SIMULATION OF MCU HARDWARE PERIPHERALS

R. Bartosinski, J. Kadlec

Institute of Information Theory and Automation, Dept. of Signal Processing

Abstract

This paper describes one possible way, how MCU hardware peripherals (e.g. PWMs, Timers, ADCs, etc.) can be simulated in Matlab/Simulink environment and how code controling these peripherals can be generated. The described implementation of the MCU hardware peripherals is based on using function-call outputs as peripheral interrupts and external blocks called Methods as peripheral control functions. In the generated code by Real-Time Workshop, the Method block is represented by directly called corresponding function of the used periphery.

1 Introduction

We needed to simulate hardware MCU on-chip peripherals in our project SESA. The project is focused on integration of Matlab/Simulink and Processor Expert (PE), where Matlab/Simulink is used for system modeling and code generation from model for embedded systems. Simulink with Real-Time Workshop and a project specific target called PEERT generate high-level code of the designed system. PE generates code on a low-level level, which is used as a hardware abstraction layer with the same application programing interface for all supported MCUs. PE checks all settings of each periphery according to settings of a CPU and all other peripherals in a MCU to manage implementation of the designed system.

The PEERT target generates code for peripherals as one main function (e.g. converting analog signal to digital for ADC block) with interrupts - called 'events' in PE. PE offers prepared functions (called 'methods' in PE) for all peripherals too. Methods can read and set the current values and settings of each periphery and they affect the main function of the periphery (block in Simulink) therefore we needed to simulate not only the main function but to simulate methods as well. Many offered RTW targets are for specific MCUs and majority of them haven't got implemented simulation behaviour.

In the paper, there is briefly described the project SESA at first. Then MCU hardware peripherals are generally described. The main section is focused on simulation of MCU peripherals with external simulink blocks.

2 SESA project

The SESA project is mainly focused on integration of two tools - Matlab/Simulink and Processor Expert for automatic code generation for embedded systems [1].

PE is advanced, component oriented, open Rapid Application Development (RAD) environment for embedded systems. The main task of PE is to manage CPU and other hardware resources and to allow virtual prototyping and design. PE contains knowledge base of information about MCUs and its on-chip peripherals. They are divided to separate units called Embedded Bean (bean). Beans encapsulate functionality of basic elements of embedded systems like MCU core, MCU on-chip peripherals, standalone peripherals, virtual devices, and pure software algorithms and change these facilities to properties, methods, and events (like objects in OOP). Beans are well tested software and that is why they can save months of work of the expert programmer [4].

Simulink is well known environment for model creation and simulation and it is used with Real-Time Workshop for code generation of a control application in this project. The SESA

simulation and generation flow is in the figure 1.

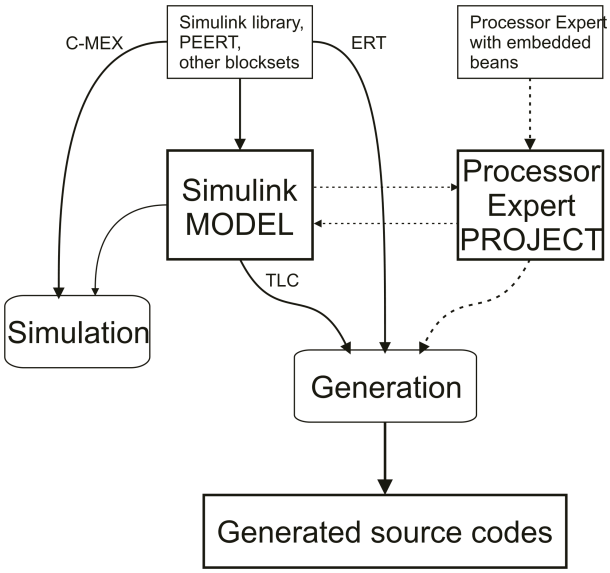

Figure 1: The SESA flow for generation code from a Simulink model

Simulation and code generation of implemented blocks is dependent on properties of blocks which are synchronized with PE beans through COM connection. It allows to use PE Bean Inspector window for property settings, as well. This concept takes advantage of PE knowledge base about entire created system and its dependency on the other used beans/blocks.

The result of the SESA project is a library and a blockset for Matlab/Simulink. The library provides interconnection between Simulink and PE. The blockset contains blocks which represent selected hardware peripherals (figure 2). All blocks have S-functions which simulate behaviour of the coresponding peripherals. [1, 2, 3]

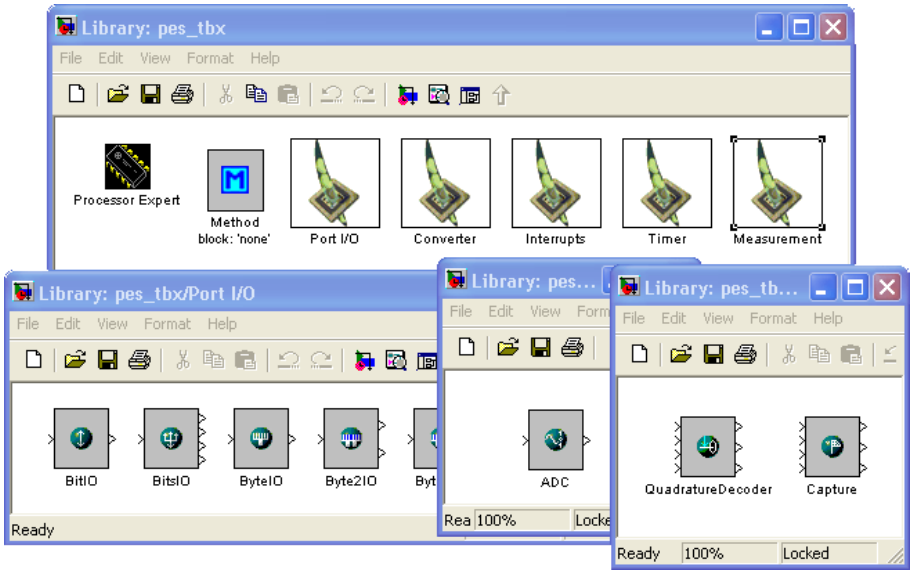

Figure 2: The SESA blockset

3 MCU hardware peripherals

MCUs have implemented many kinds of on-chip peripherals. For example the Freescale DSP 56F8367 contains up to two 6-channel PWMs, four 4-channel 12-bit ADCs, two Quadrature Decoders, four quad Timers, digital IOs and others [5].

Each periphery provides its specific functionality separately and according to setting of its own registers. These registers are set and read from the CPU by user program. One side of periphery is normally directly connected to MCU external pins as inputs (e.g. analog input of ADC) or outputs (e.g. output of generated signal from PWM) or both directions (e.g. GPIO). Majority of peripherals invoke interrupt to inform CPU that something is happening, as well.

PE provides low-level code for periphery initialization, control and interrupt processing functions and it provides uniform API for all supported MCUs which contain the selected periphery. A user then uses prepared functions and he needn't write his own initialization and control code.

For example we can see more detailedly on a Quadrature Decoder (QD) [6]. A QD circuit decodes quadrature encoder signals, normally two 90 degrees out-of-phase pulses, into count and direction information. This periphery has minimally two inputs - signals PHASEA and PHASEB and it can has next two signals Index and Home for counting number of rotations and indicating home position. If PHASEA leads PHASEB, assuming motion is in the positive direction. However, if PHASEA lags PHASEB, the motion is in the negative direction (figure 3).

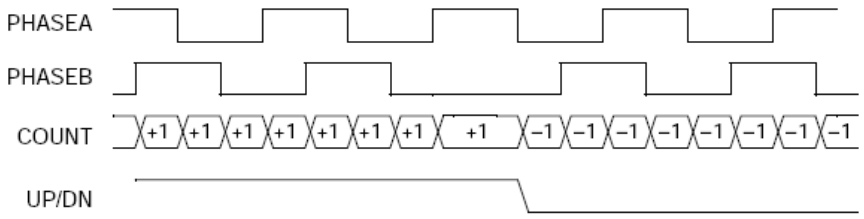

Figure 3: Quadrature Decoder signals

The main function of a QD periphery is a position counter and checking if the current value isn't out of bounds saved in registers. Other functions are difference counting and rotation counting. PE implements a QD periphery in its bean Quadrature Decoder. It offers all necessary functions for parameter setting and value reading - GetPosition, GetDifference, SetPosition, GetRevolution, etc.

A QD generates two interrupts - when index signal is changed which indicates next rotation and when home signal is changed which indicates that it is in its home position. Processing of both interrupts is supported by PE as events. PE prepares empty interrupt service routines for each used interrupt and a user has only to write his own code inside.

4 Simulation of MCU hardware peripherals

The main goal of our project is generating source code from Simulink model by RTW and PE tools. Our second goal is to allow simulation of these models and therefore each supported hardware periphery must have described its behaviour in a simulation model.

A simulation of the hardware peripherals is not a simply issue in Matlab/Simulink because it works partially asynchronously and Simulink simulates mainly synchronous systems or more precisely there is no way how hardware asynchronous events (and also interrupts) can be correctly simulated. The next difficulty is with peripherals which provide different functions and it is necessary to use each function in different time domain or unrepeatedly.

We have proposed and implemented the following methodology of simulation of HW peripherals. Every periphery is represented by one Simulink block. The block outputs are generated normally in the "calculate outputs" function in the corresponding S-function as shown in figure 4. For example the current values of position and rotation counters are such outputs of the QD periphery. Block input and output data types are exactly the same as registers of the periphery where to (or where from) values are written (or read). Inputs and outputs which

represent external inputs and outputs (e.g. analog input for ADC periphery) have dynamically determined data type which allows to interconnect with any blocks.

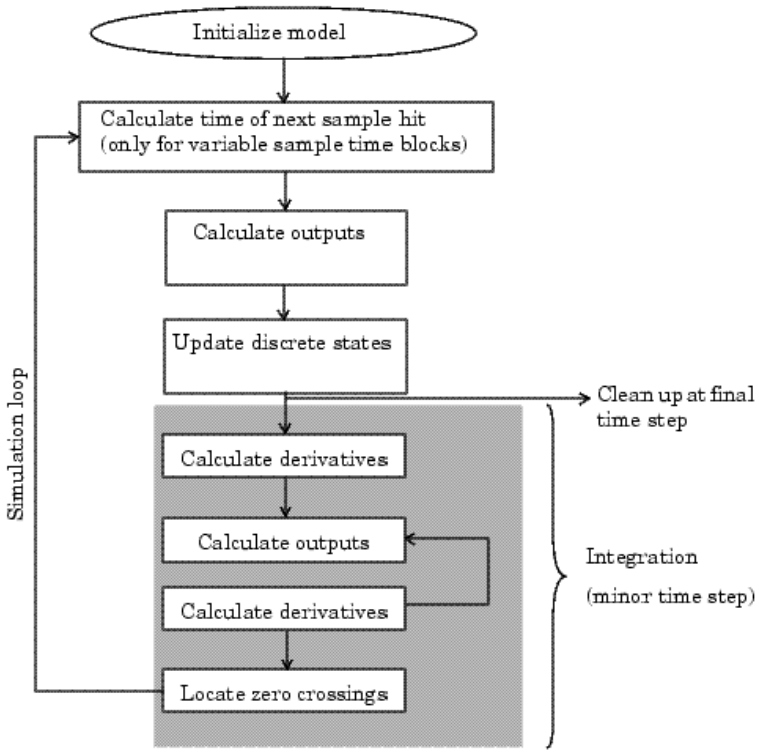

Figure 4: How Simulink Performs Simulation [8]

If a periphery invokes any interrupts they are represented as function-call signals outgoing from the corresponding block. A function-call subsystem which is connected to such function-call output represents interrupt service routine in these cases (figure 5). The function-call outputs can be excited only in the "calculate outputs" phase of block simulation.

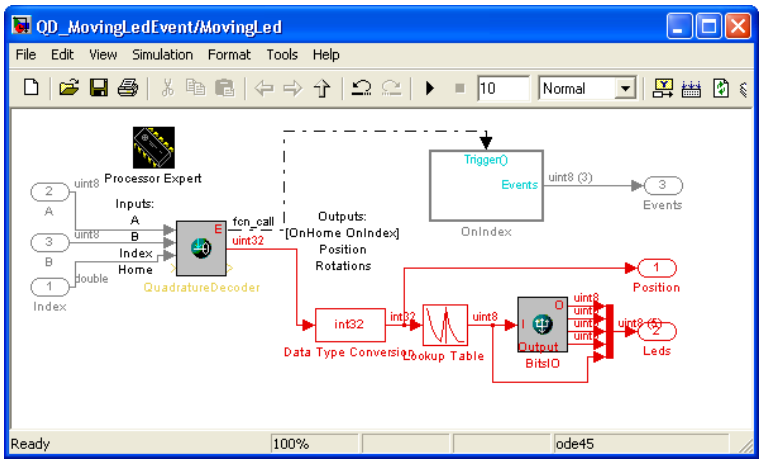

Figure 5: Model with a Quadrature Decoder periphery with interrupts

A choice of block sample time is one of crucial factors for the correct behavioural simulation of a periphery and for higher simulation speed. Because we simulate only behaviour of peripherals we couldn't implement delays between inputs and outputs. Then peripherals without internal timer (GPIO,ADC,QD,...) are only dependent on sample time of input signals and they use port-based timing; peripherals with internal timer (TimerInt, PWM, ...) use fixed timer with period sets to minimal time when the outputs are not changed or the variable sample time solver must be used, e.g. a PWM periphery generates PWM signal according to 16-bit value of

ratio and then corresponding block has sample time set to PWM period divided by 65536.

There is a special block in the blockset - Method block. It represents external functions for all peripherals. For example the block method can represents the following QD periphery functions: GetPosition, GetDifference, GetRevolution, GetHome, GetIndex, SetPosition, SetRevolution. The method block isn't directly connected to the main block and therefore it can be placed anywhere and also with different sample time in the model. Practically the method block is connected with the periphery main block through shared data structure which is persistant during model simulation. In this way, methods can get and change current values and parameters of the peripheral blocks and access to the blocks is performed in "calculate outputs" phase in method block. This solution hasn't any problems with reading current values and parameters from the block but if the method block changes current values or parameters of the block which directly affect block outputs the changes will be used in the first next sample time of the block. This is a disadvantage of simulation in Matlab/Simulink which must know time of the next sample time. This feature isn't so important in the behaviour simulation of hardware peripherals. And it can be partially compensated in some cases by using time-dependent computation of outputs against direct computation because time of changes can be saved from method block and the time is the same as the block time.

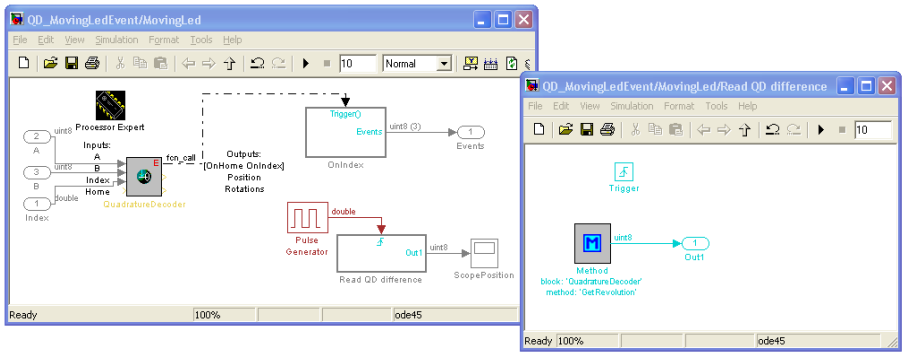

Figure 6: Submodel with QD block and an external QD method in different time domain

The method block has been added to the blockset mainly for inserting additional peripheral functions to the output generated code but it can simulate implemented peripheral functions as well and it can be helpful for simulation and code generation of unrepeated states, e.g. for initialization and in interrupts and generally different time domains.

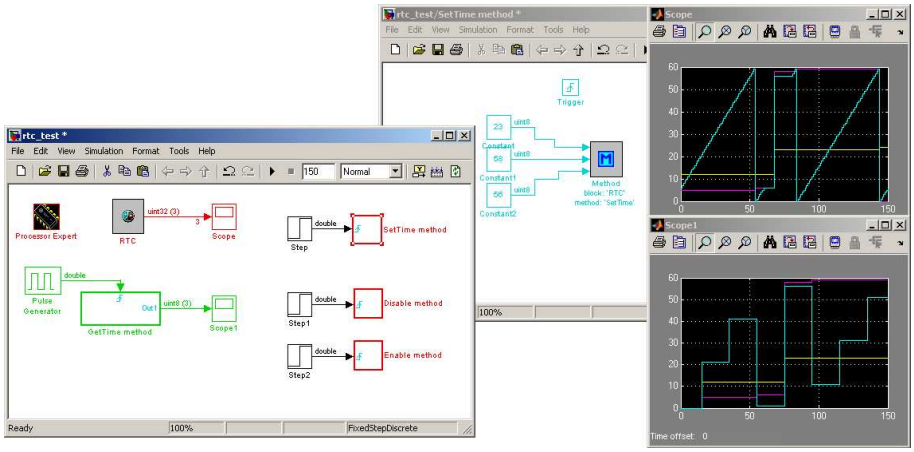

Figure 7: Model with RTC block and several external methods

5 Generated code by RTW

In our project we generate output C code from model with Real-Time Workshop and project specific target based on Embedded Real-Time Coder [7]. In this target, each block which represents hardware periphery is replaced by predefined code using the API from PE for corresponding bean and the code performs the selected main function of the periphery (e.g. generating PWM signals for PWM blocks). All method blocks are replaced with calling of the selected function corresponding to implemented simulation behaviour.

6 Conclusions and future work

The paper describes one possible way how hardware peripherals can be simulated. External standalone blocks are used for simulation of peripheral functions which must run in different subsytem or with different time domain. These blocks called Methods share data structure with main blocks and they can get or change values and parameters of main blocks. S-functions must be implemented with consideration to fact that external blocks can change block properties in anytime and not only in "calculate outputs" phase of block simulation.

Simulation with method blocks has been developed for project SESA which integrates two tools - Matlab/Simulink for simulation of systems and code generations from them and Processor Expert for preparing a hardware abstraction layer of embedded systems and supported MCUs. The method blocks are directly replaced by calling functions corresponding to simulated behaviour.

The future work will be focused on cleaning of the implementation and verification of the used metodology.

7 Acknowledge

This work is supported by Academy of Sciences of the Czech Republic under project No. 1ET400750406 (SESA project).

References

[1] R. Bartosinski, P. Stružka, L. Waszniowski. Peert-blockset for Processor Expert and Matlab/Simulink integration. In B. Walden C. Moler, A. Prochzka, editor, *Technical Computing Prague 2005 : 13th Annual Conference Proceedings*, pages 1–8, November 2005.

[2] R. Bartosinski, Z. Hanzálek, P. Stružka, L. Waszniowski. Processor Expert Enhances Matlab Simulink Facilities for Embedded Software Rapid Development. In *ETFA 2006 Proceedings*, pages 625–628, Piscataway, 2006. IEEE.

[3] R. Bartosinski, Z. Hanzálek, P. Stružka, L. Waszniowski. Integrated Environment for Embedded Control Systems Design. In *IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2007.

[4] UNIS s r.o. *Processor Expert help*, 2007.

[5] Freescale Semiconductor. *56F8300 Data Sheet, 16-bit Digital Signal Controllers*, 2007.

[6] Freescale Semiconductor. *56F8300 Peripheral User Manual*, 2007.

[7] Inc. The MathWorks. *Real-Time Workshop Users Guide*, 2007.

[8] Inc. The MathWorks. *Simulink - Writing S-functions*, 2007.

Roman Bartosinski
Department of Signal Processing,
Institute of Information Theory and Automation,
Czech Academy of Sciences,
Pod vodárenskou věží 4, 18208 Praha 8, Czech Republic
Email: bartosr@utia.cas.cz
www: sp.utia.cz

Jiří Kadlec
Department of Signal Processing,
Institute of Information Theory and Automation,
Czech Academy of Sciences,
Pod vodárenskou věží 4, 18208 Praha 8, Czech Republic
Email: kadlec@utia.cas.cz
www: sp.utia.cz