Creating support for fully automatic code generation for Cerebot MX7cK hardware from Simulink environment

V. Lamberský, J. Vejlupek, V. Sova, R. Grepl

***Abstract*—**When starting a new project, a higher programming language is usually selected for developing the control algorithm. And it is only natural to have support for automatic code generation, which would provide functionality for generating executable code for embedded processor directly from the high programming language, where the algorithm was originally developed. Currently there are several products on the market which implement described functionality. However this functionality is very limited. It is provided only for selected microcontrollers and simple peripheral modules. Increasing computing power of embedded processors allows implementing more advanced algorithms and to use more complex peripherals, for instance displaying units. This paper presents a method of creating support for fully automatic code generation for Cerebot MX7cK hardware from Simulink. This target uses complex peripherals, which are not supported in this extent by any other commercial product. Created support for automatic code generation is demonstrated by generating executable code for magnetic levitation plant controller, directly from Simulink.

***Keywords*—**Cerebot MX7cK, automatic code generation, Simulink, complex peripherals

I. Introduction

EACH day, there is always increasing need to develop a new products, with always increasing complexity and higher function requirements. The need to shorten and make more efficient development cycle of new applications is obvious.

Over period of time, the V-cycle becomes a standard scheme which describes relations between various stages during development of a new application [1]. This scheme is illustrated in Fig. 1.

This work was supported in part by the BUT grant FSI-J-13-2026 “Development of support for automatic code generation for the Cerebot MX7 cK platform”

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

J. Vejlupek is a PhD candidate at the Brno University of Technology – Faculty of Mechanical Engineering. His current research focuses on Hardware in the Loop simulation, Rapid Control Prototyping and Rapid Code Generation for embedded applications.

V. Sova is a PhD candidate at the Brno University of Technology – Faculty of Mechanical Engineering. . His current research focuses on advanced BLDC motor control algorithms and Rapid Code Generation for embedded applications.

R. Grepl is associate professor at the Brno University of Technology – Faculty of Mechanical Engineering.

Introducing fully automatic code generation during the development cycle has several benefits, besides shortening the development cycle. It helps reduce errors which would be inevitable when manually rewriting the code from higher programming language to C language. And making changes in project definitions is much easier as small change in model does not require manual rewriting several lines of code. Thus, higher programming languages are becoming preferred over low level ones [2], [3] in embedded design applications.

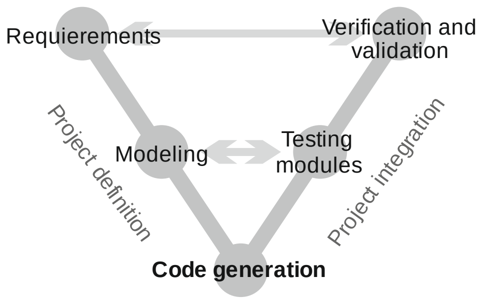


Fig. 1 Automatic code generation in context of development cycle

Several products support direct code generation from higher programming language for limited number of embedded processors and selected peripheral modules. However, always increasing power and decreasing price of embedded processors [4] allows us implement more advanced algorithms and use complex peripherals where it was not reliable before. And most of these new targets and almost all advanced peripheral modules are not supported in higher programming languages for direct code generation in commercial products.

One of these unsupported targets for automated code generation is a Cerebot MX7cK hardware, which has several unique properties among other products. It is becoming very popular mainly for its very low price and fairly high computing power (it is equipped with the fastest 32bit Microchip MCU). Beside these properties, Cerebot introduces a new modular standard for embedded rapid prototyping hardware [5], [6].

Cerebot presents a standardized way to interface with peripheral modules. That means a peripheral module can be easily connected to a connector on board without the need to solder wires between microcontroller pins and peripheral module. A wide range of different peripheral modules available for the Cerebot hardware makes this platform very versatile. The concept of detachable peripheral modules is illustrated in Fig. 2.

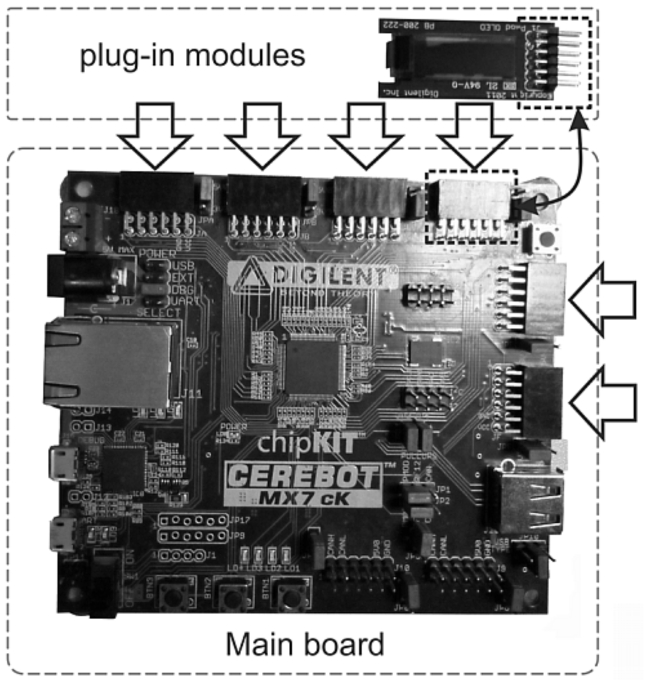


Fig. 2 Cerebot main board with plug in module

This concept of unified interface with peripheral modules places Cerebot hardware in superior position in its hardware category.

Based on previously described advantages of Cerebot platform, and the benefits which would present an option to generate executable code directly from higher programming language, it was decided to create a toolset which would support direct code generation. The created blockset for Simulink and supporting programs enables fully automatic generating of executable code directly from Simulink environment for Cerebot hardware and selected peripheral modules.

Created blockset presents several benefits over existing ones. It was created as a freeware; therefore anyone can use and modify the files. This can be very beneficial, especially when some functionality is not implemented. A small modification in source codes can easily change generated code behavior. Compared to commercial products, where source code is not available, when some functionality is not available user usually has to write its own block from scratch, which takes much longer time compared to modifying an existing one.

Beside this, the Cerebot platform can be now used much more efficiently as designing a new application in higher programming language is much faster. Therefore more time can be spent on algorithm development and less on algorithm coding. This hardware can be now used even with people with limited or none knowledge of C language as the program can be generated entirely from Simulink without any need to further modify generated C files.

Main benefits of created tools are demonstrated on a particular model situation. A code for magnetic levitation controller was designed completely from Simulink environment. The levitation of steel ball is maintained by magnetic field, which is controlled by changing the current running through the coil to maintain the ball in selected position. This plant is illustrated in Fig. 3

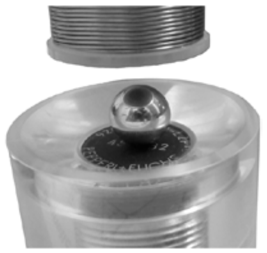


Fig. 3 Scheme of magnetic levitation plant

This plant is known to be very unstable, however can be controlled with a fast enough PID controller. The last chapter of this paper demonstrates how significantly can be shortened time needed to design a controller for this plant when using tools for automatic code generation which were created.

II. TODAY'S TOOLS USED FOR AUTOMATIC CODE GENERATION TARGETED AT EMBEDDED APPLICATION

This section presents an overview of available embedded platforms and software products. Further it explains mechanisms used for translating code from higher level programming language to executable code.

A. Generating a code from higher language

When generating code from higher programming language two main problems have to be encountered. First, the higher programming languages were developed for simulations on a personal computers and therefore do not generate code which is very efficient (or optimized for low power CPU). The second one are target specific functions. When running simulation on a processor without an OS, on each processor type, functions for interfacing peripherals are different.

In order to generate efficient C code for embedded processor from Simulink model, the model representation needs to be transformed. For instance, article [7] presents concepts for transforming code to a different language. There are several tools designed for this task. Official one is Embedded Coder (created by the Matworks company) and free alternatives, for instance the Gene-Auto [8]. But this freeware tool supports much lower number of Simulink block for code generation compared to Embedded Coder.

During code generation stage, the Simulink model represented in *RTW* record is translated into C code. In next step the generated C files are processed with target specific compiler and linked to generated executable file, which can be then directly loaded into the flash memory and run on the target hardware.

As mentioned before, Matworks provides a tool for translating the Simulink model into C code that can be efficiently run on a low power embedded processor. But it also contains a hooks and callback functions entries to execute user scripts and programs. These entries can be used to call a compiler and loader once all necessary C files are created. Therefore only one user action is required to generate the executable code and load it to the target hardware.

To achieve this type of functionality, two type files and scripts need to be created. The first group of scripts creates files used in Matlab/Simulink environment to control code generation process. The second group consists of programs and scripts running outside Simulink whose are used to translate generated files and load the generated executable code to a microcontroller.

Steps involved during the code generation process are illustrated in Fig. 4.

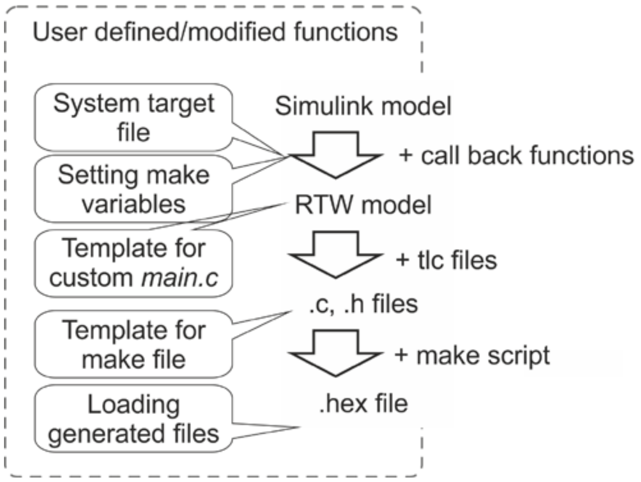


Fig. 4 User defined files used during various stages of code generation process

The entry point to code generation process from Simulink is invoking the make_rtw function (this is done when the build button is pressed in Simulink model). In first stage, Simulink model is prepared for code generation, no modifications of this function are necessary as this procedure only rewrites Simulink model representation (.mdl) to a *RTW* one.

In the second step the *RTW* file is translated using *TLC* templates to .c and .h files.

In order to compile generated files, the makefile template (.tmf) which controls creation of makefile to match used C compiler need to be modified. The makefile is automatically executed after its creation, so no specific actions are required when using default settings.

Hook functions are ideal for calling external user program or scripts during various stages of code generation process (e.g. before or after make command). This option is suitable to call external program to load generated binary file into microcontroller after compiling and linking is done.

The second problem when generating executable code from higher language are target specific functions. Particularly, the function that is implements scheduler module in code entry function (the main function) and is maintaining code execution. Beside this we need functions that are setting MCU peripherals and interfacing with them.

The Cerebot main file is created using file customization template. The main file contains macros for hardware initialization and a function, which is called periodically and is used to schedule execution of generated code in target hardware. The file customization template can be set from target tlc file, which is the first function called after code generation is started. Besides setting the target specific main file it is used to collect parameter needed during generating c files or building process by providing user interface (graphical dialogs) for setting user parameters affecting the generated code (*TLC* and *MAKE* variables). These variables are used to specify compiler settings, enable automatic load of generated code, etc.

The second group of target specific function (functions for peripheral handing) is added as a special block in Simulink model. Scheme of this concept is illustrated in Fig. 5.

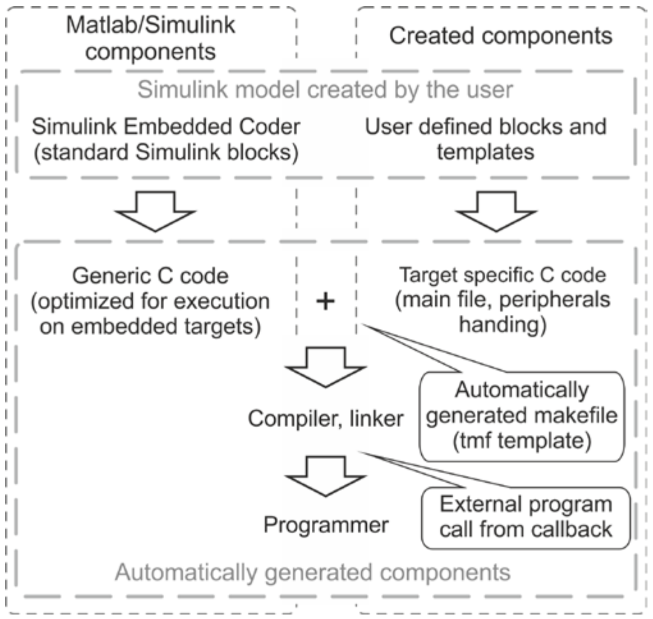

Matlab/Simulink components
Created components
Simulink model created by the user
Simulink Embedded Coder (standard Simulink blocks)
User defined blocks and templates
Generic C code (optimized for execution on embedded targets)
+
Target specific C code (main file, peripherals handing)
Automatically generated makefile (tmf template)
Compiler, linker
Programmer
External program call from callback
Automatically generated components

Fig. 5 Simulink model and components for code generation

Matlab/Simulink provides tools for creating blocks that can easily implement functions for peripheral handing; the key concept is that the block can provide two types of functionality for each Simulink block. One type is used during running the simulation and second one is used during code generation process.

The function used during simulation, referred as *MEX* function, can be implemented in several supported languages and translated with supported compiler into .mex or .mex64 (based on type of operating system being used) file which will be used during Simulink simulation. When developing the Cerebot blockset, these functions were programmed in C language.

The second group of created functions, which is used during code generation process, consists of scripts and functions in *TLC* language. This language is designed to process text files and it can place any expression at the point in code "instead of" the Simulink block. It can be any c language expression or function call.

B. Platforms supported for automatic code generation

Today, various targets are already supported up to various extend for automatic code generation from Simulink. Some targets have inherent support from products that are a part of Matworks Embedded Coder, support for other products can be added by purchasing a third party tools or developing them [9].

From category of blocksets for a 32-bits Microchip processors several commercial blocksets are available. Particularly, one blockset was created by Kerhuel [10] and second from a Microchip Company is being developed, currently support only 16 bit microprocessors [11]. However, these blocksets support only some build-in MCU peripherals. Beside this, in some cases a non-typical application can require peripheral functionality which is not supported by the available blocksets.

In cases where missing support from blocked does not allow fully automatic code generation from Simulink some options are available. Automatically generated code from Simulink can be imported into a hand written C project.

This process is usually referred as cogeneration. Matlab provides tools for simple creation of generic C code which is ready to be imported into C project where target specific functions (for handing peripherals) are written manually. This concept is further explained and demonstrated in [12]. This approach is suitable for applications with hardware which is not expected to be used in further projects.

For other applications creating a blockset is recommended approach. Although creating a new block set is quite complex task, when reusing created blocks in various designs the time saved when generating code directly from Simulink can compensate costs for developing new block set.

C. Cerebot platform

Cerebot MX7 cK target is equipped with one of the most powerful 32bit PIC microcontrollers available (PIC32MX795). This makes the board suitable for implementing fairly complex algorithms. Beside this it provides several very complex interfaces, which are not common in embedded applications, for instance Ethernet, or USB interface.

There is a wide range of various peripheral modules which can be purchased and used with Cerebot board. It contains various types plug-in modules suitable for mechatronics applications (temperature sensor, acceleration, etc.)

Cerebot has a superior position in its category of embedded rapid prototyping boards for its number of ports trough which extending peripheral modules can be connected. And for high number of various types of extension peripheral modules that are available for this platform.

For these special properties, we have decided to create support for Cerebot platform to enable option for automatic code generation directly from the Simulink model. This blockset will further increase the number of applications for this board as now this board can be used without any knowledge of low level programming languages and the software can be developed much faster.

One of the most complex peripherals which can be connected to a Cerebot board is display unit (reffered as OLED2 module). Although it can display only 16 shades of one color and has resolution of 64x256 pixels, it can display quite advanced graphic elements.

There is no similar blockset for any platform which would enable automatic code generation for this peripheral unit. As standard approaches using masked blocks or calling Matlab GUI for configuring block parameters does not provide feasible flexibility for modifying block which generates various types and quantity of functions with different parameters. Therefore, a special java application was developed to provide such functionality in Simulink model.

Some alternative solutions to this approach represent programmable display modules which can be purchased together with graphical programming environment. From its specialized environment the code can be automatically generated, however these tools can be used only with hardware, which they are designed for. Beside this, the price for autonomous Serial graphic displays which are produced by Electronic assembly is not very low (compared to displays equipped with simple driver).

III. Blockset for Cerebot

This section will describe developed Simulink blocks, programs and templates that are part of the Cerebot blockset.

A. Simulink scheduler templates

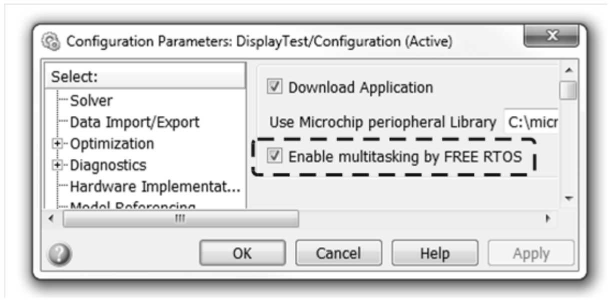
Since the Cerebot platform is designed only for one type of microcontroller without interchangeable crystals, unlike other blocksets, this one does not support setting parameters altering CPU clock speed by configuration options for clock dividers or multipliers (the CPU is configured to run on full clock speed all the time). Although this can be seen as unwanted limitation of the blockset flexibility, it makes using this blockset with Cerebot platform easier as no initial configuration is required from the user.

The blockset supports generating single and multitasking code. The multitasking code can be generated either as a cooperative multitasking or preemptive multitasking scheduled by the FreeRTOS system.

The multitasking code generated from Simulink model without the option to preempt slower task has obvious disadvantage; all tasks (even the slow ones) has to be finished within the period of fastest task. The FreeRTOS overcomes this limitation, however the code needed to maintain simulation is much bigger. Beside this, especially slower tasks have to use thread safe functions and implement critical section, where blocks of code should not be interrupted during execution.

Thus, the FreeRTOS scheduler presents a very straight forward solution for projects, which would be difficult to divide into separate functions running in cooperative mode otherwise. Compared to nested interrupts which would enable preemptive multitasking for simple algorithms, the RTOS implements separate stack for each thread and provide functions for managing functionality, which are necessary for instance in situations, where certain blocks of code can't be interrupted during execution.

The integration of generated code with a FreeRTOS requires selecting only one option in Cerebot configuration parameters dialog (see the option in Fig. 6)


Configuration Parameters: DisplayTest/Configuration (Active)

☑ Download Application

Use Microchip periophieral Library C:\micr

☑ Enable multitasking by FREE RTOS

Fig. 6 Enabling option for generating preemptive code powered by a Free RTOS from Simulink model

If this option is not selected, the standard rate monolithic scheduler without preemption functionality is created and is timed by periodical event generated by the on chip timer module.

When this option is selected, it enables seamless fully automatic integration of the generated code with the FreeRTOS system, where:

- *SysTick* is running with the same frequency as slowest task, therefore no "necessary" scheduler calls are performed.
- Threads are automatically created with groups of blocks that have the same execution time period.
- Each thread is timed using the *vTaskDelayUntil()* function - and synchronized (for offset) using the vTaskDelay() FreeRTOS task functions.
- Each thread is created using default size for stack (can be altered in template files).

B. Simulink blocks for simple peripherals

These blocks are used for implementing functionality of basic peripherals which does not require complex configuration. For creating these blocks standard options available from Simulink were used. That means creating a masked block using mask editor and created block c.mex and .tlc functions were not dynamically modified once the block was created.

For the intended purpose of the developed blockset, individual block functionality for simulation does not have to be implemented as modeling peripheral behavior was not necessary. This simplification saved a significant amount of time during Cerebot blockset development.

Particularly, blocks in our blockset which represent output peripherals have inputs for signals which are not used during simulation time. That means, no calculations are performed with signal which was connected to that block.

Input peripherals need to output some signal during simulation. But it can be constant value. The default zero value was outputted from blocks representing input peripherals.

This might seem as a limitation for development purposes. However, when testing the designed algorithm arbitrary signal can be used instead of input peripherals (ADC input, UART) blocks. Blocks of these peripherals need to be used only for code generation and does not to be present during algorithm development or testing.

Typical examples of blocks which are not available in commercially available blocksets are blocks for peripherals specific for particular board design

For instance the block for controlling LEDs: such block is not available on any other targets, as signaling LEDs are usually connected to various digital pins on microcontroller unit. When using a blockset for microchip MCU a digital output port block needs to be used. This concept is less clear as the user has to find to which port is the particular led connected. On the other hand, our block has inputs for each led making it easy to identify LEDs and corresponding block inputs (see Fig. 7).

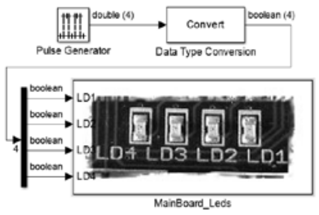


Fig. 7 Identifying block inputs and corresponding led location on board is simple

Another example of blocks crated for particular hardware version is blocks for controlling serial port (Fig. 8). One block is used for configuring UART port properties – speed, data parity etc. Other blocks are used for writing and reading data. All blocks use icons to illustrate their function. Since these blocks are configured to use only one build-in UART port, using this blockset is much easier compared to other products for automated code generation as the icon on blocks will help to identify proper port which is used with this particular block. These blocks for configuring UART interface and using the peripheral module are illustrated in Fig. 8. The configuration dialog is illustrated in Fig. 9.

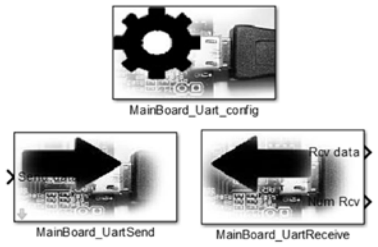
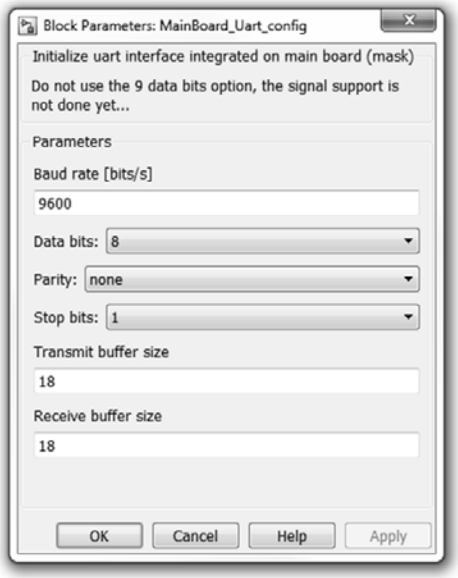


Fig. 8 Blocks and user interface for UART peripheral


Block Parameters: MainBoard_Uart_config

Initialize uart interface integrated on main board (mask)

Do not use the 9 data bits option, the signal support is not done yet...

Parameters

Baud rate [bits/s]: 9600

Data bits: 8

Parity: none

Stop bits: 1

Transmit buffer size: 18

Receive buffer size: 18

Fig. 9 Dialog for configuring UART peripheral module

C. *Simulink blocks for complex peripherals*

Implementing block for controlling display peripheral unit using Simulink mask editor or Matlab GUI wouldn't provide simple way for implementing tool for setting graphic components for display unit driven by embedded microcontroller. Therefore a special application was developed for this task.

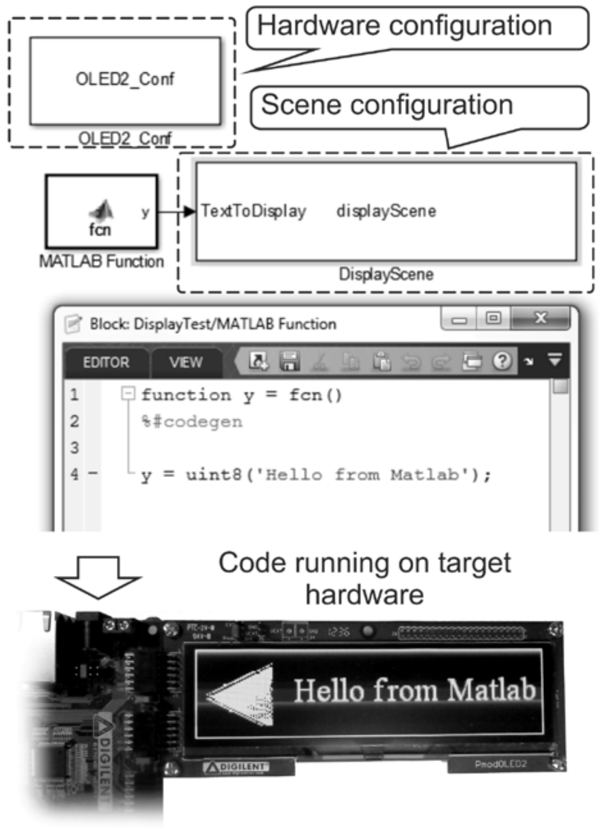
The Microchip Company provides free graphic library with source code for creating various graphic object which can be implemented with arbitrary graphic display unit (user only need to implement low lever layer of hardware drivers and configure the library based on used hardware type).

Implementing functions from Microchip graphic library saved significant amount of time since we can call these functions from generated code without the need to create them by ourselves.

The main task of blocks used to generate code for display unit is to:

- Place selected functions calls from graphic library into generated code. Most functions are executed during model step. Object initialization is moved to initialization section.
- Configure and maintain variables used as input for functions from graphic library. If the function requires an object as an input parameter, the generated code has to pack variables to appropriate structure.
- Link the selected variables with the Simulink signal (this mechanism allows controlling displayed objects from the Simulink model). Each signal used in the generated code has to have proper parameters (data type and size).
- Generate the functions which will be needed by the graphic library functions during linking code. Some functions use extern functions or variables, modules containing this objects need to be added to the compiled units.
- Generate request for adding library modules to compiled code. As some functions are using functions from other modules when starting the code generation process, all the required modules for the compilation are needed to be saved to MAKE variables. Once the *TLC* compiler starts, *RTW* parameters are locked and no changes done to them will be reflected in generated code.
- Provide the suitable user interface for selecting object which will be displayed and configure them. Easy to use graphical user interface was created for comfortable and easy display layout configuration.
- Create a link for calling the Microchip Graphic Resource Converter program. Before for instance bitmaps can be loaded and displayed, they need to be converted to proper format. Microchip provides the utility, which can perform the conversion. This utility can be started from program for modifying scene options. The generated files are detected automatically and objects (fonts and images) are then available for selection in object options in scene editor program.

Previously specified functionality was implemented using two Simulink blocks for the configuration of the generated code. Fig. 10 illustrates created blocks and hardware which is running code generated from this Simulink model.



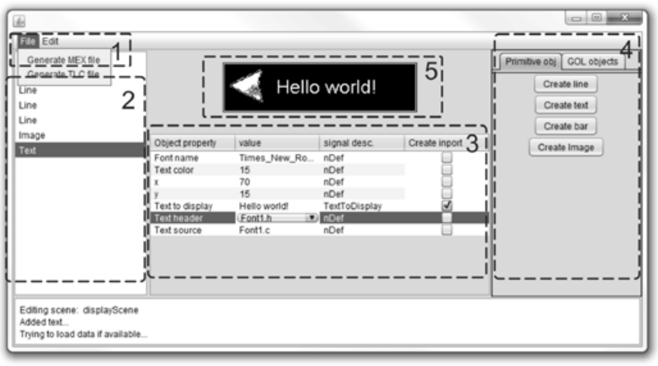
```
function y = fcn()
%#codegen

y = uint8('Hello from Matlab');
```

Fig. 10 Code generated from Simulink running on target hardware

One block (Hardware configuration) creates interface for configuring the selected display unit (creating driver layer based on the resolution of the display and the pin connection configuration. Second block (Scene configuration) configures the display layout. Since the layout represents a very complex structure a separate java application was created to provide user interface for configuring scene and mechanism for generating required files.

Creating a new instance of display scene block will create empty scene layout. When double clicking on scene layout block the external application will be called. After modifications of the scene parameters are complete, the *MEX* file is compiled if necessary and block mask updated. Scheme of the external application used for configuring display scene is illustrated in Fig. 11.



Line
Line
Line
Image
Text

Object property	value	signal desc.	Create inport
Font name	Times_New_Ro...	nDef	☐
Text color	15	nDef	☐
x	70	nDef	☐
y	15	nDef	☐
Text to display	Hello world!	TextToDisplay	✓
Text header	Font1.h	nDef	☐
Text source	Font1.c	nDef	☐

```
Editing scene:  displayScene
Added text...
Trying to load data if available...
```

Fig. 11 Overview of display scene layout editor hardware

Functions of different areas labeled in Fig. 10 are explained below.

1. The File option popup panel which presents options to generate new .mex and .h file for particular scene.
2. List of object placed to scene and allows its selection is displayed in this section.
3. The selected scene object properties are presented in this table. Based on element type, its properties can be fixed or entered from a Matlab Simulation. When Create inport option is checked a input port to a block with label corresponding to text in signal desc option is created. Value from this signal then modifies displayed item.
4. Clicking on buttons in this panel will place a corresponding object on screen.
5. Is used as a preview for generated scene layout preview.

As described in this chapter generating code for project which uses the display peripheral is very easy when using Cerebot blockset. Adding one configuring and at least one scene block is sufficient to display objects on OLED2 module. The display scene is configured in graphical user interface and selected object properties can be controlled from model using Simulink signals (connected to input port of scene option block).

D. Blockset "supporting" files

Another group of files which were created or configured, to integrate external tools to support code generation process, were make script and callback hooks.

When make script is generated it is automatically calls XC32 compiler with proper parameters. Once the executable code is generated, callback method is used to invoke loader, which loads generated program to a MCU flash memory. This loader was written in Java language using free libraries for communication with the programmer provided by a Microchip Company. The programmer/debugger chip is a part of the Cerebot platform and has an USB interface for communication with a PC.

IV. Magnetic levitation controller

A. Plant and hardware setup

Controlling the magnetic levitation is a little tricky task. The plant is unstable and with a very small time constant. The steel ball is floating in magnetic field, which has to be strong enough to "levitate it" but not too strong to "suck it". Therefore the control loop has to be executed very quickly (the frequency of 1000Hz is sufficient for PID controller). Beside this the controller uses a display module to show value for desired steel ball position. Redrawing the display is a very slow process; therefore it is convenient to have a slower task which will update the display with lower period than the one used for computing PID control algorithm. The controlled plant is illustrated in Fig. 12, left.

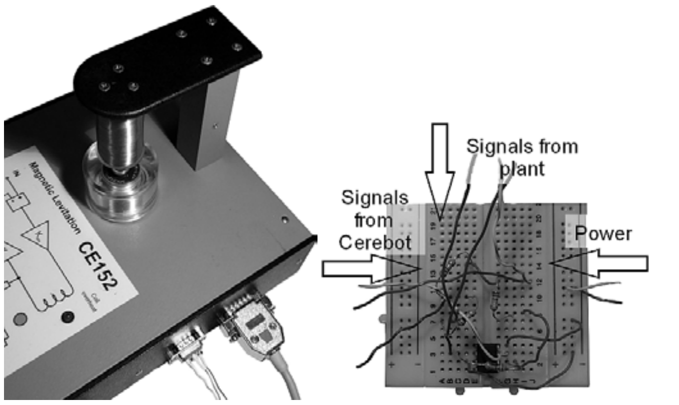


Fig. 12 Overview controlled plant (left) and signal conditioning circuits (right)

This plant can be almost directly connected to a Cerebot hardware and modules. Only the voltage has to be properly scaled (down for output signals, up for input signal). We use only one output signal from the controlled plant – the height in which is the steel ball floating. And one input signal, that is used for controlling current going through coil in the electromagnet levitating the steel bal. The voltage conversion circuits consist only from resistors and operating amplifier. Fig. 12, right illustrates how simply and quickly can be created module for scaling the voltage between Cerebot and controlled plant.

The Cerebot hardware uses several modules. A display module is used to provide information to user, a digital to analog converter module is used for generating control signal for magnetic levitation and one on chip analog to digital converter module is used to read the voltage from ball position sensor. The complete Cerebot hardware setup is illustrated on Fig. 13.

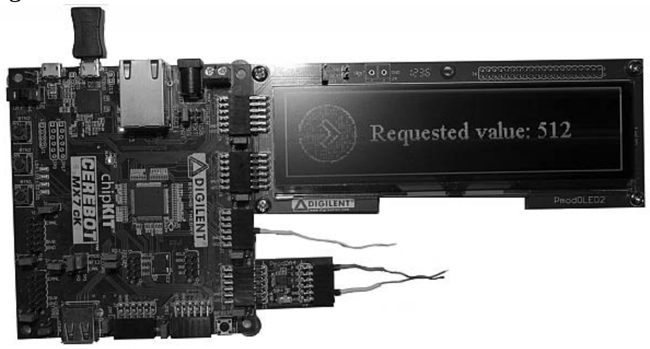


Fig. 13 The Cerebot hardware with connected peripheral modules

The next step is to create software. This can be done in Simulink environment as described in next chapter.

B. *Software setup*

The Simulink model running on the target consists of two types of blocks. The one group is the control algorithm and second one consists of blocks used for handing the peripherals. Implemented model is illustrated on Fig. 14

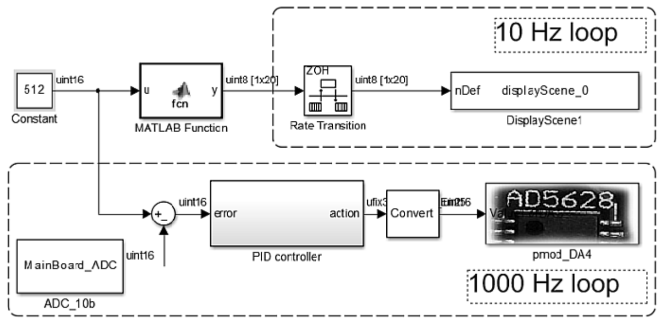


Fig. 14 Simulink scheme of algorithm implemented in Cerebot

This simulation is powered by a freeRTOS kernel, which creates one task for fast and one for slow simulation part (the fast and slow tasks are marked in Fig. 14). The implemented PID control algorithm uses following peripheral modules:

- Block for reading value from onchip ADC module.
- Block for writing value to Pmod DAC module.
- Block for writing value to Pmod display module.

For reference, the entire simulation consists only of roughly 20 blocks and 20 lines of code in embedded Matlab function block (implements functions for formatting strings for display module). It generates 25 .c and .h files and uses another 12 .c modules from libraries (for display and RTOS system) having together several thousand lines of code. Most of these files would have to be written or configured manually if the support for automatic code generation is not available.

V. Conclusion

Created support for automatic code generation from Simulink presents a tool which can very significantly speed up the application development cycle.

As demonstrated in previous chapter, the executable code can be created very quickly. Once the control algorithm is designed it can be used in Simulink model from which the executable code is generated. Only blocks for interfacing with peripheral modules need to be added.

Created blockset can help speed up the development process of any application which is using this hardware. Since it is an open source, missing blocks for unsupported peripherals can be easily created by modifying existing ones, which makes this blockset more flexible than available commercial products.

References

[1] E. Markopoulos, J. Bilbao, E. Christodooulou, T. Stoilov, T. Vos, C. Makatsoris, "Process Development and Management: towards the maturity of organizations", in: NAUN International Journal of Computers Vol2 (4) 2008, pp.361-370

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

[3] ITO, Kunihihiko; MATSUURA, Saeko. Model driven development for embedded systems. In: Proceedings of the 9th WSEAS international conference on Software engineering, parallel and distributed systems. World Scientific and Engineering Academy and Society (WSEAS), 2010. p. 102-108.

[4] Kuhl, M.; Reichmann, C.; Protel, I.; Muller-Glaser, K.D.; , "From object-oriented modeling to code generation for rapid prototyping of embedded electronic systems," Rapid System Prototyping, 2002. Proceedings. 13th IEEE International Workshop on , vol., no., pp. 108-114, 2002

[5] Duma, R.; Dobra, P.; Abrudean, M.; Dobra, M.; , "Rapid prototyping of control systems using embedded target for TI C2000 DSP," Control & Automation, 2007. MED '07. Mediterranean Conference on , vol., no., pp.1-5, 27-29 June 2007

[6] M. Stanek, D. Manas, M. Manas, J. Navratil, K. Kyas, V. Senkerik, A. Skrobak, "Comparison of different rapid prototyping methods", International Journal of Mathematics and Computers in Simulation 6 (6), pp. 550-557

[7] Bližnák, Michal, et al. "Optimized Production-Ready Source Code Generation Based on UML." INTERNATIONAL JOURNAL OF SYSTEMS APPLICATIONS, ENGINEERING & DEVELOPMENT Issue 1, Volume 7, 2013

[8] Toom, A., Izerrouken, N., Naks, T., Pantel, M., Ssi-Yan-Kai, O.: Towards reliable code generation with an open tool: Evolutions of the Gene-Auto toolset. In: ERTS. Societe des Ingenieurs de l'Automobile (2010)

[9] Netland, Ø.; Skavhaug, A.; , "Adaption of MathWorks Real-Time Workshop for an Unsupported Embedded Platform," Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on , vol., no., pp.425-430, 1-3 Sept. 2010.

[10] "Simulink - Embedded Target for PIC." Lubin Kerhuel's Website. N.p., n.d. Web. 31 Jan. 2014.

[11] "MPLAB 16-Bit Device Blocks for Simulink." Development Tools. N.p., n.d. Web. 31 Jan. 2014.

[12] Lambersky, V., "Model based design and automated code generation from Simulink targeted for TMS570 MCU," Education and Research Conference (EDERC), 2012 5th European DSP , vol., no., pp.225,228, 13-14 Sept. 2012

V. Lamberský was born in Moravska Trebova, Czech Republic, in 1985. He received the BSc degree in Mechatronical Engineering in 2008 from the Faculty of Mechanical Engineering at Brno University of Technology, and engineer degree in Mechatronical Engineering in 2010 from the Faculty of Mechanical Engineering at Brno University of Technology. Currently he is pursuing his ph.D degree. His current research focuses on methods for predicting computing performance of control algorithms on embedded hardware and methods for implementing support for automated code generation from higher programming languages.