Introducing Model-based Design Methodology with LabVIEW to Teaching ARM-based Embedded System Design

[1]Lakshmi Teja Mullapudi and [2]Nannan He*
Department of Electrical and Computer Engineering Technology
Minnesota State University at Mankato
[1]E-mail: lakshmi-teja.mullapudi@mnsu.edu
[2]E-mail: nannan.he@mnsu.edu

Abstract

This paper presents our latest experience of introducing the new topic of model-based design (MBD) concepts and tools to a Programming Tools (PT) course for educating students to be capable of utilizing modern tools for correctly developing complicated ARM-based embedded systems. It describes the course contents, student outcomes and lecture and lab preparation for teaching this topic with the emphasis on two sub-topics. Firstly, we present the details of using NI LabVIEW tool in programming ARM Cortex-M MCUs or ARM Cortex-A9 MCUs on the embedded device like NI myRIO for fast developing embedded applications. Secondly, to integrate an on-going research effort on the model-based verification into this course, we also introduce model-checking and the tools that have been utilized in the research project. This new topic helps introducing students the latest research advances which promote the wide applications of the MBD in safety-critical embedded applications. Our primary experience shows that the project-based learning approach with the graphical programming tools and selected MCUs is efficient and practical to teach the MBD of 32-bit MCUs programming.

1. Introduction

As the computing power of microprocessor and the complexity of peripheral devices greatly increase, the microcontroller (MCU)-based embedded system design become more complicated. As a result, the complexity of the program for controlling modern MCUs also increases. On the other hand, programming is often considered to be difficult for engineering students. As we observe, electrical and computer engineering students usually study only one software programming language such as C for one semester in the first year, but have little chance to apply the programming skills from that course in other courses compared with software engineering or computer science major students. When building embedded systems using MCU in design projects in the senior year, many students spend a large amount of time in debugging the C program so as to pass the compilation, but have little time to design algorithm and verify the correctness. This problem becomes worse when the code size and complexity increase. Moreover, as powerful 32-bit MCUs are embedded in larger scale mechanical devices, some mechanical engineering students are also interested in applying MCUs to their senior projects. Programming MCUs is even more challenging for them.

Model-based design (MBD) is an emerging development methodology for modern software systems. Its efficiency has been demonstrated in the development of safety-critical embedded software systems in industry. MBD promotes the use of graphical domain-specific notations to create executable design models. Automated code generation from design models to the implementation is an important feature of MBD. Engineers with limited programming experience could be greatly relieved from low-level programming and focus on the domain-specific problems. Moreover, continuous model-based verification and validation as another feature of the MBD enable early identifying design flaws so as to avoid the costly late-stage design fixes. More graphical system modeling tools started supporting MBD, such as LabVIEW from National

Instruments (NI). To be specific, NI supports using LabVIEW to directly program various microcontrollers from 8-bit microprocessor on the Ardunio, to 32-bit ARM Cortex-M and ARM Cortex-A series microprocessors (like the one integrated in the embedded hardware device myRIO).

The main objective of our Programming Tools (PT) course is to teach students modern programming tools so that they are capable of utilizing these tools to facilitate the design and implementation of embedded systems. The new topics integrated into this course cover the basic workflow of the MBD methodology, unique features of the MBD like executable specification, automated C code generation and continuous model-based verification and validation (V&V), the software tools that support MBD in the development of MCU-based embedded systems and their high-level graphical input programming languages, such as LabVIEW and Matlab/Simulink. Moreover, in order to encourage students to participate research, we also introduce a new topic of the model checking and the relevant tools that facilitates the model-based V&V, which relates to an ongoing research project conducted by the faculty. Among these newly added topics, we focus on teaching students how to use LabVIEW to directly program ARM Cortex-A MCUs and the model-based V&V. Although, LabVIEW, as a virtual instrument environment to create graphical diagrams to interface with certain NI-supported hardware and perform numerical analysis, was also exposed to students, it is our new experience of teaching students to use LabVIEW to program general-purpose MCUs.

The LM3S8962 evaluation kit from Texas Instruments and the embedded device NI myRIO are employed as two main hardware platforms to teach programming MCUs with LabVIEW in this course. The LM3S8962 kit includes an ARM Cortex-M3 MCU. We use it in lab sessions. The myRIO is more powerful. It uses the Xilinx Zynq Z-2010 reconfigurable multiprocessor architecture with two microprocessors, each of which has its own memory and peripherals. The fixed processor is an ARM Cortex-A9 dual-core processor with a fixed set of peripherals. It can be programmed by either LabVIEW or C. Another processor is reconfigurable with the Xilinx Artix-7 field-programmable gate array (FPGA). Hardware peripherals associated with it can be customized in the FPGA. We apply myRIO to developing the course projects.

This paper reports our continuous teaching efforts of introducing MBD and tools to the development ARM-based embedded systems. Our previous effort of integrating MBD concepts and supporting tools (i.e., Matlab/Simulink and Eclipse-based software programming tools, etc.) for system design modelling into this course has been reported before [1]. This paper mainly presents our experience of integrating the new topics of the MBD methodology and supporting tools for programming 32-bit ARM Cortex MCUs into an existing PT course. The description of this course is first given, including learning outcomes, course contents and project organization. Next, we present the LabVIEW lab projects developed using the selected boards. Then we present our experience of incorporating on-going V&V research to this course. Finally, the paper is ended with conclusion and future work.

2. Course description

The PT course is usually required for computer engineering or computer science major students, but an elective course for other engineering major students in many universities in US. We offer this course at the systems level, which focuses on key concepts of system-level programming (e.g., LabVIEW and Matlab/Simulink); tool chains for group software development; and advanced topics on software system design, implementation, testing strategies and documentation [2]. It is organized as 2 hours of lecture and 2 hours of laboratory per week. At the end of the course, students are capable of utilizing existing programming tools to develop a complete hardware/software embedded system as their course project. As most embedded systems design efforts in industry have moved to from simple 8-bit MCUs to modern 32-bit MCUs with real-time computing and networking capabilities, the software development became more complicated. Programming such devices to develop complex embedded systems with a large number of constraints is increasingly challenging for engineering students with limited programming background. Educators have recognized the

need to introduce some efficient and cost-effective programming tools to students [3]. Our new course enhancements on the MBD have three objectives.

- To improve students' awareness of the MBD methodology in programming modern MCUs.
- To introduce students, the MBD tools which allow programming MCUs using high-level graphical programming/modelling languages.

To develop students' appreciation for formal verification tools such as model checker to automate or semi-automate the model-based V&V for the cost-effective development of reliability-critical embedded systems.

2.1 Course learning outcomes

We derive the following course learning outcomes under the above three major objectives.

1. To demonstrate the knowledge of the MBD methodology.
   1.1 To articulate the workflow of the MBD and its unique features.
   1.2 To understand the benefits of the MBD in programming MCUs.
2. To demonstrate the capability of programming MCUs using at least one high-level graphical programming/modeling language
   2.1 To get familiar with common software tools supporting programming MCUs using graphical programming/modeling languages.
   2.2 To comprehend the underlying working mechanism of programming MCUs using high-level programming/modeling languages.
   2.3 To be able to create high-level programs using at least one graphical programming language (i.e., LabVIEW, Matlab/Simulink) to control/augment MCUs or hardware devices with microprocessors.
3. To demonstrate the knowledge of advanced model-based V&V techniques
   3.1 To understand the unique features of model-based V&V techniques.
   3.2 To understand automated formal methods (i.e. model checking and theorem proving) for proving the correctness of design and code in safety-critical applications.
   3.3 To understand the formal verification tools for the model-based V&V.

2.2 Course contents

As described in the introduction section, the new topics integrated into this course include the basic workflow of the MBD methodology, unique features of the MBD, the MBD tools that support programming MCUs using the high-level graphical programming languages, and the formal verification for the model-based V&V. Among these newly added topics, we emphasize using LabVIEW to directly program ARM Cortex-A MCUs and the application of automated formal method like model checking for the model-based V&V. Course materials were drawn from white papers, NI tutorials, presentations and manual, on-line forum, textbook, technical papers and example projects [4-12]. Table 1 shows the major course topics.

Table 1. Course topics

1. MBD concepts
Basic MBD workflow
Key features of MBD (i.e. executable design models, automated code generation)
Comparison of MBD with conventional development methodologies
2. Programming MCU using MBD
Introduction to common MBD software tools (i.e. LabVIEW, Matlab/Simulink)
Event-driven programming using LabVIEW

Graphic component library in LabVIEW with mathematical, signal processing functions
LabVIEW Embedded module for ARM MCUs
LabVIEW C Code generation
Develop LabVIEW application to program MCUs (i.e., myRIO, TI LM3S8962)
3.Model-based V&V and tool for automating verification
Model-based testing and formal method (i.e., model checking)
Model-based verification tools for testing and model checking

2.2.1 Model-based design concepts

We introduced the workflow of the MBD methodology to our students during the first week. Five basic phases from requirement analysis, system design, implementation, integration to continuous verification, were covered. Based on the MBD process illustrated in Figure 1, we discussed main differences between MBD and other software development processes like the waterfall model. Students were also guided to learn new concepts along each basic MBD step. Three unique features of the MBD were taught in detail: executable design specification, automated code generation and model-based V&V. The LabVIEW and its embedded module for developing ARM MCUs are introduced as the case study of teaching these features in practice. The experience of teaching the model-based V & V is introduced in the next section.

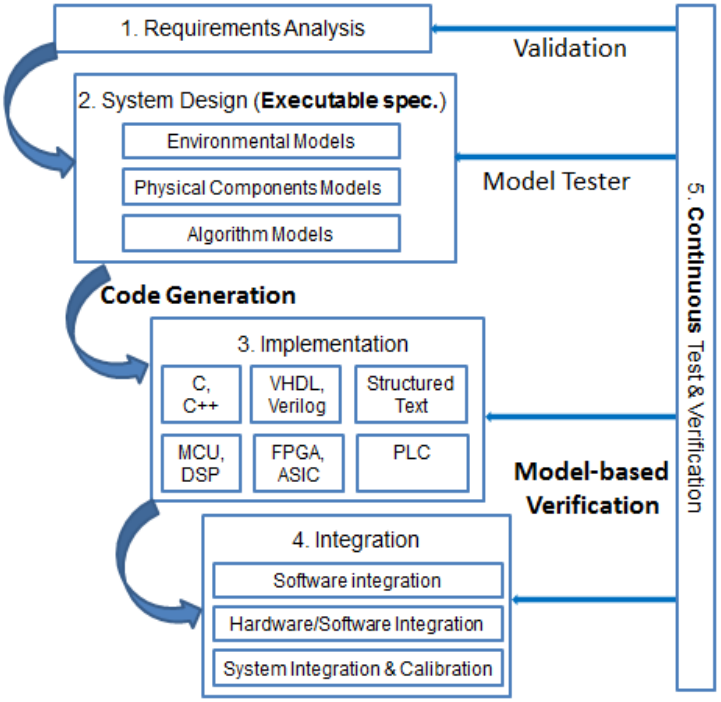

Figure 1. Main steps in Model-based design

2.2.2 Executable design specification

In the MBD, the design specification should be formulated in terms of executable models shown in Figure 1. It can unambiguously model the entire system functionality, including the environment, physical component and design algorithm. Compared with text-based design specifications, these models have one distinguished benefit of supporting early validation and testing via models simulation.

NI LabVIEW supports the MBD and creating executable specification models. One of its major strengths was in giving non-programmers a productive programming tool, lowering the barrier to software development. Its graphical design environment provides developers with a higher level of abstraction than conventional imperative programming models like C. LabVIEW supports heterogeneous compositions of several models of computation: continuous system expressed as differential equations, discrete system expressed as difference equations in the signal flow model and concurrent state machines expressed as State charts model. The predominant input language in LabVIEW is structured Dataflow which is referred to as the graphical language. LabVIEW applications are commonly design to interface with sensors, actuators, data acquisition devices and other embedded devices.

2.2.3 Automated code generation

Increasing number of automated code generation tools has been created in past ten years. Engineers with limited programming experience could be greatly relieved from low-level programming, and focus on domain-specific problems. Moreover, depending on application purposes, design models can be synthesized to different implementation languages. For example, programs coded in the Structured Text (a PLC language) are generated for PLC automation applications; VHDL or Verilog code is generated for hardware specification models in FPGA or ASIC applications; for MCU control or DSP applications, models are translated to C/C++, which are dominant programming languages in these applications.

We studied the basic features of the LabVIEW C Generator. It allows users to quickly develop design algorithms graphically in LabVIEW, create the C code that translates from the algorithms, and integrate the C code into any third-party embedded toolchain to download, run and debug it.

As a practical example, we could demonstrate students how to develop a LabVIEW program for C code generation, configure C code generation settings, generate optimized C code, and deploying the C Code from the TI Code Composer Studio IDE to the TI MSP 430 MCU.

3. Lab projects design using NI LabVIEW

The lab projects that we prepared for this course serve different teaching purposes. The lab exercises utilize the TI EK-LM3s8962 board and Keil ULINK2 debugger. These exercises were mainly developed based on the 12 example projects provided in the NI LabVIEW Embedded Module for ARM package. These projects demonstrate common MCU applications such as reading Analog inputs, CAN, interrupts, PWM generation and serial communication, etc. They can also be used as the basis to develop more complicated project. Table 2 gives an overview of seven lab projects that were introduced to students in this course.

After students completed all lab exercises, they made use of the NI myRIO to develop course projects. The online myRIO community provides a rich set of completed or ongoing projects [13]. Students have developed multiple course projects with myRIO, such as acceleration indicator, current measurement, motion detection for home security application. We expect that students could use myRIO to efficiently develop complicated applications in their senior design project.

Table 2. Lab exercises on programing ARM Cortex M3 with LabVIEW

Project Name	Function description	Learning purposes
Blinky	- Blink the status LED on the board at a fixed frequency.	- Create, build and debug the LabVIEW project for programming the EK-LM3s8962 board; - Get familiar with the component library in LabVIEW; - Program the core peripherals of ARM Cortex-M3 MCU in LabVIEW.
Analog inputs reading	- Sample analog signals and display a waveform on the on-board Organic LED screen.	- Study resources in the Element I/O category provide by LabVIEW to define System Inputs and Outputs. - Use the OLED screen control component included in the ARM category for the LabVIEW programming.
Basic interrupts	- Configure Timer to trigger an event every 500 ms;	- Study interrupt related components in the ARM category (i.e., Allow/Ignore/ Enable/Disable interrupts)

	- Once the event occurs, the program spawns a new thread and invokes the interrupt handler, which increments a counter.	- Use these components to specify the interrupt source, the condition on which the interrupt source triggers an interrupt, and the VI that handles the interrupt.
Digital Input/ Output	- Detect a button press on the UpButton and DownButon. - Increment or decrement the blinking rate of the LED according to which button was pressed.	- Study the digital Input and Output components in the sub category of "Elemental I/O" under the ARM category. - Add and apply these new elemental I/O components to a LabVIEW program.
PWM waveform display	- Initialize LCD. - Output the control value calculated by a PID module using the PWM. - Write PWM output to LCD.	Study the PWM output module and display modules specific to the selected MCUs (i.e., Display init/set background/set foreground/draw string) - Use these modules in a LabVIEW program.
CAN	- Connect the CAN board to the MCU board. - Display the status of the elements on the CAN board back to the MCU board.	Study the CAN related components in the ARM package (i.e., CAN open/start/Set receive/ read/write) - Use these modules for the CAN communication in a LabVIEW program.
TCP	- Set up a TCP data server which runs on a host PC and TCP data client which runs on the MCU board. - Exchange data between the server and client.	Study TCP components under the data communication category in LabVIEW. (i.e., TCP open connection, close connection, read, write, listen, wait on Listener) - Use these modules for developing the TCP/IP based Ethernet connection applications.

4. Model-based validation and verification (V&V)

Model-based V&V represents a set of V&V techniques continuously applied through the MBD process. All of them contribute to three important benefits: (1) Detect errors early in the development; (2) Reuse test throughout development process. (3) Reduce use of physical prototypes. In this course, we introduce model-based V&V techniques in following three aspects.

Firstly, common quality control techniques in software engineering [14] were recapped and compared. Validation targets at answering the question "Are we developing the *right* system?", while verification aims at answering a different question "Are we developing the system *right*?" Formal method and testing are two kinds of verification approaches. In mission-critical systems, where bugs may incur disastrous effects, formal methods are employed to guarantee the correct behavior with respect to the system requirements. In comparison, testing is scalable and easy to apply although it is limited to detect bugs, but cannot ensure the correctness. Unit testing, integration testing and system testing are three common testing practices in software systems development. The purpose of recapping quality control techniques is for students to clarify the typical usage differences among these techniques.

Secondly, V&V techniques applied at different MBD steps are discussed in class. During the initial requirement analysis step, a validator is applied to ensure that extracted requirements correctly match the intended use. In the design step, a model tester or a simulator can be utilized to check whether the executable design specification satisfies requirements obtained in the first step. Unit testing is typically applied to check

if the implementation is consistent with design models. Integration testing and system testing are initiated from the integration step. Formal methods are applied to check critical components in both design and implementation.

Thirdly, latest advances of model-based V&V in both academy and industry are exposed to students. This is one of our new teaching endeavors in integrating an on-going research results into advanced undergraduate or graduate courses. This research project investigates the hybrid of automated formal method and model-based testing to achieve synergistic benefits. One of the research topics is about using model checking tool to automatically generate test cases from the design models and then applying the test cases to verifying the consistency between the design and the implementation in the later development stage. In this course, we first introduced students the basic features of model checking and then the model checking tools.

Model checking is one of the most commonly used formal techniques. The inventors of model checking have been recognized by the 2007 ACM Turing award due to the signification impacts of this approach in the hardware and software verification in industry. Model checking can be fully automatic without much expertise in formal logic reasoning. It differs from testing as it aims at an exhaustive exploration of the state space of the model, thereby providing a correctness guarantee that is rarely achieved by means of testing. More importantly, when the models under verification fail to satisfy a given specification, counterexamples (i.e., error traces) can be generated, which illustrate the erroneous behaviours of the system design. This information can be very valuable for debugging. In our research, error traces are utilized to derive test suites from design models. In this course, two active open source model checking tools were introduced to students.

UPPAAL: It is a well-known model checking tool for real-time systems [15]. With UPPAAL, the behavior of timed systems can be graphically modeled using the timed automata formalism extended with various modeling features. This tool consists of a graphical editor and simulator, and a model-checker. This checker performs an exhaustive symbolic analysis of the model and provides either a proof that the model satisfies a property, or a counterexample including a trace of actions and delays exemplifying how the property is violated. It has been applied successfully to a variety of industrial cases. Recently, this tool has been extended with new functions for test generation and controller synthesis. The ultimate goal of these updates is to enhance UPPAAL as an integrated tool suite for the MBD development lifecycle of embedded real-time systems

CBMC: It is a model checker for software verification [16]. It can take as input a low-level ANSI-C program and, formally check safety properties like the correct usage of pointer constructs, array bounds and user-provided C assertions. Given a program *C*, a property *P* and a bound *k*, the verification includes three steps: i) unrolling k times all loops structures in C; then ii) translating the resulting program without loops and property into a Boolean formula in Conjunctive Normal Form (CNF); and finally (iii) giving the result to a SAT solver. If the SAT solver returns false, the property holds, otherwise the property does not hold within the bound *k*. This tool can be used to directly verify safety-critical properties in the implementation source code and indirectly verify high-level language models like Simulink after being transformed into C code.

5. Conclusions

This paper presents our latest experience of introducing the new topic of model-based design (MBD) concepts for educating students to be capable of utilizing modern tools for correctly developing complicated ARM-based embedded systems. We firstly describe the details of using NI LabVIEW tool in programming ARM Cortex-M MCUs or ARM Cortex-A9 MCUs on the embedded device like NI myRIO for fast developing embedded applications. Secondly, to encourage students to participate the research on the model-based verification, we also introduce model-checking and the tools that have been utilized in the research project. Our primary experience shows that the project-based learning approach with the graphical programming tools and selected MCUs is efficient and practical to teach the MBD of 32-bit MCUs programming.

6. References

[1] Nannan He and Han-Way Huang, "Utilization of Eclipse-based Software Tools in Teaching a New Software Development Methodology to Engineers", *Proceedings of Annual ASEE conference, International Forum, 2014*.

[2] Links to some system-level PT courses: http://www.cs.washington.edu/education/courses/cse374/; http://web.eecs.utk.edu/~huangj/cs360/; http://school.eecs.wsu.edu/undergraduate/cpts/courses/360

[3] Paul G. Flikkema. "Approaching the Design of Complex Engineered Systems: A Model-based Approach Informed by System Thinking". *Proceedings of ASEE PSW Conference, 2012.*

[4] Joseph Yiu, Ian Johnson, "The Many Ways of Programming an ARM Cortex-M Microcontroller", ARM white paper, March, 2013.

[5] NI Tutorial (NI-Tutorial-11784-en.pdf). "LabVIEW C Code Generation Technology Basics", Dec., 2010. NI manual (373192a.pdf). "Getting Started with the NI LabVIEW C Code Generator".

[6] NI Tutorial (NI-Tutorial-6207-en.pdf). "ARM Microcontroller Development with LabVIEW", Dec., 2014.

[7] M. Kaufmann, J. Kornerup and M. REitblatt. "Formal Verification of LabVIEW Programs Using the ACL2 Theorem Prover". ACM ACL Conference, 2009.

[8] J. Jensen, E. A. Lee and S. A. Seshia. "An introductory Lab in Embedded and Cyber-Physical Systems", http://LeeSeshia.org/lab, First edition v1.6, 2014.

[9] Joseph Yiu. "The Definitive Guide to the ARM Cortex-M3 and Processor", Texas Instruments, Imprint Newnes, 2009.

[10] NI myRIO online resource, http://www.ni.com/myrio/.

[11] NI presentation. "Next Generation Graphical Programming with LabVIEW for ARM Microcontroller".

[12] NI myRIO online forum, https://decibel.ni.com/content/community/academic/products_and_projects/myrio

[13] R. Pressman, "*Software Engineering: A Practitioner's Approach*", New York: McGraw-Hill, 2009.

[14] UPPAAL tool, http://www.uppaal.com/

[15] E. Clarke, D. Kroening, and F. Lerda. "A tool for checking ANSI-C programs". *In TACAS, pp 168-176*. Springer, 2004.