Model-Based Development of MPPT Algorithm with Legacy Components Integration

Josip Babić, Tihomir Čihak, Siniša Marijan
Končar – Electrical Engineering Institute Inc.
Zagreb, Croatia
{jbabic, tcihak, smar}@koncar-institut.hr

***Abstract*— Model-based paradigm offers many opportunities for real-time embedded software development. On the other hand, there is a very high level of confidence in legacy software components and industry is reluctant to give them up in order to use modern development procedures. This work presents an approach to model-based development of real-time embedded control systems with systematic legacy components integration. The approach is validated on the development of real life maximum power point tracking algorithm.**

***Keywords*— model-based development; embedded system; real-time, MPPT, photovoltaic**

I. Introduction

Embedded systems are being used in an ever growing number of applications, from simplest toys to highly complex industrial, military and space systems. Along with faster and more complex hardware, embedded software is gaining importance and, according to [1], makes up to 85% of the value of the entire embedded system. Under the market pressure the software must be produced quickly and as bug-free as possible. Driven by these two opposing requests, Model Based Development (MBD) has emerged as the design approach of choice. As stated in [2], MBD is not just application of graphical domain-specific languages, i.e. "programming by drawing". A good definition of MBD is given in [3]: *In model-based development, the model is the central artefact and is used and systematically refined through the entire development process which is literally based on or centered around it.*

At the same time, a strong affinity for legacy software components exists in industry, accompanied by mistrust for new development processes that could introduce unexpected additional costs. This is especially true in traditionally conservative industries that deal with safety critical applications. Legacy software components are attractive because of the effort already invested in their development and testing and because they have acquired a high level of confidence through prolonged exploitation. It is generally accepted that value and reliability of a software component are increased with number of its applications, [4]. Legacy components can prove to be more efficient and more adequate for real-time (RT) applications than code automatically generated by modern MBD tools, [5].

This paper deals with the approach that enables model based development of embedded real-time control software with systematic integration of legacy components, thus providing benefits of both approaches. MATLAB/Simulink product family has been chosen as a MBD environment. Legacy components used in the development process have been inherited from proprietary application development environment that is used in the development of application programs for rail vehicles and power generation units for over two decades. Thus, many of the inherited software components have acquired huge amounts of working hours, sometimes up to several hundred millions.

The new MBD approach with legacy component integration has been validated in this paper on the example of maximum power point tracking algorithm (MPPT). This algorithm is used to extract maximum power from a power source with typically non-linear current over voltage characteristic. Typical example for MPPT application is a photovoltaic (PV) inverter. In this case, a MPPT algorithm is used to control the interface between the PV field and the load, i.e. to create an “adaptable load”, by means of power inverter.

Second section of the paper provides an introduction on model-based development of embedded software, while the third section gives an overview of legacy development process, tools and components. Integration of legacy software components into MBD flow is presented in the fourth section. Real-time model-based approach to control systems testing is introduced in fifth section. Sixth section describes used model of photovoltaic panels and structure of the implemented MPPT algorithm. Test results are given in the seventh section and the paper ends with conclusion and outlook for future work.

II. Model-Based Development

MBD process is illustrated in Fig. 1. It starts with requirements elicitation and formalization, proceeds with modeling and ends with code generation. Two-way traceability between various development artefacts should be ensured – from requirements through model to code and test cases. Testing is performed throughout the development: model is checked against requirements and the code, executed on PC or on the target, is checked against the model.

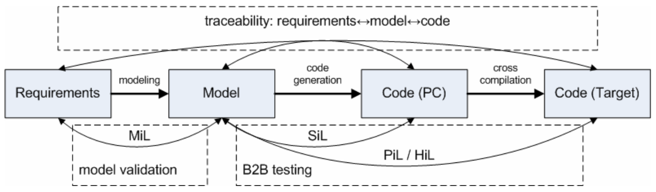

Fig. 1. Model-based development workflow

Existing MBD approaches for real-time embedded control applications do not systematically tackle legacy software component integration issue: the available tools rely on modern automatic code generation (ACG) tools and allow for simple integration of legacy component on a per-component basis, [6], while the literature mostly merely mentions the possibility of legacy code integration.

Analyzed model-based testing (MBT) methodologies and tools are not tailored for real-time embedded control systems. On the one hand, there are MBT methods that utilize formal procedures, e.g. [7], and approaches on the model-in-the-loop integration level, e.g. [1], that can only be used on higher abstraction levels where real-time properties aren't modeled. On the other hand, real-time MBT methods are designed for discrete systems, e.g. [8], or they require complex hardware-in-the-loop setups, e.g. [9]. The method proposed in this paper enables validation of real-time properties of hybrid control systems, which have both discrete and continuous behaviors, on the processor-in-the-loop (PiL) integration level.

III. Legacy Development Process, Tools and Components

A. Process

Embedded real-time software development process considered in this work is based on proprietary integrated development environment that utilizes legacy software components for code generation. Application software consists of one or more application programs, each executed on a separate hardware module based on microcontroller, microprocessor or signal processor.

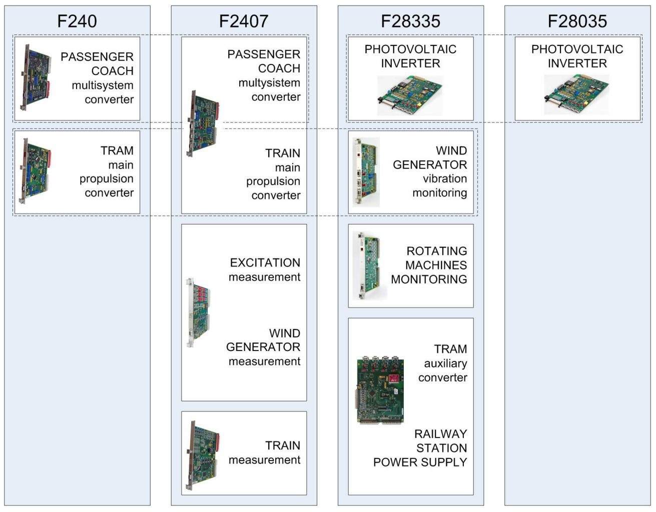
F240
F2407
F28335
F28035
PASSENGER COACH multisystem converter
TRAM main propulsion converter
PASSENGER COACH multisystem converter
TRAIN main propulsion converter
EXCITATION measurement
WIND GENERATOR measurement
TRAIN measurement
PHOTOVOLTAIC INVERTER
WIND GENERATOR vibration monitoring
ROTATING MACHINES MONITORING
TRAM auxiliary converter
RAILWAY STATION POWER SUPPLY
PHOTOVOLTAIC INVERTER

Fig. 2. Legacy components inheritance

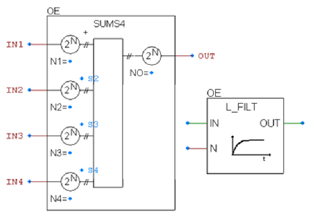

Fig. 3. Atomic component symbols

B. Tools

Proprietary integrated development environment consists of: GRaphical APplication (GRAP) programming tool, PC-based service and debugging application ZZT, microcontroller specific project environment and RTOS (Real-Time Operating System). Application programs developed in this environment are intended to be executed on hard real-time systems. The environment supports several different architectures of microprocessors, microcontrollers and signal processors. It is completely graphical, so there is no need for textual coding. Software components on higher level of complexity are built by interconnecting atomic software components represented by graphical symbols, similar to block diagrams, Fig. 3. Successfully built application is loaded onto the target, on which RTOS is running, using ZZT.

C. Components

On code level, each atomic component is a legacy macro program, hand-written in assembly language, carefully optimized and thoroughly tested. In Fig. 2, proprietary hardware modules based on Texas Instruments' C2000 family of microcontrollers are shown, together with projects in which they are used. A number of legacy software components have been inherited among these projects, e.g. a component for scaled summation of four signals and a component for filtration of logical signal, whose symbols are shown in Fig. 3.

IV. Legacy Software Components Integration

The in-house development of entire model based tool environment would be exceedingly expensive, so integration of legacy tools and processes into an off-the-shelf environment has been evaluated. MATLAB/Simulink tool family by MathWorks has been chosen. To preserve knowledge accumulated in macros behind GRAP atomic components, they have been used in code generation from Simulink models. In essence, Simulink has been used as a replacement for GRAP in building graphical modules, their compilation and linking into executable applications. The approach has been verified for C2000 family, but it can easily be expanded to other GRAP-supported processor or microcontroller architectures.

Generation of executable code from GRAP application program is illustrated in Fig. 4. Firstly, the graphical application program is built using atomic component symbols from a graphical library. Application can be divided into an arbitrary number of program modules, which can then be reused. From graphical program modules, GRAP creates source files. For every programming element in a module a macro call with all the necessary arguments is placed in the source file. In the compilation step, these macro calls are expanded using macro library resulting in a complete assembly listing of each module. By executing assembler on these files, object files are generated that can be linked into executable files.

Instead of graphical programming and code generation in GRAP, Simulink and MATLAB can be used as shown in Fig. 4. GRAP compatible model is created in Simulink and converted into GRAP source file by a set of M-functions. Further code generation is performed by calling the same tools as was case with GRAP, only here they are called from within MATLAB environment. The crucial step in this process is creation of source files appropriate for code generation from Simulink model, i.e. conversion of Simulink models into GRAP modules.

Assembled object files corresponding to modules designed in both GRAP and Simulink can be linked together. This way, algorithmically intense parts of application program can be developed in MATLAB environment and linked to hardware dependant input/output modules or legacy modules developed in GRAP. The greatest advantage of using Simulink for application or module development is its simulation capability. Built-in Simulink blocks are simulated in usual fashion, while GRAP-specific blocks are modeled in a way to closely emulate their assembly counterparts so model of the application should during simulation behave much the same as the generated code during execution on the target. Besides simulation, Simulink provides access to various other MATLAB features like requirements management and model checking tools.

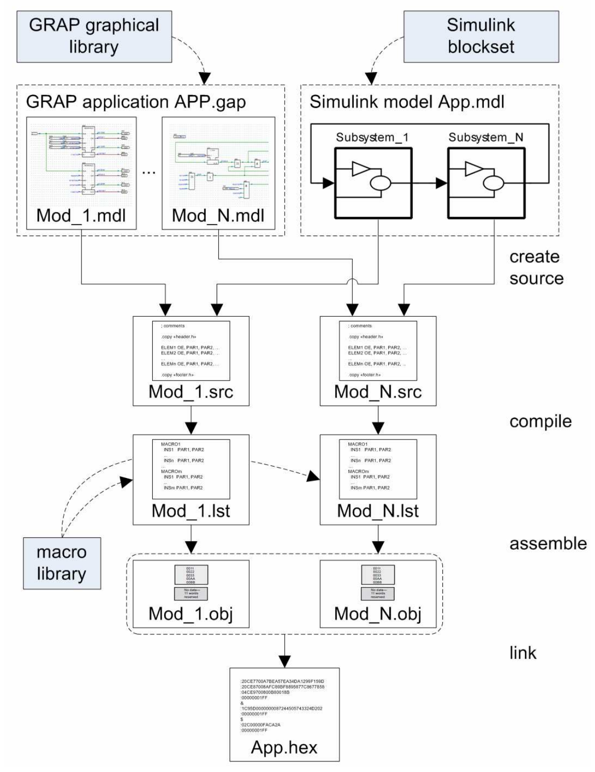

Fig. 4. Code generation using legacy software components

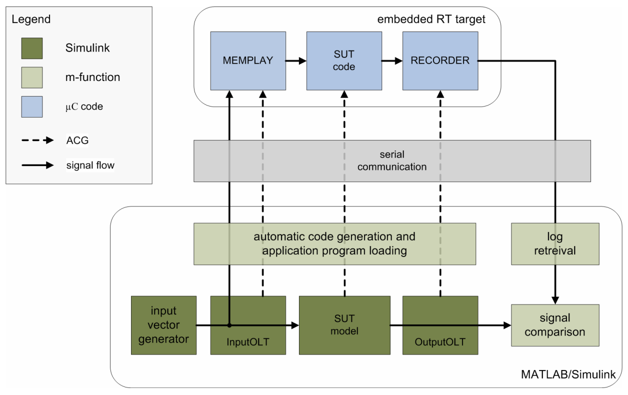

Fig. 5. Open loop testing

V. Model-Based Testing

Testing of RT control systems is particularly demanding because of their tight coupling to the plant (i.e. environment). Timing constraints additionally complicate this kind of testing. To tackle the complexity of control systems integration testing, we propose MBT on processor-in-the-loop integration level in three stages: 1) testing in an open loop, 2) testing in a closed loop with simulated environment, and 3) RT testing in a closed loop. RT properties are tested by applying RT testing patterns. These patterns as well as RT testing results have not been presented in this paper due to constricted space.

A. Open loop test

In the first step, algorithm is tested in an open loop, Fig. 5. Test input vector is generated in the simulation environment. System under test (SUT) inputs and outputs are recorded during SUT model simulation. Next, code for target microcontroller is automatically generated from the model. MEMPLAY component in Fig. 5 refers to reproduction of input vector from microcontroller memory, SUT code is generated from its' model and all relevant outputs are recorded to microcontroller memory, as depicted by RECORDER component in Fig. 5. Generated code is downloaded to the target through a serial communication interface, e.g. RS232, the code is executed and outputs are retrieved from the target memory. Simulation traces are compared to code execution results and test verdict is produced.

B. Closed loop test with simulated environment

The second stage is validation of the SUT in a closed-loop test with simulated environment, illustrated in Fig. 6. Here, models of the SUT and the environment are simulated inside the simulation environment, while SUT code generated from the model is executed on the target. In each simulation step, input is fed to the SUT model and to the code, one simulation step and one target task execution are performed and, finally, model and code outputs are compared. Feedback can be closed with SUT model or code in the loop, depending on the *OutputPiL* component settings. Signal exchange between simulation environment and the target is controlled by *InputPiL* and *OutputPiL* Simulink blocks and corresponding GRAP components RS232IN and RS232OUT.

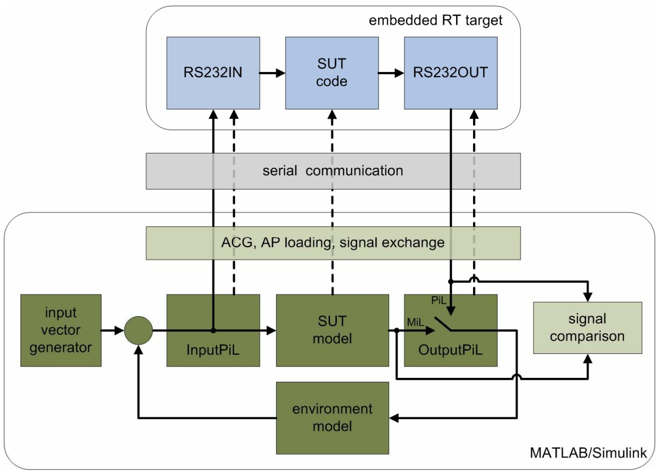

Fig. 6. Closed loop testing with simulated environment

This kind of testing cannot be executed in real-time because the simulation environment is executed on plain desktop computer. (*OutputPiL* and *InputPiL* blocks are implemented using Real-Time Windows (RTW) Target MATLAB toolbox.) Its' main application is validation of the SUT inside a closed loop after RT properties have been validated by open loop testing.

C. Real-time closed loop test

Sometimes, it can be necessary to test real-time properties of the control algorithm in a closed loop setting. This can be achieved by testing the control system in a loop with environment model running on the same target as the control algorithm. The preparation for this kind of scenario includes: 1) validation of the control algorithm code in open loop and non-real-time closed loop testing, 2) refinement of the environment functional model into implementational model adequate for ACG, 3) validation of the environment code by open loop testing and 4) validation of the closed loop containing control and environment code, as depicted in Fig. 7. Here, the same inputs are fed to closed loops in both simulation environment and target, and relevant signals are compared.

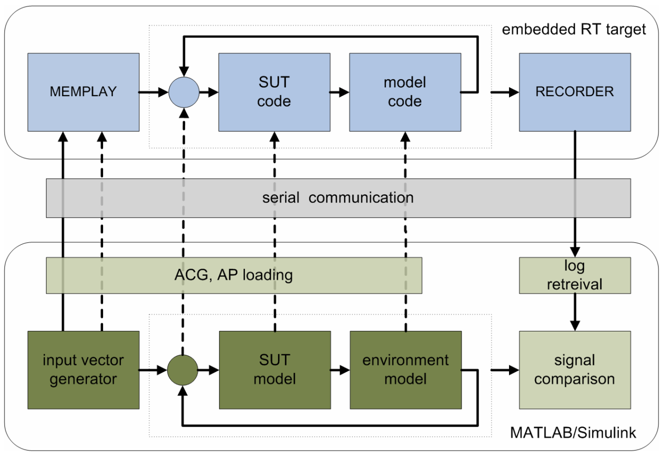

Fig. 7. Real-time closed loop testing

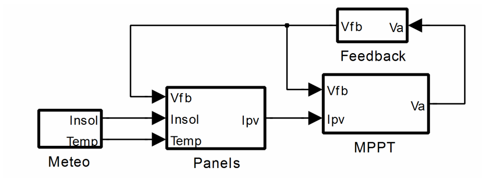

Fig. 8. Root view of the Simulink model

VI. Photovoltaic Panels Model and MPPT Algorithm

The presented MBD process and real-time testing methods are validated on the example of MPPT algorithm applied to a photovoltaic power generation unit. The root view of the Simulink model with meteorological conditions definition subsystem *Meteo*, photovoltaic panels' model subsystem *Panels*, and MPPT algorithm implementation subsystem *MPPT* is shown in Fig. 8. MPPT algorithm task period is by several orders of magnitude longer than inverter's step response to operating point change. That is why inverter response from the viewpoint of MPPT algorithm is approximated as instant and is not included in the model, [10] and [11].

A. Photovoltaic panel model

PV fields are constructed out of a number of PV panels connected in series and parallels. PV panels are, again, constructed out of a number of PV cells. Therefore, to validate an MPPT algorithm, a sufficiently accurate model of PV cell is needed. As shown in [12], PV cell can be modeled with (1), where I and V are cell output current and voltage, I_0 is saturation current, R_S and R_P are cell shunt and series resistance, I_S is the current generated by sunlight and A, T and k are diode ideality factor, cell temperature and Boltzmann constant respectively.

$$I = I_S - I_0 \cdot \left(e^{\frac{q \cdot (V + I \cdot R_S)}{k \cdot T \cdot A}} - 1 \right) - \frac{V + I \cdot R_S}{R_P} \qquad (1)$$

Typical photovoltaic cell characteristics used in our research are given in Fig. 9.

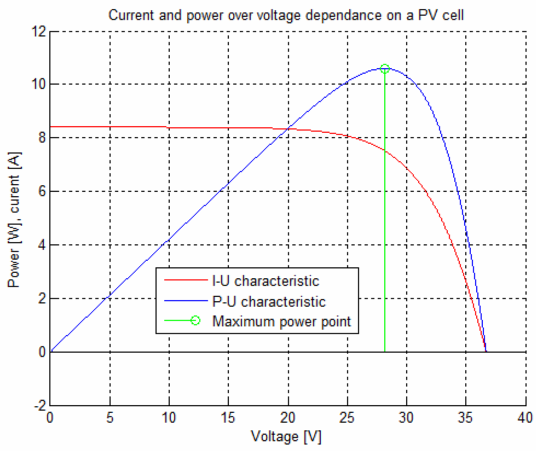

Fig. 9. Photovoltaic cell caracteristics

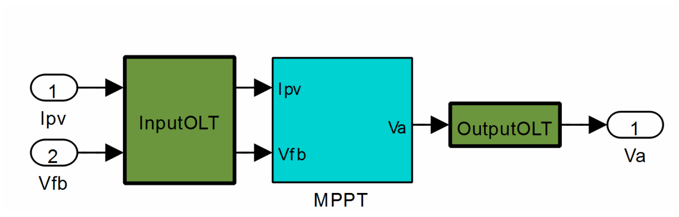

Fig. 10. MPPT model for open-loop test

The panels are modeled by an M-function. PV field was simulated by increasing PV cell outputs until satisfactory PV field power at MPP was gained.

B. *MPPT algorithm*

Many different MPPT algorithms have been developed and implemented with various degrees of success, [13] and [14]. In general, MPPT algorithms can be divided into "hill climbing" types, advanced MPPTs based on neural networks, fuzzy logic and similar, and "approximate" types where part of the PV cell characteristics are known or estimated. In practice, mostly "hill climbing" methods are used, as advanced algorithms can be hard to implement due to computing requirements, while other types do not operate with adequate precision for modern systems. Tested algorithm is a "hill climbing" variation known as Incremental Conductance algorithm, [15]. The algorithm is based on assessment of the slope of power-voltage curve of the PV panel.

VII. Test Models and Test Results

A. *Open loop testing*

For open-loop testing, the MPPT model is "padded" with *InputOLT* and *OutputOLT* blocks, where OLT stands for Open-Loop Test, as shown in Fig. 10. These Simulink blocks correspond to the blocks with same names given in Fig. 5. Back-to-back (B2B) comparison between simulation traces of the model and execution results of the code automatically generated from the MPPT subsystem is shown in Fig. 11, where a perfect match can be seen. The code is executed on embedded target system based on Texas Instrument's TMS320F28335 floating-point digital signal controller.

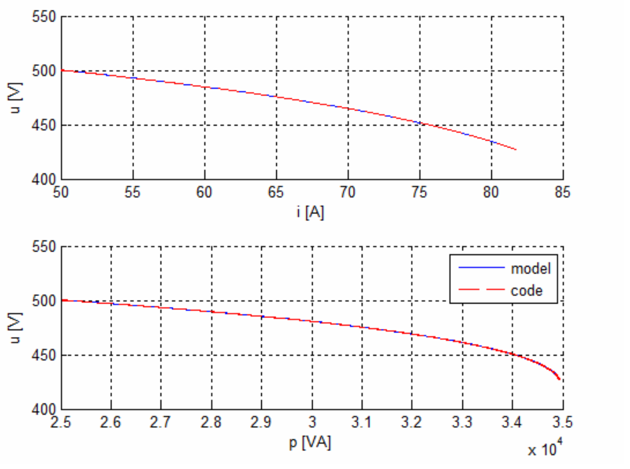

Fig. 11. U-I and U-P characteristics in open-loop test

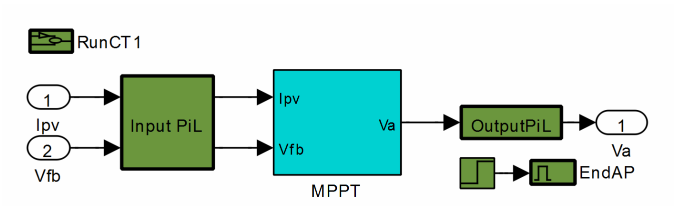

Fig. 12. MPPT model for non-RT closed-loop test

B. *Closed loop test with simulated environment*

MPPT algorithm model is prepared for non-real time closed-loop testing by adding *InputPiL* and *OutputPiL* blocks into signal path in same way as in Fig. 6. Here two additional blocks are added. *RunCT1* block sends a command to the target to execute code generated from the MPPT subsystem in each simulation step, Fig. 12. This way the algorithm executed on the target is fed with input and algorithm output is passed back into the model. The *EndAP* block stops target application when simulation time elapses.

Fig. 13 shows B2B comparison of U-I and U-P characteristics for cases when feedback loop is closed with model of the MPPT algorithm and when feedback is closed with code executed on the target. Here a mismatch can be seen caused by a single step delay introduced by the RTW communication. Nevertheless, code response follows simulation traces and reaches the same steady state.

C. *Real-time closed loop test*

For the testing of real-time properties of the MPPT algorithm, the whole model, including photovoltaic panel model, must be executed in real time on the target. Because ACG procedure doesn't support code generation from M-functions, the model of the panels must be re-implemented using Simulink blocks. The panels' model is broken down into two parts, Fig. 14. The *Panels1* subsystem implements the part of the M-function that is executed once in each simulation step, while *Panels2* subsystem implements the iterative part of the M-function.

Once again, Fig. 15 shows flawless comparison of U-I and U-P characteristics of the simulated model with the ones obtained by real-time code execution.

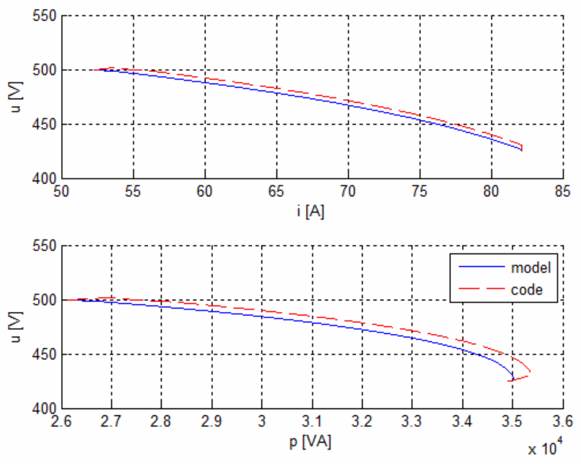

Fig. 13. U-I and U-P characteristics in non-real-time closed-loop test

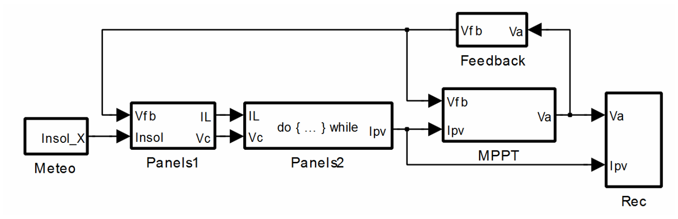

Fig. 14. Root view of the model for real-time closed loop testing

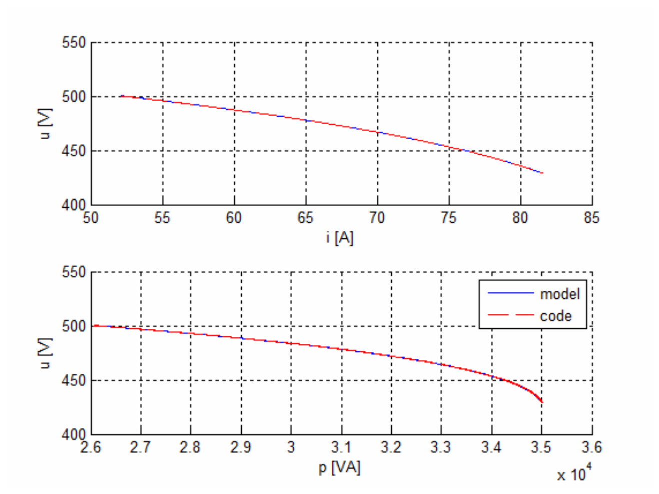

Fig. 15. U-I and U-P characteristics in real-time closed-loop test

7. CONCLUSIONS AND FUTURE WORK

Our work shows that it is possible to systematically integrate legacy software components into the workflow of model-based development of real-time embedded control applications. A framework has been implemented by means of MATLAB/Simulink environment that consists of a custom Simulink blockset and a set of conversion utilities. This concept enables automatic code generation from Simulink models using legacy software components.

A gap has been identified in model-based testing methodologies for real-time hybrid control systems and a three-step testing method has been proposed to fill it. Real-time properties of a control application are firstly validated in an open loop test where results of automatically generated code execution are validated against simulation traces of the model. Secondly, behavior of the control algorithm in the closed loop with the simulated environment is checked in a non-real time closed loop test. Thirdly, if needed, real-time properties of the tested system can be validated in a closed loop by executing model of the environment alongside control code on the target system.

The proposed approach has been verified on an example of maximum power point tracking algorithm for a photovoltaic power generation unit. Results show that synthesized MPPT algorithm is fast, without MPPT tracking error (steady state error). Also, no stability problems were found during testing and the algorithm responded adequately to both slow and fast simulated environment changes. Moreover, results show that for algorithm with adequately selected parameters, no overshoot or oscillations around maximum power point are present. The algorithm is currently being validated on a PV inverter developed at our company.

To provide truly functional real-time model-based development environment with legacy components integration, our custom Simulink blockset should be expanded to encompass more of the available legacy software components. Also, the development framework should be adapted to other GRAP-supported families of target systems. Currently, the environment is used in testing SIL4 compliant control system running on microcontroller based on popular 8051 architecture. A workaround the delay in non-real time closed loop testing should be devised.

The proposed testing procedure for the real-time control applications should be supplemented with automatic test case generation capability that verifies real-time properties such as execution time, interrupt latency, multitasking, priority scheduling and others.

References

[1] J. Zander-Nowicka, “Model-based testing of real-time embedded systems in the automotive domain,” PhD thesis, Technische Universität Berlin, 2009.

[2] B. Schätz, A. Pretschner, F. Huber, and J. Philipps, “Model-based development of embedded systems,” tech. rep., Technische Universität München, 2002.

[3] A. Rau, “Model-based development of embedded automotive control systems,” PhD thesis, University of Tübingen, 2002.

[4] S. Marijan, “Sustainability of embedded control systems for rail vehicles and power generation units,” PhD thesis, University of Zagreb, 2011.

[5] J. Babić, S. Marijan, and I. Petrović, “The comparison of MATLAB/Simulink and proprietary code generator efficiency,” in Proceedings of the International Conference on Electrical Drives and Power Electronics, EDPE 2009, 2009.

[6] N. Ajwad, “Evaluation of automatic code generation tools,” Master thesis, Lund University, April 2007.

[7] J. Jürjens, D. Reiß, D. Trachtenherz, “Model-based quality assurance of automotive software,” in Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems, MoDELS '08, 2008.

[8] H. Wolfram Neukirchen, “Languages, tools and patterns for the specification of distributed real-time tests,” PhD thesis, Georg-August-Universität zu Göttingen, 2004.

[9] B. Lu, X. Wu, H. Figueroa, and A. Monti, “A low-cost real-time hardware-in-the-loop testing approach of power electronics controls”, IEEE Transactions on Industrial Electronics, vol. 54, no. 2, pp. 919-931, 2007.

[10] M.P. Kazmierkowski, R. Krishnan, and F. Blaabjerg, “Control in power electronics – selected problems,” Elsevier Science, Academic Press, 2002.

[11] R. Teodorescu, M. Liserre, and P. Rodríguez, “Grid converters for photovoltaic and wind power systems,” John Wiley & Sons, 2011.

[12] F. M. González-Longatt, “Model of photovoltaic module in Matlab™,” 2DO Congreso Iberoamericano de estudiantes de ingenieria electrica, electronica y computacion, II CIBELEC 2006, 2006.

[13] D.S. Morales, “Maximum power point tracking algorithms for photovoltaic applications,” Masters thesis, Aalto University, 2010.

[14] N. Femia, G. Petrone, G. Spagnuol, and M. Vitelli, “Power electronics and control techniques for maximum energy harvesting in photovoltaic systems,” CRC Press, 2012.

[15] D. Sera, T. Kerekes, R. Teodorescu, and F. Blaabjerg, "Improved MPPT Algorithms for Rapidly Changing Environmental Conditions," in Proceedings of the 12th International Power Electronics and Motion Control Conference, 2006. EPE-PEMC 2006., pp.1614-1619, 2006.