BLDC motor control using rapid control prototyping

Radu Duma, Mirela Trușcă, Petru Dobra

Technical University of Cluj–Napoca, Automatic Control Department

Abstract: The paper presents a Rapid Control Prototyping (RCP) toolbox, Target for Renesas M32C87, for Matlab/Simulink which can be used to generate real-time C code for the Renesas M32C87 microcontroller. The RCP toolbox contains a digital motor control library which implements a brushless direct current (BLDC) motor control algorithm. A practical application for closed loop speed control of brushless direct current motor is presented.

Keywords: rapid control prototyping, automatic code generation, real-time, M32C87 microcontroller, PID, brushless direct current motor

1. INTRODUCTION

Developing controllers for applications (electrical drive systems) means large expenditure, when performed with usual development methods. The workload comprises development of a mathematical model as well as algorithm design and implementation, off-line simulation, and optimization. The whole process has to be restarted on occurring errors or divergences, which makes the development process time consuming and costly [8].

RCP is a way out of this situation, especially if the control algorithm is complex and a lot of iteration steps are necessary. RCP requires two components: a Computer Aided Control System Design (CACSD) software and a dedicated hardware capable of running hard real-time tasks (Fig.1).

CACSD tools are extensively used to generate real-time code automatically. The graphical programming approach removes the need to write software by hand and allows the engineer to focus instead on improving functionality and performance. Complete system design is carried out within the simulation environment.

With the great diversity of applications, a development environment must be flexible and provide exactly the functionality necessary for efficient problem solving. Mathworks’ Simulink [11] software is such a graphical modeling tool. MathWorks developed toolboxes for some wildly used targets: Motorola MPC555, Infinion C166, C2000 and C6000 DSP families from Texas Instruments. Rebeschiss [14] developed the MICROS toolbox for standard 80C166 microcontrollers. A DSP based RCP system for engineering education and research using as CACSD tool Matlab/Simulink is presented in [9].

Hanselmann [7] from the dSPACE GmbH presented ‘Total Development Environment’ (TDE) for rapid control prototyping. TDE includes MATLAB, Simulink, powerful hardware, based on the DSP’s, and an additional set of software tools for online data visualization (COCKPIT, TRACE). Controller boards like DS1104 and DS1103 are appropriate for motion control and are fully programmable from the Simulink environment.

Microchip [12] developed a Matlab RCP toolbox for the dsPIC33 Digital Signal Controllers (DSC), while Kerhuel [10] developed a toolbox which offers support for several Microchip microcontrollers and DSCs.

A free, open source, solution is based on Scilab/Scicos [16] and Linux-RTAI [15], which uses the processor of a general purpose computer for executing real-time tasks. A real-time patch is applied to the standard Linux kernel. A modified version of the Scicos code generator, RTAI-Lab generates hard real-time code compatible with Linux-RTAI. Acquisition cards can be directly integrated into the Scicos scheme and into the generated code using drivers provided by the COMEDI [1] project.


In Fig. 2 is presented the bloc diagram for the information flow (from concept to model) and for the data flow (between PC and the Renesas M32C87 microcontroller), for the implemented toolbox.

Control algorithms, for the M32C87 microcontroller can be developed using blocks from the Target for Renesas M32C87 toolbox and predefined blocks from Simulink or user defined blocks. After the validation of the model the code can be
generated using Real-Time Workshop. At the end of the code generation process the HEW IDE is automatically started, a new HEW project is created, the generated code-source files are added to the project, the project is compiled and the binary file is downloaded to the target processor. The execution of the code can be monitored in real-time using the implemented CAN drivers.

The processor has several communication interfaces: 6 serial interfaces for synchronous and asynchronous communication, I²C, GCI and IrDa; 2 CAN channels.

The Renesas microcontrollers are supported in several IDEs: HEW, IAR, TASKING. Each of this IDE has its own integrate compiler for the M16C family. The executables can be downloaded on the processor using the E8 or E8a debuggers, or the Flash Development Toolkit and M16C Flasher software.

The features presented in this section make the ModSDKM16C-CM32C87 development kit an optimal solution for an RCP toolbox.

3. TARGET FOR RENESAS M32C87

The Simulink toolbox for the M32C87 processor is presented in Fig. 4.

The toolbox contains five libraries: I/O drivers (contains drivers for the I/O peripherals of the target processor), DMC (contains blocks which implement digital motor control algorithms), CAN (contains blocks which implement drivers for the CAN bus of the M32C87 processor), SCI contains blocks which implement drivers for the serial interface of the M32C87 processor) and M32C87 Target Preferences (defines a target preference class for the implemented toolbox). In the following subsection the DMC library will be presented in detail.

3.1 M32C87 Digital Motor Control library

The DMC library (Fig. 5) contains blocks which implement digital motor control algorithms.
The implemented algorithms are optimized for the M32C87 microcontroller, which is a fixed point processor. The hardware multiplier was used and only integer variables where declared, in order to obtain the lowest sampling rate, without breaking the real-time constrains.

The library contains the following blocks: **M32C87 PID** (implements a PID controller with antisaturation), **M32C87 Three-Phase Motor Control** (configures the M32C87 processor to generate six complementary PWM signals with dead-time) and **M32C87 BLDC Motor Control** (implements a trapezoidal six step control algorithm for a BLDC motor).

### 3.1.1 M32C87 PID

The standard equation of a PID controller is presented below:

\[
\begin{align*}
    u(t) &= K_p \left( \epsilon(t) + \frac{1}{T_i} \int_0^t \epsilon(\tau)d\tau + T_d \frac{de(t)}{dt} \right) \\
    \epsilon(t) &= y_{sp}(t) - y(t) \quad \text{is the error signal, } y_{sp} \text{ is the set point and } y \text{ is the output of the process.}
\end{align*}
\]

where \(\epsilon(t)\) is \(y_{sp}(t) - y(t)\) is the error signal, \(y_{sp}\) is the set point and \(y\) is the output of the process.

During years of research several modifications have been done to the standard PID controller structure. A pure derivative cannot and should not be implemented, because it will give a very large amplification of measurement noise. The gain of the derivative must thus be limited. This can be done by low-pass filtering the derivative term, resulting in a limited gain \(N\) at high frequencies. \(N\) is typically in the range of 3-20. The derivative effect will work only on the output of the system, and not on the command. Also only a fraction \(\beta\) of the command signal will act on the proportional part.

If the control error \(\epsilon(t) = y_{sp}(t) - y(t)\) is so large that the control output saturates the actuator, the feedback path will be "broken", because the actuator remains saturated even if the process output changes. The integrator may then integrate up to a very large value. When the error is finally reduced, the integral may be so large that it takes a considerable time to reach a normal value again. This effect is called integrator windup and one way to avoid it is shown in Fig. 6.

An extra feedback path is provided in the controller by measuring the actuator output and forming an error signal \(u-v\), which is filtered and fed back through the integrator part of the controller.

Taking into consideration the above remarks, the PID algorithm can be described by the following equations:

\[
\begin{align*}
    P(n) &= K_p (\beta y_{sp}(n) - y(n)) \\
    \alpha_d &= T_d / (T_d + N h) \\
    \beta_d &= \left( K_p T_s N \right) / (T_d + N h) \\
    \beta_i &= K_p T_s / T_i \\
    \beta_v &= T_s / T_i
\end{align*}
\]

A Simulink block is defined by two functions: a MEX function written in the C language and a Target Language Compiler (TLC) implementation. The MEX function configures the number of inputs/outputs ports of the block and transfers the parameters specified in the graphical user interface of the block in the code generation process. The TLC implementation initializes the I/O device, computes the value for the output of the block and terminates the program by setting the hardware to a "neutral" state. Equations (1) were used for TLC implementation of the M32C87 PID block.

### 3.1.2 M32C87 BLDC Motor Control

BLDC motors are one of the motor types rapidly gaining popularity. BLDC motors are used in industries such as Appliances, Automotive, Aerospace, Consumer, Medical, Industrial Automation Equipment and Instrumentation. BLDC motors do not use brushes for commutation; instead, they are electronically commutated. BLDC motors have many advantages over brushed DC motors and induction motors.

The M32C87 BLDC Motor Control block implements a trapezoidal six step control algorithm. The speed of the motor...
is controlled using PWM signals. The block uses as modulation technique the upper modulation. Only the upper transistors of the three phase bridge are controlled using PWM signals while for the lower transistors enable/disable signals are used.

The block has an input which represents the duty cycle of the PWM signals. The block can determine the speed of the motor using the signals from the Hall sensors. In this case, the block has an output port which represents the speed of the BLDC motor in rotation per minute. Also, for speed measurement an encoder can be used, in which case the block will not have an output port.

Fig. 7 presents the Function Block Parameters window for setting the parameters of the M32C87 BLDC Motor Control block.

For the M32C87 BLDC Motor Control block the following parameters have to be specified: the input pins on which the signals form the Hall sensors will be applied, the commutation sequence, the modules of timer A which will be used for the generation of the PWM signals, the period of the PWM signals, the pins which will be used for the enable/disable of the lower transistors of the three phase bridge.

4. BLDC MOTOR CONTROL

The test setup presented in Fig. 8 was used to test the Target for Renesas M32C87 toolbox. An application for controlling the speed of the BLDC motor was implemented. A Hurst-Emerson motor was used. On the shaft of the motor an Agilent HEDS-5500 encoder was mounted. The integrated power module IRAMS10UP60A was used as motor drive, which is optimized for electronic motor control.

Unlike a brushed DC motor, the commutation of a BLDC motor is controlled electronically. To rotate the BLDC motor, the stator windings should be energized in a certain sequence. The energizing sequence must be introduce in the Function Block Parameters window of the M32C87 BLDC Motor Control block by selecting a corresponding option from the drop down menu corresponding to Step1, Step2, Step3, Step4, Step5 and Step 6 options.

The block diagram for a closed control system, using the Renesas M32C87 microcontroller is presented in Fig. 9.

The timer A of the microcontroller is used to generate three PWM signals for the upper arm transistors of the three phase bridge, while three general purpose I/O pins are used to generate the enable/disable signals for the lower arm transistors of the three phase bridge.

The speed of the motor can be determined using the signals from the Hall sensors. A timer of the microcontroller can be used to count the clock cycles between two Hall transitions, and using the timer value the speed of the motor can be computed. For applications which require a high accuracy of the measured speed an encoder must be used. The encoder generates two-phase pulse signals.
4.2 Controller tuning

For controller tuning the robust tuning method presented in [2] was used. The paper presents a new tuning rule which gives a new relationship between $T_i$ and $T_d$ in stead of the equation $T_i = 4T_d$, proposed in the modified Ziegler-Nichols methods [5][6]. The equations for determining the parameters of the PID controller are presented below:

$$K_p = \frac{\cos(\gamma_m)}{|P(j\omega_c)|1 + \tan^2(\gamma_m - \angle P(j\omega_c))}$$

$$T_i = \frac{-2}{\omega_i s_p(\omega) + \tan(\Phi) + \tan^2(\Phi)s_p(\omega_i)}$$

$$Td = \frac{-T_bw_0 + 2s_p(w_0) + \sqrt{\Delta}}{2s_p(w_0)w_0^2T_i}$$

where: $\gamma_m$ is the phase margin, $\Phi = \gamma_m - \angle P(j\omega_c)$, $\Delta = T_i^2w_0^2 - 8s_p(w_0)T_bw_0 - 4T_i^2w_0^2s_p^2(w_0)$.

In order to estimate the phase and the module of the process at the specified frequency a closed loop system with relay and time delay has to be implemented. The Simulink model for the closed loop system with relay and time delay is presented in Fig. 10.

The reference speed is set from the potentiometer of the reference board, using the M32C87 ADC block from the I/O drivers library. For speed measurement the M32C87 Encoder block from the I/O drivers library is used. The signal from the encoder is filtered. If the difference between the reference speed and the measured one is smaller than zero, the duty ratio of the PWM signal will be 92%, while if the error signal is greater than zero the duty ratio of the PWM signal will be 68% thus obtaining a limit cycle. In order to retrieve data from the target in real-time, blocks from the CAN library are used. A configuration M32C87 CAN Configure block is added to the model. It does not connect to any other blocks, but stands alone to configure the selected CAN module. An M32C87 CAN Transmit block is used for sending the speed of the motor over the CAN bus.

A target preference block has to be added to the model. The M32C87 Target block does not connect to any other blocks, but stands alone to set the target preferences for the model.

After the validation of the model the code generation process is invoked, and the generated executable file is downloaded to the M32C87 processor.

The application presented in Section 5 is used for communication with the target processor over the CAN bus. Using this application the limit cycle is logged, and the results are presented in Fig. 11.

4.3 Controller implementation

The closed loop Simulink model for BLDC motor speed control is presented in Fig. 12.

The reference speed is a rectangular signal whose level is set over the CAN bus, using the M32C87 CAN Receive block. The reference speed and the measured speed, using an M32C87 Encoder block, are applied at the input ports of a M32C87 PID block. The M32C87 PID will compute a new value for the duty cycle of the PWM signal. The reference speed, the measured speed and the duty cycle of the PWM signal are concatenated into an array and are sent over the CAN bus using a M32C87 CAN Transmit block.

Fig. 9. Block diagram for a closed control system.

Fig. 10. Simulink model for the closed loop system with relay and time delay.

Fig. 11. Limit cycle of the BLDC motor.

From Fig. 11 the module and phase of the process are estimated, and the parameters of the controller are computed.

Fig. 12. Simulink model for BLDC motor speed control.
After the validation of the model the code generation process is invoked, and the generated executable file is downloaded to the M32C87 processor.

Using the graphical user interface presented in Section 5, the reference speed, the measured speed and the duty cycle of the PWM signals can be monitored and analyzed. The response of the motor using the determined PID controller is presented in the upper graphic of the Fig 15. The motor phase voltages during commutation are presented in Fig. 13.

5. INSTRUMENTATION, DATA ACQUISITION AND RESULTS

The target should support a mechanism that can be used to observe the target code as it runs in real-time (outside of a debugger). One industry-standard approach is to use the CAN bus.

For this purpose the CAN library was implemented. Using blocks from this library data can be fetched and retrieved from the target processor in real-time.

In order to communicate between the target processor and the PC an USB-CAN converter produced by Systec-Electronic [17] was used. There is an API for the converter in the form of an .dll file.

Using the USBCAN32.dll file a Matlab application for the CAN bus monitoring was implemented. After loading the .dll file in the Matlab environment, using the loadlibrary command, functions are called which implement the communication with the USB-CAN converter. The instrumentation panel implemented is presented in Fig. 14 and Fig. 15.

In Fig. 14 is presented the step response of the BLDC motor to a rectangular reference signal when the integral correction gain is set to zero, and in Fig. 15 the response of the motor when the integrator windup is avoided. The overshoot and the time response are smaller when the saturation of the command is avoided.

6. CONCLUSIONS

The paper presented the implementation of a rapid control prototyping, Target for Renesas M32C87, toolbox for Matlab/Simulink, which generates real-time C code for the Renesas M32C87 microcontroller. For this processor until now there was not a RCP toolbox. Using the toolbox one can generate code for the M32C87 processor without knowing in detail the architecture and peripherals of the microcontroller.

Another contribution of the paper is the implementation of a block for BLDC motor control, which is used for automatic code generation. Using this block one does not have to implement the control algorithm manually.

A closed loop control system for BLDC motor speed control was implemented. For the controller tuning, a robust tuning method based on the flat phase criteria was used. For this method in the literature there were only results proved by simulation. A contribution of the paper is
the utilization with success of this method for a BLDC motor.

For real-time data exchange between the target processor and the PC, the CAN bus was used. A CAN library which contains blocks for sending/receiving messages over the CAN bus, and a graphical user interface for control and monitoring of the CAN bus were implemented.

REFERENCES