MODEL-BASED DESIGN OF EMBEDDED SYSTEMS USING MATHWORKS TOOLS – A CASE STUDY

Laura Fábián[1], Gergely Somlay[1,2], János Márkus[1,2]

[1] Budapest University of Technology and Economics,
Faculty of Electrical Engineering and Informatics,
1117 Budapest, Magyar tudósok körútja 2.
[2] Gamax Ltd.
1114, Budapest, Bartók B. u. 15/d

gergelys@gmail.com (Gergely Somlay)

Abstract

A model-based development environment for simulation and automatic code generation had been developed using the MathWorks' products *Simulink* and *Real-Time Workshop Embedded Coder*. The main goal of the work was to create an environment for seamless integration of simulation and real-time operation. The simulation target is the MITMOT, a modular platform consisting of a 32-bit ARM-based microcontroller and some custom I/O peripherial developed at the Dept. of Measurement and Information Systems of the Budapest University of Technology and Economics.
The eCOS embedded real-time operating system has a MITMOT ARM target specific version. In this project, our aim was to present an easily usable graphical modeling environment which can be used to model the MITMOT target, and the C code automatically generated from the model would run under eCOS.
The development consisted two main parts: the first was to create models of the different I/O units in the Simulink environment using S-functions, while the second was to write the TLC (target language compiler) files which are responsible for the C-code generation. With our tool the time to develop embedded software for the MITMOT target has been reasonably reduced, and since the environment is fully graphical, the C programming skill requirements for software development to this target became unnecessary.

Keywords: Model-Based Design, Simulink - Real-Time Workshop, code generation, modeling embedded system, eCOS.

Presenting Author's biography

Laura Fábián was born in Budapest, Hungary, in 1983. She received the M.S. degree in technical informatics from the Budapest University of Technology and Economics (BUTE), Budapest, Hungary, in 2007. Currently, she is a Software Developer at Ericsson Hungary Ltd. Her research interests are in the areas of embedded systems and sensor networks.

1 Introduction

As systems become more and more complex and development time is getting shorter due to the pressure of time-to-market, engineers have to search for efficient development techniques, in which errors and design flaws are catched early, reusability of modules is easy and the time of moving from prototypes to final hardware is short.

Model-Based Design (MBD) is a technique, which lets engineers use state-of-the-art software development methods for rapid prototyping [1,2]. The basic principle of MBD is that instead of using paper-based requirements, followed by physical prototypes and final target development, the focus is on the model of the system to be developed. This model can be used throughout the development, from specification to final implementation. The system-level model serves as an executable specification, which is continuously refined through the development phase. The final code is created by automatic code generation. The main advantage of MBD is that continuous test and verification of the requirements is possible by simulation throughout the complete design process.

The Mathworks, Inc. provides the Simulink product as a graphical simulation tool for MBD. Simulink is an excellent interactive graphical environment for a high level description of embedded algorithms and additionally, with add-on products, the Simulink environment is also capable of automatic code generation. To accelerate embedded system development, there are blocksets available for specific targets, such as Infineon C166, TI C6000 [1].

In this paper the design procedure of a Simulink blockset library for a new embedded real-time target is presented. The developed Simulink blockset library can be used for target-specific simulation and code generation purposes. A similar solution, based on a Texas Instruments Digital Signal Processor has been proposed in [3].

1.1 The Simulink Environment

The Simulink software [4] is a graphical simulation tool integrated into MATLAB [5], which is a numerical environment for algorithm development, data visualization, data analysis and numerical analysis. The Simulink environment is well known for system engineers designing e.g., communication systems, signal processing and control algorithms.

The Real-Time Workshop (RTW) [6] is an extension tool for Simulink. Using Real-Time Workshop, it is possible to generate ANSI/ISO C code from a Simulink model. In Simulink, the set of blocks used for simulation can be extended with user-specified ones, based on Simulink S-functions.

An extension of the RTW is the Real-Time Workshop Embedded Coder (RTW-EC) [7], which is specially designed for the needs of embedded system design. It is capable of highly optimized and efficient C-code generation, according to the limited resources of an embedded system.

1.2 The MITMOT Target

The MITMOT system, shown in Fig. 1, is a “mote”, i.e., a small embedded system developed at the Budapest University of Technology and Economics, by the Department of Measurement and Information Systems, mainly for educational purposes [8]. It is a modular architecture consisting of a controller card and several cards for measurement, display and communication. The controller card is available with two different processors depending on the application’s requirement: one consists of an 8-bit Atmel AVR microcontroller, while the other is based on an ARM7TDMI 32bit Phillips LPC2000 series microcontroller.

An earlier attempt for creating a Simulink target library targeting the 8-bit AVR processor card for code generating purposes had been introduced at the department in 2005 [13]. The current work builds on that previous one, but the whole system had been reimplemented and targeted for the 32-bit ARM processor, the hardware simulation has been implemented as well and the generated code is interfacing with the eCOS real-time operating system running on the processor.

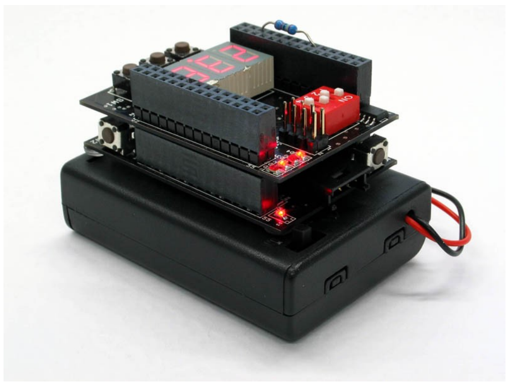


Fig. 1 The modular MITMOT system [8].

For the MITMOT ARM7 processor and display module, an API (Application Programming Interface) is defined in C, which makes the code generation process much easier. In the development this API was used, thus the functions of the display card could be accessed via simple C function calls (e.g., the actual temperature can be read by a simple command).

2 Development of the tool

The main motivation of the development of the tool was (i) to be able to simulate the hardware behavior of the system in the Simulink environment and (ii) to be able to generate C-code from the Simulink environment without using any additional tool. For these purposes we used *MATLAB-Simulink* and the *Real-Time Workshop Embedded Coder.* The environment currently supports single tasking, single-rate systems.

To define user specific blocks, two types of code is needed: the first one specifies the behavior of the block during Simulink simulation, the second one (the block target file) specifies the behavior of the block after automatic code generation. This latter one is written in the Target Language Compiler (TLC) language. These two codes are not tightly linked to each other, so they can be developed separately, thus the differences between the simulation environment and the real world can be taken into consideration: e.g., a thermometer in the real target provides real measurements, while in a simulation environment it should provide predefined data-set at its output.

To create a new target for the Simulink environment, several files have to be developed:

- Graphical model (block) of each I/O unit;
- Compiled files of the Simulink blocks for simulation;
- Source code of the blocks (S-functions);
- TLC files to control the code generation of each block;
- TLC files for controlling code generation and for interfacing with the actual hardware and the real-time operating system.

In the following two subsections the S-functions and TLC-files are described briefly.

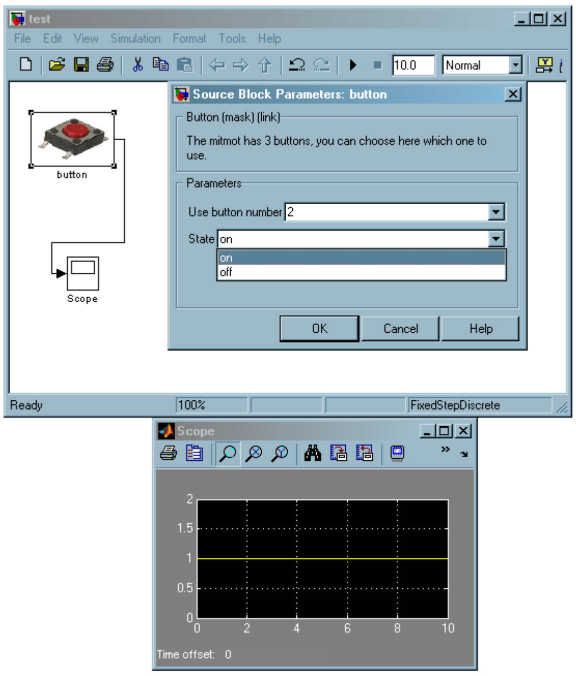
2.1 Device drivers under Simulink: S-functions

Simulink gives the possibility of the implementation of a device driver for simulation purposes with S-functions (System functions) [9-11]. The S-function, which can be written in C, C++, Ada, MATLAB or Fortran programming languages, is the base of every Simulink block and its specification is available for the user to create customized ones.

For creating the MITMOT-specific Simulink blocks the S-functions were written in C. The C code is compiled by MATLAB using the *mex* command to MEX format, which is actually a DLL-file under Microsoft Windows platform.

The S-function's C code specifies the actual behavior of the block during simulation, and it is called by the Simulink solver engine in every simulated time-step. The structure of such function is predefined, and the required and optional function calls (entry points for the generated DLL) can be developed by using template S-function files.

The blocks' graphical appearance is fully customizable using the masking function. A parameter dialog box can also be assigned for each block which enables user interaction. As an example, Fig. 2 shows a push button model in the Simulink environment, its parameter window (consisting of two parameters: (i) which physical push button is used and (ii) what is its actual state) and the simulated output, which is logical one in this case. Fig. 3 shows part of the S-function written for the simulated push-button.



Source Block Parameters: button

Button (mask) (link)

The mitmot has 3 buttons, you can choose here which one to use.

Parameters

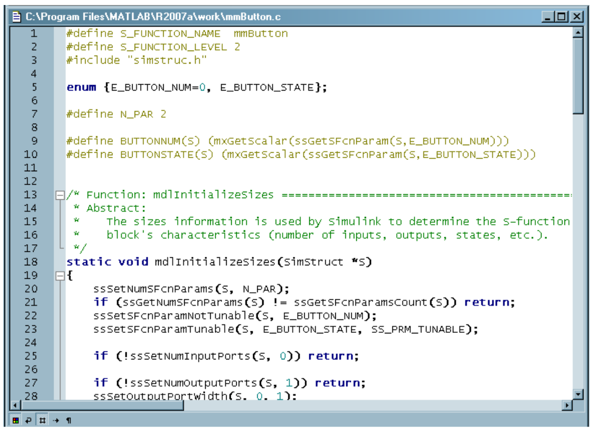
Use button number 2

State on

on

off

Fig. 2 The block and parameter window of the push button hardware model in the Simulink environment.


C:\Program Files\MATLAB\R2007a\work\mmButton.c

```c
#define S_FUNCTION_NAME  mmButton
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"

enum {E_BUTTON_NUM=0, E_BUTTON_STATE};

#define N_PAR 2

#define BUTTONNUM(S) (mxGetScalar(ssGetSFcnParam(S,E_BUTTON_NUM)))
#define BUTTONSTATE(S) (mxGetScalar(ssGetSFcnParam(S,E_BUTTON_STATE)))


/* Function: mdlInitializeSizes ==========================================
 * Abstract:
 *    The sizes information is used by Simulink to determine the S-function
 *    block's characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, N_PAR);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) return;
    ssSetSFcnParamNotTunable(S, E_BUTTON_NUM);
    ssSetSFcnParamTunable(S, E_BUTTON_STATE, SS_PRM_TUNABLE);

    if (!ssSetNumInputPorts(S, 0)) return;

    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);
```

Fig. 3 The S-function of the push button.

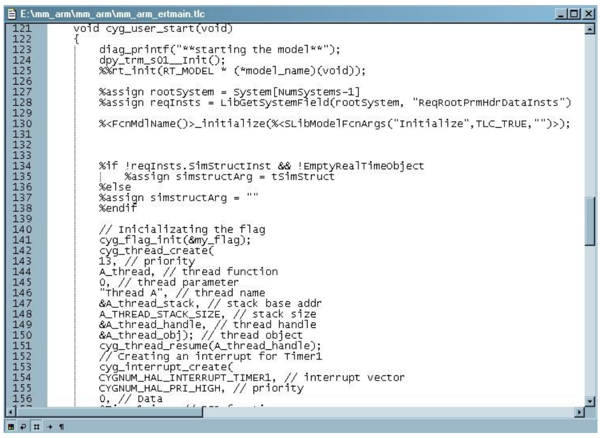
For the simulation environment the implementation of 5 S-functions were needed, each of them representing one of the MITMOT's peripheries: LED, pushbutton, three-digit seven-segment display, switch and thermometer. Like in real use, the pushbutton and switch block's state (on/off) can be tuned during the simulation [14].

2.2 Code generation procedure: the TLC language

The Target Language Compiler (TLC) [12] is responsible for the code generation process. For each Simulink block a TLC file had to be written, which file is used to customize the generated code for the specified block. In addition, another TLC file, the System Target File (STF) had to be specified, which is responsible for the overall control of the code generation stage of the build process. It provides definitions of variables that are fundamental to the build process, e.g., code format to be generated.

When a Simulink model contains an S-function and a corresponding TLC block target file also exists for that S-function, Real-Time Workshop *inlines* the S-function. Inlining means that during build-time the TLC-file is executed instead of setting up function calls for the S-function C-file. Such an inlining can produce more efficient code by eliminating the S-function API layer from the generated code. In our case, inlining the S-functions were especially useful because the device driver I/O S-functions read from the MATLAB workspace during simulation, but in the generated code they read from an actual hardware address [6].

For the MITMOT-eCOS proper C code generation, nine TLC files were developed, from which four assures the generation of the overall eCOS specific features (e.g., the eCOS main function syntax, and includes) and the remaining five TLC files represent the code for the MITMOT I/O units. Fig. 4 shows a part of the main TLC code.



```
void cyg_user_start(void)
{
    diag_printf("**starting the model**");
    dpy_trm_s01__Init();
    %%rt_init(RT_MODEL * (*model_name)(void));

    %assign rootSystem = System[NumSystems-1]
    %assign reqInsts = LibGetSystemField(rootSystem, "ReqRootPrmHdrDataInsts")

    %<FcnMdlName()>_initialize(%<SLibModelFcnArgs("Initialize",TLC_TRUE,"")>);



    %if !reqInsts.SimStructInst && !EmptyRealTimeObject
        %assign simstructArg = tSimStruct
    %else
    %assign simstructArg = ""
    %endif

    // Inicializating the flag
    cyg_flag_init(&my_flag);
    cyg_thread_create(
    13, // priority
    A_thread, // thread function
    0, // thread parameter
    "Thread A", // thread name
    &A_thread_stack, // stack base addr
    A_THREAD_STACK_SIZE, // stack size
    &A_thread_handle, // thread handle
    &A_thread_obj); // thread object
    cyg_thread_resume(A_thread_handle);
    // Creating an interrupt for Timer1
    cyg_interrupt_create(
    CYGNUM_HAL_INTERRUPT_TIMER1, // interrupt vector
    CYGNUM_HAL_PRI_HIGH, // priority
    0, // Data
```

Fig. 4 A MITMOT ARM-eCOS TLC code.

The generated code is working with a simple periodical interrupt scheme, using one of the ARM processor's timer unit for the scheduled interrupt. The generated code's lifecycle is infinite, after an initialization phase starts a loop, where each simulation step is represented by the occurrence of an interrupt [14].

3 Application Example

The designed and tested application is based on the functionality provided by the display card, which is a simple man-machine interface. On the display card there are four red LED's, a three-digit seven-segment display, three pushbuttons, four switches and a thermometer IC communicating via I2C bus.

For each device a Simulink block was designed with parameter dialog boxes for easy user interaction – the user can control the state of the I/O's. Fig. 5 shows the developed target block library.

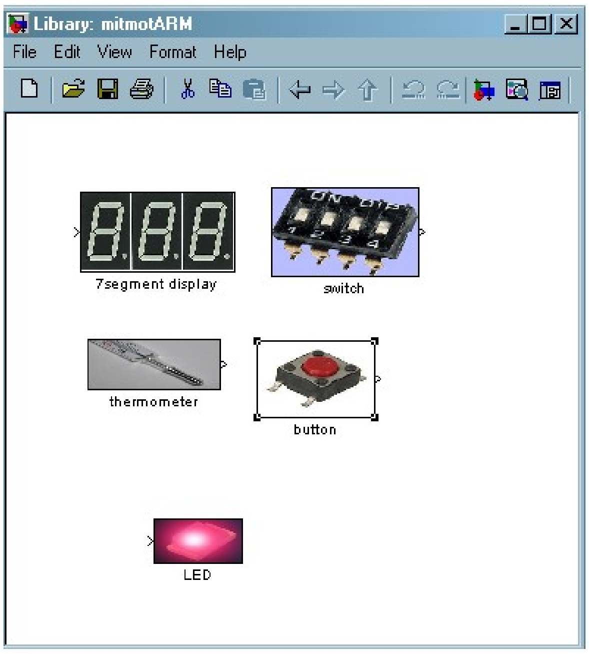


Fig. 5 The device driver block library in the Simulink environment.

For the demonstration of the code generating environment a simple application was developed [14], which realizes a stopwatch-like function. The application's functions are:

- The numbers from 0 to 9 are running cyclically on the display with a given refreshing frequency.
- The button #2 is a reset button, which sets the counter's value to 0.
- Switching on the switch #2 the actual value can be represented in binary format on the LEDs.
- Counting can be stopped and restarted with the switch #1.

Fig. 6 shows the Simulink model of this application example.

Besides the elements of the mitmotARM library, several native Simulink blocks have been used in this model, like the switch logic, the multiplexer and the constant block.

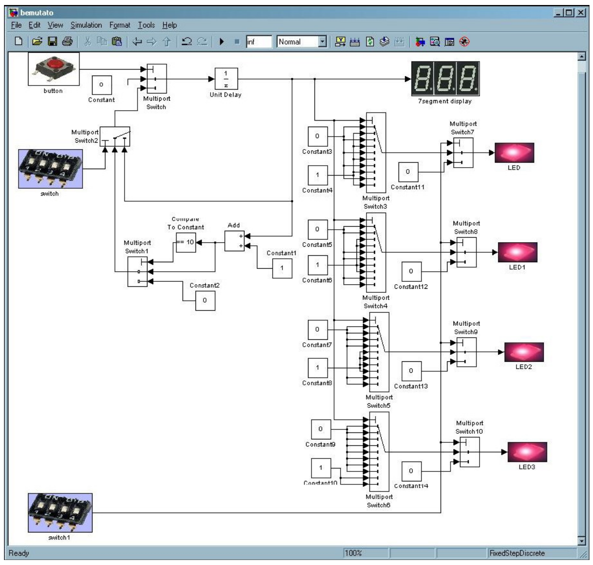


Fig. 6 The Simulink model of the application.

The model's behavior during simulation can be verified by inserting display blocks. With display and scopes blocks the actual value of the outputs can be visualized and checked as shown in Fig. 7.

As in the figure, the simulation behavior fulfills the expectations.

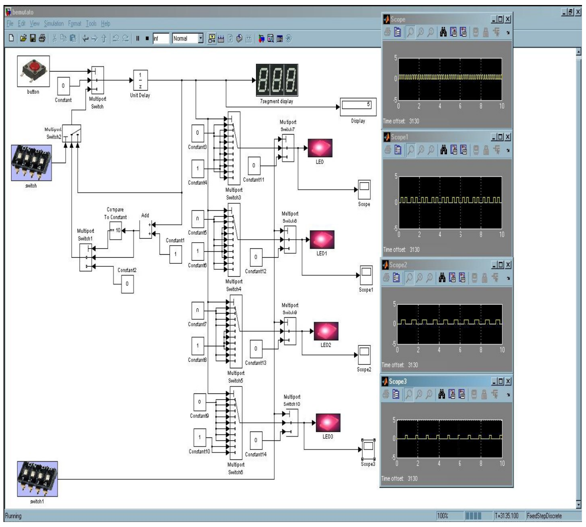
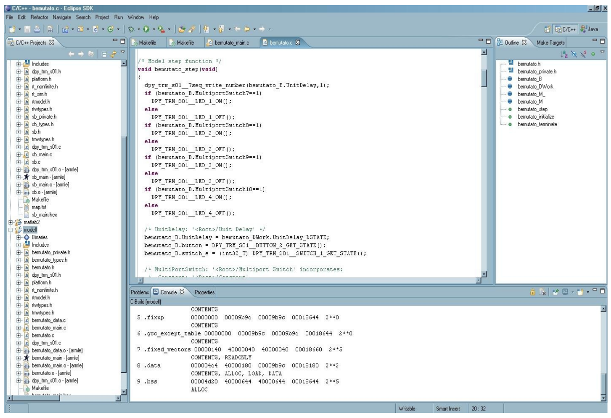


Fig. 7 The simulation of the model.

From this Simulink model 3 C- and 4 H files were generated by the Real-Time Workshop Embedded Coder. These C- and H files had to be compiled for the MITMOT ARM module to hex file format, which compilation procedure can be done with the use of the MITMOT ARM specific Eclipse-eCOS environment, as shown in Fig. 8.



```
/* Model step function */
void bemutato_step(void)
{
  dpy_trm_s01__7seq_write_number(bemutato_B.UnitDelay,1);
  if (bemutato_B.MultiportSwitch7==1)
    DPY_TRM_S01__LED_1_ON();
  else
    DPY_TRM_S01__LED_1_OFF();
  if (bemutato_B.MultiportSwitch8==1)
    DPY_TRM_S01__LED_2_ON();
  else
    DPY_TRM_S01__LED_2_OFF();
  if (bemutato_B.MultiportSwitch9==1)
    DPY_TRM_S01__LED_3_ON();
  else
    DPY_TRM_S01__LED_3_OFF();
  if (bemutato_B.MultiportSwitch10==1)
    DPY_TRM_S01__LED_4_ON();
  else
    DPY_TRM_S01__LED_4_OFF();

  /* UnitDelay: '<Root>/Unit Delay' */
  bemutato_B.UnitDelay = bemutato_DWork.UnitDelay_DSTATE;
  bemutato_B.button = DPY_TRM_S01__BUTTON_2_GET_STATE();
  bemutato_B.switch_e = (int32_T) DPY_TRM_S01__SWITCH_1_GET_STATE();

  /* MultiPortSwitch: '<Root>/Multiport Switch' incorporates:
```

```
C-Build [modell]
                 CONTENTS
 5 .fixup        00000000  00009b9c  00009b9c  00018644  2**0
                 CONTENTS
 6 .gcc_except_table 00000000  00009b9c  00009b9c  00018644  2**0
                 CONTENTS
 7 .fixed_vectors 00000140  40000040  40000040  00018660  2**5
                 CONTENTS, READONLY
 8 .data         000004c4  40000180  00009b9c  00018180  2**2
                 CONTENTS, ALLOC, LOAD, DATA
 9 .bss          00004d20  40000644  40000644  00018644  2**5
                 ALLOC
```

Fig. 8 The MATLAB generated code for the example applicaion in the Eclipse environment.

When the code is in hex format, it can be downloaded to the target processor. The automatically generated code from the Simulink model worked properly, as shown in Fig. 9.

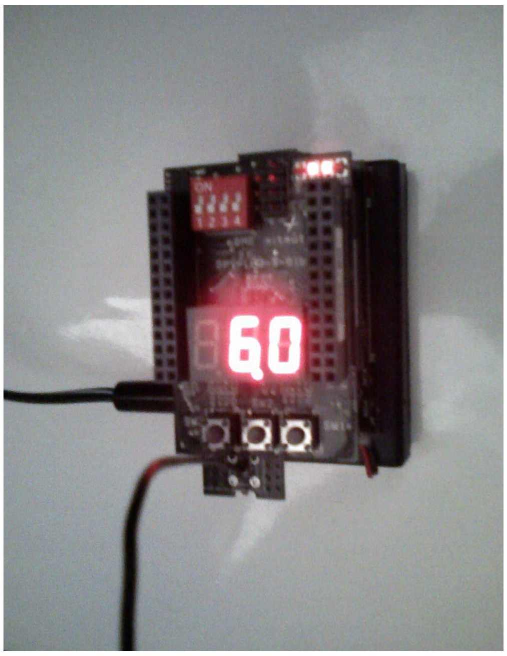


Fig. 9 MITMOT with the running application.

After the final compilation, the presented application takes 113 kB in hex format.

4 Conclusion

This paper presented the design of a simulation and code generation environment for the MITMOT target using tools from The Mathworks, Inc. In the Simulink environment a block library was developed for representing the sensor cards' basic I/O interfaces. In this paper the functionality of an application example using the MITMOT ARM Simulink library blocks has been demonstrated. With this code generation and simulation environment it became possible to simulate the behavior of the simple sensors and I/O interfaces found on the MITMOT modules and from the Simulink model eCOS-compatible C code can be automatically generated. Hence, the application development for the target moved from C code to the graphical Simulink language, and new applications can be easily developed without knowing the actual hardware architecture or the main principles of the eCOS operating system.

5 Acknowledgement

This work has been supported by Gamax Ltd. (http://matlab.gamax.hu), a Hungarian company, official distributor of the products of The Mathworks, Inc. in south-east Europe.

6 References

[1] The MathWorks, Inc, Matlab and Simulink: the platform for Model-Based Design, 2007, www.mathworks.com/mbd.

[2] G. Reed.: "Model-based design aids test and verification", Test & Measurement World, 12/1/2006, http://www.reed-electronics.com/tmworld/article/CA6401653.html?industryid=21385.

[3] Hercog, D.; Curkovic, M.; Edelbaher, G.; Urlep, E.: "Programming of the DSP2 board with the Matlab/Simulink", Industrial Technology, 2003 IEEE International Conference on Volume 2, Issue , 10-12 Dec. 2003 Page(s): 709 - 713 Vol.2.

[4] The MathWorks, Inc.: Simulink® 6 Using Simulink (sl_using.pdf), version 6.6, March 2007, www.mathworks.com.

[5] The MathWorks, Inc.: Matlab® 7 Desktop Tool and Development Environment (matlab_env.pdf), version 7.4, March 2007, www.mathworks.com.

[6] The MathWorks, Inc.: Real-Time Workshop® 6 User's Guide (rtw_ug.pdf), version 6.6, March 2007, www.mathworks.com.

[7] The MathWorks, Inc.: Real Time Workshop Embedded Coder 4 User's Guide (ecodder_ug.pdf), version 4.6, March 2007, www.mathworks.com.

[8] Cs. Tóth et al: The modular MITMOT system, technical report, 2006. http://bri.mit.bme.hu/?l=mitmot&p=what%20is.

[9] The MathWorks, Inc.: Real-Time Workshop® Embedded Coder 4 Developing Embedded Targets (ecoder_det.pdf), version 4.6, March 2007, www.mathworks.com.

[10] The MathWorks, Inc.: Simulink® 6 Writing S-Functions (sfunctions.pdf), version 6.6, March 2007, ww.mathworks.com.

[11] The MathWorks, Inc.: Real-Time Workshop® Embedded Coder 4 Reference (ecoder_ref.pdf), version 4.6, March 2007, www.mathworks.com.

[12] The MathWorks, Inc.: Real-Time Workshop® 6 Target Language Compiler (rtw_tlc.pdf),version 6.6, March 2007, www.mathworks.com.

[13] V. Huszar: Rapid Prototyping Using MATLAB-Simulink and MITMOT, Student's Scientific Contest, Budapest University of Technology and Economics, Dept. of Measurement and Information Systems, Nov., 2005.

[14] L. Fabian: Automatic code generation to MITMOT ARM target using the MATLAB Simulink environment, Master's Thesis, Budapest University of Technology and Economics, Dept. of Measurement and Information Systems, May, 2007.