Hardware-In-the-Loop simulator for turboprop and turboshaft engine control units

J. Vejlupek, M. Jasanský, V. Lamberský, R. Grepl

Abstract— This paper presents the development and implementation of the Hardware-In-the-Loop (HIL) Simulator for turbo-prop and turbo-shaft engine control units (ECUs) on a low-cost embedded microcontroller. Developed HIL Simulator is a subsystem of a complex test device TPR_CPSP_SIM designed for use in the development and also in the manufacturing process of the ECUs.

In this document, we describe the development process of the part of the HIL simulator, which runs the engine simulation model and provides selected signals for the tested control unit. Main goal for this project was to implement turbo-prop and turbo-shaft engine (gas turbines) TP100 and TS100 models into the microcontroller and set-up the peripherals for the interaction with the rest of the system to obtain reliable HIL simulation platform.

Keywords—Hardware-In-the-Loop Simulation, Engine Control Unit, ECU, CAN Aerospace, Rapid Code Generation.

I. Introduction

ONE of the key steps in a modern product development is a product testing stage. HIL simulation is often used for complex tests of electronical control units (ECUs), sometimes also coupled with power electronics (Power-HIL), placing additional requirements on the HIL simulation. HIL Simulation techniques are widely used specially in automotive and aerospace industry [1], [2], [3]. Various hardware and software platforms specially designed for HIL simulations are commercially available, from low cost [4] to very expensive devices [5], [6]. For this project we have selected custom build hardware, as there were many specific requirements for the signal conditioning. Software was created using Rapid Control Prototyping [7] and Rapid Code Generation [8] techniques in Matlab/Simulink.

Test stand TPR_CPSP_SIM is designed for HIL testing of the ECUs for turbo-prop and turbo-shaft engines (gas turbines) TP100 and TS100. Part of this test stand is the EVA_PIC32 module, which task is to run the engine simulation and provide respective signal processing. Either the signals are generated based on the operator request via the software running on external PC, or based on the engine model running on the microcontroller which is the core part of the EVA_PIC32 module. A 32-bit PIC microcontroller implements the model (both engines TP100 and TS100), and handles the IO signals for the ECU and also the rest of the test stand. Chapter SYSTEM DESCRIPTION AND REQUIREMENTS provides more detailed description of the whole HIL simulator system, and summarizes the requirements for the EVA_PIC32 subsystem. In chapter SW IMPLEMENTATION we describe the implementation of the engine model, and how the peripherals are handled.

This work was supported by the European Commission within the FP7 project Efficient Systems and Propulsion for Small Aircraft "ESPOSA", contract No. ACP1-GA- 2011-284859-ESPOSA.

J. Vejlupek is a PhD candidate at the Brno University of Technology – Faculty of Mechanical Engineering. His current research focuses on Hardware in the Loop simulation, Rapid Control Prototyping and Rapid Code Generation for embedded applications. (e-mail: vejlupek@fme.vutbr.cz)

M. Jasanský is with the UNIS, a.s. company, Division of Aerospace and Advanced Control. (e-mail: mjasansky@unis.cz)

V. Lamberský is a PhD candidate at the Brno University of Technology – Faculty of Mechanical Engineering. His current research focuses on Rapid Code Generation for embedded applications.

R. Grepl is a Associate Professor at the Brno University of Technology – Faculty of Mechanical Engineering.

II. System Description and Requirements

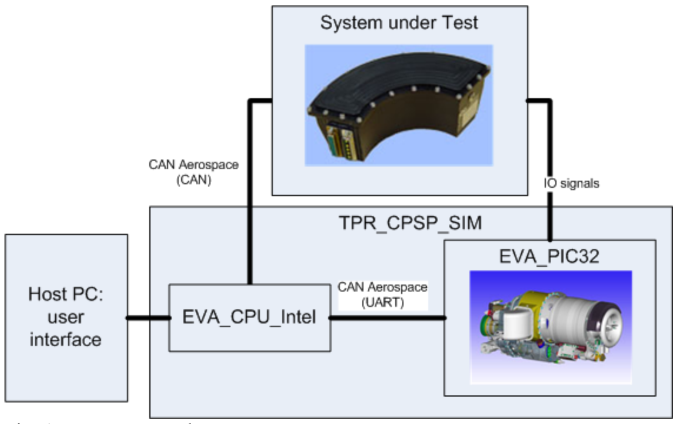

Fig. 1: HIL system scheme

Fig. 1 shows the scheme of the HIL system setup, individual components and related items are described below:

A. *System under test: Engine Control unit (UNIS)*

The ECU is an electronic device intended for control of TP100 / TS100 gas turbine engines (manufactured by PBS Velká Bíteš a.s., Czech Republic). The ECU also provides +28 V DC power supply for other onboard devices. There are two types of ECU: one for turbo-prop engine, and one for turbo-shaft engine. They are both very similar in design.

Main functions of the ECU are the following:

- regulation of turbine rational speed to desirable value (0 to 60000 rpm)
- regulation of the free turbine speed
- turbine start and stop regulation

- onboard electric supply
- turbine operational parameters monitoring and checking

B. *TP100 / TS100 engines (PBS)*

TP100 is turboprop engine designed for use in ultralight airplanes (piloted as well as UAVs) and TS100 is a turboshaft engine intended for use in ultralight helicopters. Both engines are based on the TJ100 turbojet engine. In a simplified way, the TJ100 works as the gas generator driving the free turbine. The main mechanical difference between TP and TS variant is the gearbox, where the TP100 nominal speed is 2158 rpm, and TS100 has nominal speed of 6000 rpm.

C. *TPR_CPSP_SIM*

Test stand TPR_CPSP_SIM is a HIL simulator designed for manual and automatic HIL tests of the ECUs. Tests are managed by the user from an application (created by UNIS) running on external PC.

TPR_CPSP_SIM enables two modes of operation: *Tester* and *Simulator*. The *Tester* mode is designed to inspect individual functions of the system, such as analogue and digital read-outs, communication, software functionality, etc. The *Simulator* mode is using the TP100 / TS100 engine model for the HIL test, in this mode, the ECU is running as it would be with a real engine in the aircraft. Mode selection is done using the switch on the TPR_CPSP_SIM front panel.

Key components of the TPR_CPSP_SIM are:

- Central control unit EVA_CPU_Intel, which provides communication interface between EVA_PIC32 (UART), ECU (CAN), and PC (RS232).
- EVA_PIC32, module described by this document, implements the TP100 / TS100 engine model and part of the signal processing for the interaction with the System-Under-Test (SUT) (ECU)
- BLDC Electronic Speed Controller (ESC) and BLDC motor M5 representing the jet engine. M5 is mechanically connected with M3 – BLDC motor used as starter-generator, driven by ECU.
- Fuel pumps M1 and M2, connected together with BLDC motor M4 creating the load for both fuel pumps by being permanently loaded with resistors. M4 is also connected to EVA_PIC32 for speed sensing.
- Relay and I/O boards used for fault condition simulation by disrupting selected signals.

D. *EVA_PIC32 engine simulator module*

EVA_PIC32 module is one of the key components of the TPR_CPSP_SIM HIL simulator. Its purpose in the system is to provide signals generated by the TP100 / TS100 engine for the ECU. These signals are, depending on the mode of operation (*Tester* / *Simulator*), generated either by the user through an external application, or by the engine model running on the EVA_PIC32 microcontroller.

Key functions of the EVA_PIC32 are:

- Communication with the EVA_CPU_Intel via UART using CAN Aerospace protocol.
- Generate signal representing the speed of the free turbine (doubled signal: nVTa, nVTb)
- Drive the ESC for the BLDC motor M5 representing the jet engine.
- Check the speed of the fuel pump through the M4 motor.
- In the *Simulator* mode, run the TP100 / TS100 engine model and handle respective signals accordingly.

Tester / Simulator switch is directly connected to the EVA_PIC32 as a digital input. Next, there is a STOP button, which when pushed resets the module to default values, most importantly stops all motors.

Both speed measuring inputs are in form of frequency: pulses with 50% duty cycle are connected to the Input Capture peripheral. Frequency is directly proportional to the speed. Signal generated for the ESC is standard RC signal: 100-400Hz and pulse width between 1ms and 2ms.

Communication with the rest of the system is realized through the UART peripheral, and the protocol used is the CAN Aerospace, data are transmitted using the HEX representation.

III. SW Implementation

Software for the 32-bit PIC microcontroller is created using the Rapid-Control-Prototyping and Rapid-Code-Generation techniques enabled by MATLAB-Simulink Embedded Coder together with Kerhuel Toolbox. This set of tools allows us to generate the C code (including microcontroller setup and peripheral handling) for the microcontroller directly from MATLAB Simulink. This is very convenient way especially for the development phase, when the model of the engine is being often modified during the development phase. Nevertheless, some functionality had to be coded manually in C code, as the Kerhuel Toolbox does not enable all the functionality needed.

This section describes the requirements in more technical details and explains how they have been implemented and fulfilled.

A. *System settings*

The core of the EVA_PIC32 board is a 32-bit PIC microcontroller pic32mx320f128h clocked with external 10MHz crystal and scaled up to 80MIPS. Main execution loop time step is set to 0.01s (100Hz), all the functions (if enabled) are executed at this rate. Actual functionality is determined by the state-machine implemented using Stateflow (MATLAB Simulink Blockset). State machine is in detail described bellow in section D. Overall system performance is summarized in section E.

B. *CAN Aerospace protocol*

CAN Aerospace protocol [9] is data format definition designed for airborne applications using microcontrollers with CAN peripheral. However in this case, ACAN protocol is used over the UART simply by taking the data part of the CAN message.

To have the complete message, the information about the length of the data (0-8), and the CAN ID is added before the actual message. As the CAN ID could range from 0 to 2031 (11-bit identifier), it has to be represented by two bytes: CAN_ID High, and CAN_ID Low. Furthermore the CAN_ID is aligned by bit-shift by 5 bits to the left, before it is split. This is due to convention used by the third-party UART-to-CAN converter which is used inside the TPR_CPSP_SIM simulator.

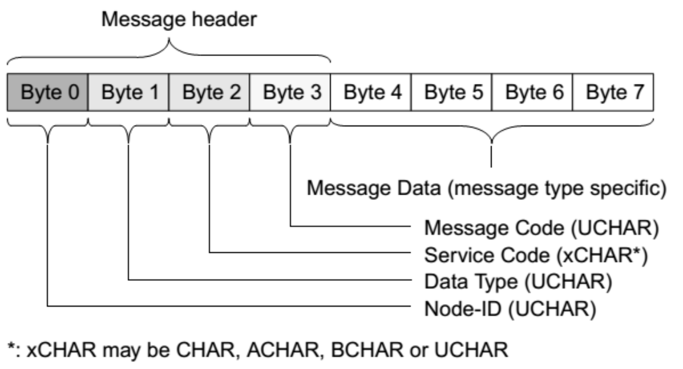

Fig. 2 ACAN General message format [1]

As an example we show how the "*Low priority Node Service data*" (NSL) message is transmitted over UART: NSL message type is defined over the CAN-ID range of 2000-2031. UNIS uses this messages for read and write operations on control registers. Where the D1 – Node ID specifies the target control unit, D2 – Data Type specifies the data type of the data in the message (for read register is D2 = 0: "No data"), D3 – Service Code determines the type of operation, where 102 means RCRS – *Read Configuration Register Service*, and 103 means SCRS – *Set Configuration Register Service*. These messages are defined internally by UNIS. D4 – Message Code then identifies the register. In Table I we show how the ACAN message asking node with ID 211 to send value of register 150 (RCRS) is transmitted over UART.

Data are transmitted using the HEX representation: Byte (0-255) is taken in the hexadecimal notation (0x00-0xFF), split into two characters, and their representation is sent in Byte representation via the UART. This method enables simple way to delimit the message frames. Data frame is terminated with *carriage return* character.

TABLE I. SENDING RCRS FOR REGISTER 150 TO NODE 211

	Data length		CAN_ID = 2000				D1		D2		D3		D4		termination character
			CAN_ID High		CAN_ID Low		Node ID		Data Type		Service Code		Message Code		
DEC	4		250		0		211		0		102		150		
HEX	04		FA		00		D3		00		66		96		
UART	48	52	70	65	48	48	68	51	48	48	54	54	57	54	13

Implementation of the receiver is done in two while loops: inner loop reads the UART buffer until it founds the termination character, or until the buffer is empty. If the termination character is found, then the complete formatted message is sent for processing. Outer loop checks only if the buffer is empty. If the buffer is empty, the receiver loop ends until the next time step iteration.

Messages transmitted from the EVA_PIC32 are basically of two kinds: response to received message, which are handled directly by the received message processing function, and data requests based upon the actual mode of operation by individual function blocks – inside enabled subsystems.

C. *IO Signals*

This section discusses individual digital inputs and outputs aside from UART communication, which has been described in the previous section.

1) *STOP button and Tester/Simulator switch*

To ensure the safety of the operator, tested control unit, and test-stand itself, STOP button has to be present on the TPR_CPSP_SIM tester. It is placed on the front panel of the device. This button is connected to the EVA_PIC32 board, and when pushed, all the driving signals need to be reset to their default values. This applies above all to the signal driving the ESC controller for the BLDC motor M4, and the free turbine speed signal.

Switching between *Tester* and *Simulator* modes is done by the Tester / Simulator switch, which is also on the TPR_CPSP_SIM tester front panel. If the state of the switch changes the state machine controlling the mode of operation goes through the stop procedure and switches the mode. Further description is below in the section D.

2) *Driving M5 with ESC*

Motor M5 represents the jet engine, and is connected to M3, which represents the starter-generator. M3 motor works in the motor mode (driven by the ECU) while the Tx100 engine is starting, after the jet engine goes to stable run mode, M3 is used in the generator mode, producing the power for the onboard electronics (through the ECU).

Motor M5 is a BLDC motor, and it is driven by the customized ESC with fast RC signal – control frequency is up to 400 Hz, with the standard pulse width between 1ms and 2ms. Signal for the ESC is generated by the PIC microcontroller using the Output Compare peripheral. As the pulse width ranges from 1ms (zero speed) to 2ms (full speed), we have decided that the 8-bit resolution would be sufficient. So the pulse width between 1 and 2 ms was divided into 256 values, with 0 representing 1ms, and 255 representing 2ms. Timer 3 is used as a time base for the OC3 providing the PWM signal. Timer 3 is scaled from the main 80 MHz clock by the factor of 64 down to 1.25 MHz, and the period register is set to 12499. This sets the Timer 3 for 100Hz period. OC3 is configured for the PWM mode with fault pin disabled (OCM = 0b110), and the secondary compare register is by default (and by the reset) set to OC3RS = 1250, generating the 1ms width pulses. Function which sets the secondary compare register has lower saturation set to 1250 (1ms) and upper saturation set to 2500 (2ms), ensuring the valid signal for the ESC. OC3 is enabled at the TPR_CPSP_SIM tester power-up and unless the value is changed by the operator (in tester mode), or by the running engine model (in simulator mode), it generates the 1ms pulses, so the M5 remains still.

3) Free turbine signal generation

Signals nVTA and nVTB are doubled signal representing the output of the free turbine speed sensor. This signal was defined as square wave signal, with frequency ranging from 60 to 6000Hz, with step of 1Hz, and maximal error of 0.08%.

Both signals are identical, to generate nVTA OC1 is used, and for nVTB OC2 is used. We will describe how these signals are generated on the nVTA, as these methods apply also for the nVTB.

Since the signal is defined as square wave signal, output compare peripheral is used in the PWM mode with fault pin disabled (OCM = 0b110). In this case, the duty cycle is 50% and the period (frequency) is variable. To guarantee the maximal error of 0.08% and required frequency range, we had to implement timer prescaller switching. As a clock source for the Output Compare, Timer 2 is used, and the prescaller is set to 1, 8, or 64, depending on the frequency generated. Fig. 3 shows achieved accuracy for the generated frequency.

Maximum theoretical error is calculated from period T corresponding to each frequency f, and prescaller resolution t_{ps} ($0.0125\mu s$ for 1x prescaler, $0.1\mu s$ (for x8 prescaler), $0.8\mu s$ for x64 prescaler).

$$e_f = t_{ps} / T_f / 100 \quad (1)$$

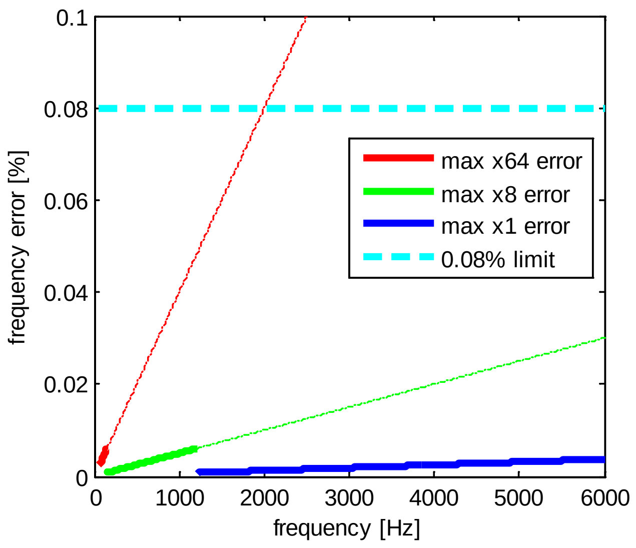

Fig. 3 Generated frequency - achieved accuracy

Reason for the prescaller switching is the wide frequency range and low error tolerance: to be able to generate as low frequency as 60Hz, prescaller with factor of 64 has to be used:

With 16-bit timer[1] the prescaller factor of 1 allows generating frequencies down to 1220Hz, prescaller factor of 8 allows frequencies down to 152Hz and prescaller factor of 64 allows frequencies down to 19 Hz. However we cannot use the prescaller factor of 64 as the low error requirement is not achievable with it. To illustrate this problem, we will calculate the period register for the frequency of 6000Hz using the prescaller factor of 64:

$$\left(\left(80,000,000 / 64\right) / 6,000\right) = 208.\overline{3} \qquad (1)$$

Since the Output Compare peripheral works only with integer numbers, we have to round the value to the closest integer (208) to calculate the frequency actually generated. In this case, we will get:

$$\left(\left(80,000,000 / 64\right) / 208\right) = 6009.615 Hz \qquad (2)$$

Since the maximum allowed error is 0.08%, which at 6000Hz is 4.8Hz and at 60Hz is 0.048 Hz, we can see that we need better resolution – lower prescaller factor.

To solve this problem, timer prescaller switching was implemented in a following way:

- Prescaller factor 1: freq. from 1221Hz up to 6000Hz
- Prescaller factor 8: freq. from 153Hz up to 1220Hz
- Prescaller factor 64: freq. from 60Hz up to 152Hz

This implementation allows us to keep the error of nVTA and nVTB signals bellow 0.015%.

Default value for nVTA and nVTB is 0Hz – logic level on OC1 and OC2 pins is held low, and Output Compare is off. First valid setting of the free turbine speed register turns the Output Compare peripherals on and starts the pulses. If the STOP button is pressed, both pins are driven low and respective Output Compare channels are disabled.

4) Starter-generator speed measurement

Measurement is done using the Input Capture peripheral which reads the square signal from the motor (after voltage level conversion). IC4 is used for the measurement. Measured frequency range is from 500Hz up to 4000Hz, which corresponds to the speed range of 7500rpm…60000rpm ($n = f \cdot 15$).

Speed of the starter-generator motor is also used in the control loop for driving the M5. As the ESC does not provide any speed feedback, or characteristics about the result speed, speed of the M5 is assumed to be the same as the speed of the starter-generator M3, since they are connected together mechanically.

5) Fuel pump speed measurement

Fuel pump speed is determined in a very similar way as the speed of the starter-generator described in section *4)*. For the signal measurement IC1 is used (also configured with Timer3 as clock source). Frequency measured is assumed in range of 150Hz…750Hz, corresponding to speed range of 3000rpm…15000rpm ($n = f \cdot 20$). Speed of the fuel pump determines the output of the jet engine, and it is the only input to the TP100 / TS100 simulator model.

D. System modes of operation

As mentioned before, there are two modes of operation: *tester* mode, and *simulator* mode. Beside this, some other events need to be considered, such as STOP button and

[1]32-bit timer is not available, due to the fact that for OC and IC peripherals are available only two 16-bit timers: T2 and T3. These would be both consumed to form one 32-bit timer, but T3 is used for OC3, which provides the control signal for ESC.

switching between the tester / simulator mode. To implement these features state machine programmed in Stateflow (MATLAB Simulink Blockset) is used. Inputs for the Stateflow besides control registers are functions reading the digital inputs: tester/simulator switch, and the STOP button. State machine determines the actual mode of operation and handles the enabling of the correct subsystems.

Both modes of operation are described in the sections below. However there are two significant events that need to be checked in parallel:

- STOP button pressed:
  - sets the ESC signal to default value (1ms pulse)
  - sets the free turbine signals to default value (zero)
  - stops the operation of all tester / simulator subsystems
- Tester/Simulator switch is toggled:
  - Performs the STOP sequence described above.

1) Tester mode

Tester mode is intended for calibration and production testing of the ECUs. It is fully manual mode – all actions have to be issued by an operator via external interface. Main functions of the tester mode are:

- measure the speed of fuel pump *nFPump*
- measure the speed of the starter-generator *nOut*
- drive motor M5 via the ESC *pwmM5*
- simulate the free turbine speed signals *nVTA*, *nVTB*

From the software view of the state machine: Transition from the default case is made in case that the STOP button is released and the switch is in the Tester position. First, the tester initialization is executed: this ensures the reset of all control registers, and that the simulator subsystems are disabled. In the tester mode, only incoming ACAN messages are handled, there are no messages generated by the microcontroller (aside from responses to the incoming commands).

2) Simulator mode

Simulator mode is intended for the semi-automated and for the automated HIL tests. In this mode, the engine model (TP100 / TS100) is run based on the input commands and provides feedback to the tested ECU. Engine model is selected at the end of the simulator initialization sequence. Selection is based upon the state of the register R120 (set by the user) designated for this purpose. Initialization sequence enables the ACAN communication, then the model waits until all conditions are met:

- Starter-generator speed is over 6000rpm
- Fuel pump speed is over 3000rpm
- Ignition is on
- Fuel valve is open
- Output temperature is higher than Tcomb

If all conditions are met, state machine decides which model (TP100 / TS100) to use, based upon the value in R120 and switches to the "rev-up" mode. The "rev-up" mode simulates the start-up and ignition of the engine, after that the transition to the state "simulator_run_Tx100" is executed. This state enables the respective model and in case of TS100 also ACAN request for the R17, which stores the settings of the collective.

Model runs continuously as long as the fuel valve is open, or until the STOP button is pressed, or the mode of operation is changed (*tester* / *simulator* modes). When the model execution is stopped, the model is disabled and all the outputs are set to default values. After that the state machine switches to the default state and waits for the next run.

E. System performance

To measure the system performance in the terms of the microcontroller load, "busy flag" method is used: At the beginning of each computational step selected digital output pin is set high, and after all the functions have been handled, it is set low. This will give us 100Hz square wave signal (with 0.01s time step), where the duty cycle represents the actual work load of the microcontroller. However, this method does not consider all the interrupt routines, but they are considered to have a minor effect on the result. More significant than the interrupt routines is the serial communication.

Measured workloads:

• Tester mode:	8 %
• TS100 simulator mode:	30 %
• TP100 simulator mode:	28 %

Code was compiled using Microchip C30 compiler; following are the memory usage statistics:

• RAM used:	6254 bytes	(38%)
• Flash used:	70500 bytes	(49%)

IV. Conclusion

Software for the EVA_PIC32 board described in this paper was created as a cooperative project between University of technology Brno and UNIS company. Results are used by UNIS and their customer (PBS) during the unit manufacturing and assembly process of the ECUs and the Tx100 engines.

Presented work describes how the methods of Rapid-Code-Generation were used in the development process of the HIL simulator subsystem. It shows, that not all of the required functionality could be created using RCG methods (i.e.: interrupt routines). However RCG is still a substantial asset as it enables very clear arrangement of the "source code" in the MATLAB Simulink, ease of implementation of the changes, and also ease of implementation of the algorithms created in MATLAB Simulink. Some of the unimplemented features could be taken as an inspiration for developers as an idea on what to improve.

References

[1] Todd R., Forsyth A.J., "HIL emulation of all-electric UAV power systems," Energy Conversion Congress and Exposition, 2009. ECCE 2009. IEEE , vol., no., pp.411,416, 20-24 Sept. 2009

[2] Vejlupek J., Chalupa J., Grepl R., "Model Based Design of Power HIL System for Aerospace Applications", 10th International Conference Mechatronics, (September, 2013, Brno) 2013

[3] Sova V., Grepl R., "Hardware in the Loop Simulation Model of BLDC Motor Taking Advantage of FPGA and CPU Simultaneous Implementation", 10th International Conference Mechatronics, (September, 2013, Brno) 2013

[4] Bin Lu, Xin Wu, Figueroa H., Monti A., "A Low-Cost Real-Time Hardware-in-the-Loop Testing Approach of Power Electronics Controls," Industrial Electronics, IEEE Transactions on , vol.54, no.2, pp.919,931, April 2007

[5] National Instruments – NI Power Electronics RCP and HIL System http://sine.ni.com/nips/cds/view/p/lang/cs/nid/211217

[6] dSPACE – HIL Simulation Systems http://www.dspace.com/en/pub/home/products/systems/ecutest.cfm

[7] Grepl R., "Real-Time Control Prototyping in MATLAB/Simulink: review of tools for research and education in mechatronics", IEEE International Conference on Mechatronics (ICM 2011-13-15 April, 2011, Istanbul), 2011

[8] Lamberský V., Grepl R. "Benchmarking Various Rapid Control Prototyping Targets Supported in Matlab/Simulink Development Environment", 10th International Conference Mechatronics, (September, 2013, Brno) 2013

[9] CANaerospace - the Airborne CAN Interface Standard http://www.stockflightsystems.com/canaerospace.html