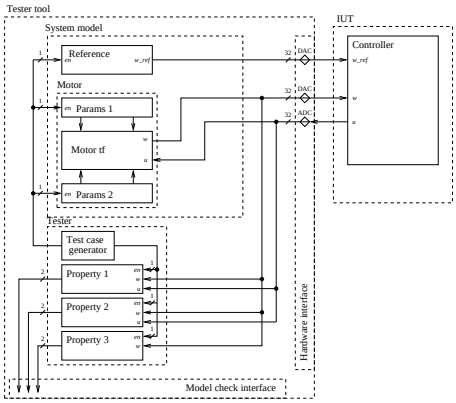


Figure 14: Case study 2 - Tester structure

The *System model* includes *Reference* block and *Motor* block. The *Reference* block sets up a required motor angular velocity ω_ref. The *Motor* block influences the tested controller. It is given as a system with time varying parameters. During the test the parameters change between transfer function 4 and 5. The *Motor* transfer function sampling period is setup to 1 μsec (using XSG up sampling and down sampling blocks).

The *Tester* consists of a *Test case generator* and three *Property* blocks to be verified.

The *Test case generator* simulates a situation when the *motor* is started from 0 $Rads^{-1}$to ω_ref and at 20 $msec$ the motor brakes are applied so the friction at motor shaft is nonzero. The motor behavior within the first interval $(0; 20)$ $msec$ is given by equation 4, the behavior in interval $[20; inf)$ $msec$ is given by equation 5.

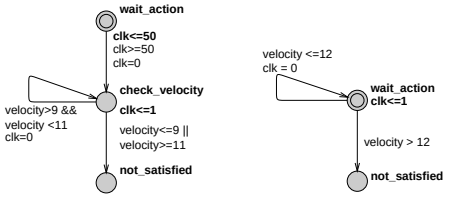
For that purpose, the *Test case generator* simultaneously enables *reference* block to propagate the w to the tested controller and enables *Params 1*. It activates *Property 1* and *Property 3* blocks as well. At 20 $msec$ *Test case generator* switches the *motor* parameters from *Params 1* to *Params 2*, consequently the *Property 2* block is activated.

The properties to be verified are derived from the case study requirements. Each property is implemented using timed automata as presented in section 2.1. The *Property* block sampling period is 100 μsec. Description of each preperty, it's according CTL formulae and equivalent timed automata are the following:

Property 1 *"The motor angular velocity is always stabilized into interval (9, 11) $Rads^{-1}$within 5 msec."*. The CTL formula is $A\Box\ (clk >= 50) \rightarrow (velocity > 9 \quad and \quad velocity < 11)$ and figure 15a) shows the property automaton.

Property 2 *"The motor angular velocity is always stabilized into interval (9, 11) $Rads^{-1}$within 5 msec."*. The CTL formula and property timed automaton are similar to property 1.

Property 3 *"The motor angular velocity does not exceed 12 $Rads^{-1}$."*. The CTL formula is $A\Box\ (speed \leq 12)$ and automaton is in figure 15b).



a) Property 1 and 2 b) Property 3

Figure 15: Properties timed automata

The controllers under test are given by transfer functions $G_{C1}(z)$ and $G_{C2}(z)$ shown in equation 6 and 7 respectively.

The test results for controller 1 are depicted in figure 16. The upper figure plot shows the motor input voltage u given by controller operation, motor angular velocity output ω and reference velocity ω_ref. The bottom plot depicts *model check interface* output.

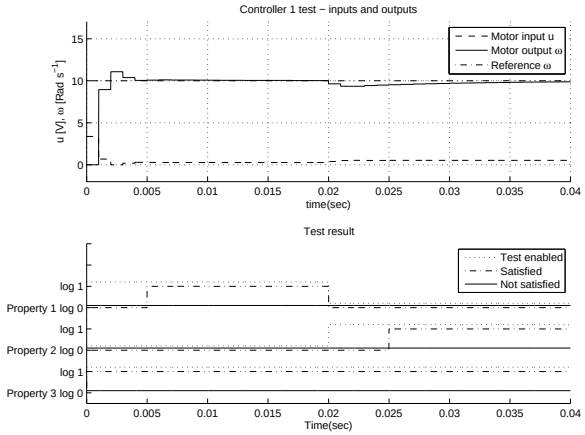


Figure 16: HIL inputs and outputs for controller 1

The required property verification results are the following

Property 1 is satisfied. As presented in figure 16, the controller controls the velocity to the required reference within 4 $msec$. The test of property 1, enabled within time interval $[0, 20)$ $msec$, shows the property is satisfied i.e. the location **not_satisfied** in figure 15 is not reached.

Property 2 is satisfied. Although the motor angular velocity is influenced by friction to the shaft at 20 $msec$ the controller is able to keep the velocity at required value. The test of property 2, enabled within time interval $[20, 40)$ $msec$ shows the property is satisfied.

Property 3 is satisfied. During the test interval velocity value does not exceed 12 $Rads^{-1}$, the property is satisfied in the whole tested interval $[0, 40)$ $msec$.

The test results for the controller 2 are presented in figure 17. The property results are the following:

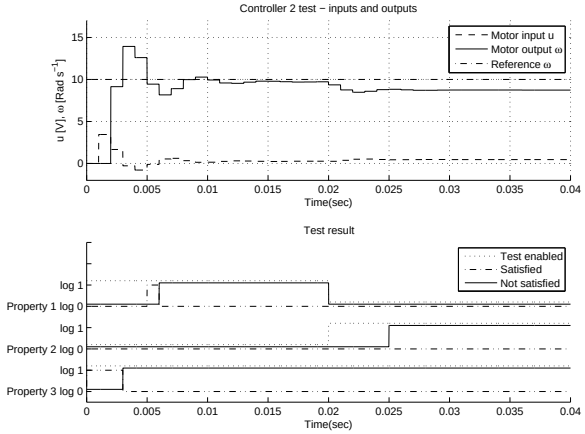


Figure 17: HIL input and outputs for controller 2

Property 1 is not satisfied. As presented in the picture, the controller is not able to stabilize the velocity as fast as the previous one. The test of property 1, enabled within time interval $[0, 20)$ $msec$, shows the property is satisfied in the first sample but after the velocity falls out the required value. Finally the property is not satisfied.

Property 2 is not satisfied. The controller is not able to operate the velocity in the required interval. The test of property 2, enabled within time interval $[20, 40)$ $msec$ shows the property is not satisfied because the value is bellow the required value interval $(9; 11)$ $Rads^{-1}$.

Property 3 is not satisfied. Velocity oversteps the 12 $Rads^{-1}$boundary at 3 $msec$.

4 Conclusion

The tester tool shown in this article enables to prove a quality of developed controllers without an assembly to the final product. Testing of the considered class of applications requires high sampling frequency, low jitter and scalability. Therefore we have chosen FPGA platform, the synchronous logic hardware capable to achieve high degree of parallelism.

Due to the hybrid nature of industrial applications, the presented methodology combines discrete event systems and continuous systems using timed automata and transfer function representation.

The sampling period of the tester tool is given by the length of the FPGA clock cycle and by the number of the FPGA clock cycles needed to evaluate the continuous part and discrete part. The shortest possible FPGA clock cycle is about 10 *nsec*, assuming the continuous system of the third order and 16-bit fixed point logics on Xilinx Virtex II. The existence of the urgent locations and channels in the timed automaton requires repetitive calls of the corresponding timed automaton function, which gives the number of the FPGA clock cycles needed to evaluate the state of timed automaton. In normal applications (free of timed automata with long sequences of urgent locations) one can easily approach the sampling period of 100 *nsec*, which is a very good speed performance. Moreover using FPGA, the synchronous logics, the jitter is less than one clock cycle. The scalability of our tester tool is given by the size of the FPGA only. Due to the physical parallelism, the blocks do not influence each other.

In the future work we will focus on the interfaces, namely improvement of ADC and DAC. Further work will deal with a dynamic reconfiguration of FPGA that can be considered for on-line generation of the test cases and automation of testing process.

References

[1] LabVIEW FPGA in hardware-in-the-loop simulation application, 2003. National Instruments corp.

[2] A-max 26, graphite brushes, 11 watt servomotor data-sheet, April 2005. Maxon ltd.

[3] Project OCERA - Open Components for Embedded Real-time Applications, 2005.

[4] RTLinux, 2005. FsmLabs ltd.

[5] Xilinx system generator v7.1 user guide, 2005. Xilinx ltd.

[6] CarMaker, 2006. IPG Automotive GmbH.

[7] Karine Altisen and Stavros Tripakis. Implementation of timed automata: An issue of semantics or modeling? In *FORMATS 2005*, pages 273–288, 2005.

[8] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[9] Gerd Behrmann. Uppaal Timed Automata Parser - UTAP, 2005.

[10] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on uppaal, formal methods for the design of real-time systems. In *SFM-RT 2004*, pages 200–236. Springer-Verlag, 2004.

[11] M. Bozga, J.C. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, and L. Mounier. IF: a validation environment for timed asynchronous systems. In E.A. Emerson and A.P. Sistla, editors, *CAV2000 proc.*, volume 1855 of LNCS, pages 543–547. Springer Verlag, 2000.

[12] E. A. Emerson. Temporal and modal logic, 1990. Handbook of Theoretical Computer Science.

[13] Moez Krichen and Stavros Tripakis. Black-box conformance testing - for real-time systems for real-time systems. Technical report, VERIMAG, www-verimag.imag.fr, 2004.

[14] K. G. Larsen, M. Mikucionis, B. Nielsen, and A. Skou. Testing real-time embedded software using UPPAAL-TRON: an industrial case study. In *EMSOFT '05*, pages 299–306, Jersey City, NJ, USA, September 18 - 22 2005. ACM Press New York, NY.

[15] M. D. Lutovac, D. V. Tosic, and B. L. Evans. *Filter Design for Signal Processing using MATLAB and Mathematica*. ISBN 0-201-36130-2. Prentice Hall, 2000.

[16] M. Mikucionis, K. Larsen, and B. Nielsen. Online on-the-fly testing of real-time systems. Technical Report RS-03-49, (BRICS), 2003.

[17] Katsuhiko Ogata. *Discrete-time control systems*. ISBN 0-13-216227-X. University of Minnesota, 1987.

[18] Katsuhiko Ogata. *Modern Control Engineering*. ISBN 0-13-598731-8. University of Minnesota, 2nd edition edition, 1990.

[19] H. Schludermann, T. Kirchmair, and M. Vorderwinkler. Soft-Commissioning: Hardware-in-the-loop-based verification of controller software. In PROFACTOR GmbH, editor, *Proceeding of the 2000 Winter Simulation Conference*, 2000.

[20] Sudhakar Yalamanchili. *VHDL Starter's Guide*. Prentice Hall, 1998. 269 pages.