Automated Code Generation for Development of Electric Vehicle Controller

Peng Geng, Minggao Ouyang, Jianqiu Li and Liangfei Xu

Abstract Code generation for Simulink model is widely used in the development of vehicle controller. In traditional way, only the code of vehicle control model can be generated automatically by Real-Time Workshop. So programmers have to handwrite code for peripheral device drivers. The objective of this study was to develop a custom driver blocksets to support I/O devices on high performance 32-bit MPC5644A microcontroller. Furthermore, to fulfill modeling, code generation, compilation and downloading all accomplished "at the touch of a button". This method is used and tested in the development of electric vehicle controller, which saves time and money greatly.

Keywords Automated code generation · Embedded target · Matlab/Simulink · MPC5644A · Electric vehicle controller

1 Introduction

In order to save time and cost, software engineers always focus on the length of code but do not pay attention to the correctness, which makes code difficult to maintain and transplant. Moreover, most software projects are completed in collaboration. If programmers handwrite code separately and combine them together at last, it is almost impossible to put into application directly because of

F2012-D03-019

P. Geng (✉) · M. Ouyang · J. Li · L. Xu
State Key Laboratory of Automotive Safety and Energy, Tsinghua University, Beijing, China
e-mail: gengp10@mails.tsinghua.edu.cn

all kinds of handwritten errors. Matlab/Simulink is a powerful tool which is widely used in the “V-Model” development of Vehicle Control Unit (VCU). “V-Model” includes function design, Rapid Control Prototyping (RCP), code generation for embedded target, Hardware-In-the-Loop Simulation (HILS) and test/calibration. Thanks to the powerful functions in modelling, simulating and analyzing, Matlab/Simulink becomes a basic tool in modelling the VCU algorithm as well as in code generation. Although this “half” automated code generation method unifies code style and increases efficiency, it still has weaknesses. In the development of VCU, code generation includes model algorithm code and microchip driver code. The later needs to be done through handwriting for the reason that Matlab/Simulink provides very limited toolboxes which only include Motorola MPC555, Infineon C166, TI’s C2000, C5000 and C6000 [1]. To solve the problem, Microchip develops RCP toolbox, which supports dsPIC33 Controller [2]. D.Hercog develops a self-made, DSP-based RCP system, which is used for motor control [3]. S.Rebeschiess launches an embedded toolbox, named “MIRCOS”, for graphical programming of 16-bit processor 80C166 [4]. R.Duma presents a RCP toolbox for Renesas M32C87 microcontroller [5], and C.RUSU invented a toolbox for F24X DSK Digital system in the purpose of education and industrial application [6]. R.Bartosinski gives a method of integrating Processor Expert Tool to Matlab environment [7]. Luo devotes himself to the method of Real Time Workshop (RTW) and developing a custom embedded target—MC9S08DZ60 [8].

This paper firstly introduces a method of developing device drivers for embedded target—32-bit MPC5644A, using Matlab Real-Time Workshop Embedded Coder (RTWEC). Then it applies this method to the development of electric vehicle controller, focus on vehicle control algorithm, real-time scheduler and generated code analysis. Verification and conclusion is given at last.

2 Developing Embedded Targets

2.1 Real-Time Workshop Embedded Coder

As a powerful tool of Matlab/Simulink, RTWEC is able to convert graphic model into high level language code, which frees software engineers from all kinds of datasheets and redundant handwriting code. Besides, this compact and fast code is very important for MCU used in production and real-time embedded systems. With RTWEC, Simulink model is translated into an intermediate RTW file first, which contains all the model-specific information used in code generation. Target Language Compiler (TLC) generates code according to this file using system control files, block target files and so on. Though RTWEC increases efficiency greatly, limitations exist. First, Matlab only supports several embedded target platforms such as MPC5xx series. In addition, these toolboxs can only realize simple hardware functions which hardly satisfy users’ need. So, it is necessary to develop custom embedded target.

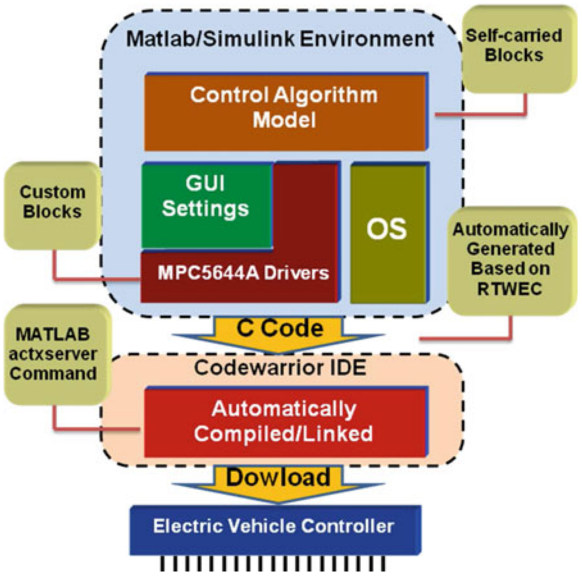

Matlab/Simulink Environment
Self-carried Blocks
Control Algorithm Model
GUI Settings
Custom Blocks
OS
MPC5644A Drivers
Automatically Generated Based on RTWEC
C Code
MATLAB actxserver Command
Codewarrior IDE
Automatically Compiled/Linked
Dowload
Electric Vehicle Controller

Fig. 1 Automated C code generation platform for MPC5644A

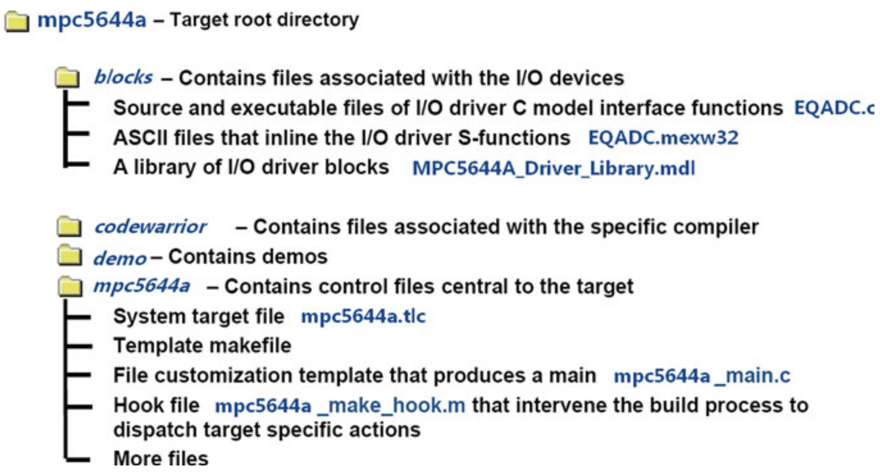
Figure 1 introduces how to develop an embedded target MPC5644A used in next-generation fuel cell vehicle control. Building a custom MPC5644A driver library is the first step, using the method mentioned later. This library provides an easy-to-use graphical user interface for device driver in the Simulink environment. Then, RTWEC can generate high quality code according to vehicle control model automatically. This method also integrates RTWEC with cross-development environments—CodeWarrior IDE which makes code generation, compilation and downloading all seamlessly accomplished "at the touch of a button".

2.2 Developing Target Directory Structure

Baseline target files include system target files, template make files, target block library files, target block files and so on. These target files are readable and easy to understand so the developer can customize the target for application needs. Initial tasks in developing an embedded target are creating a series of organized directories in Matlab path, and locating required target files in specific directories, see Fig. 2.

2.3 Modifying System Control Files

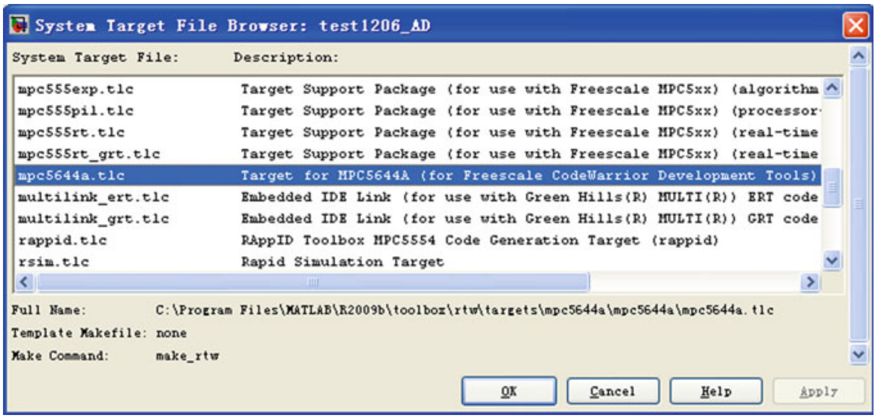
System control files includes system target files (STF), template make files (TMF), hook files and so on. STF controls the presentation of target to end user. Developer



mpc5644a – Target root directory

- blocks – Contains files associated with the I/O devices
  - Source and executable files of I/O driver C model interface functions EQADC.c
  - ASCII files that inline the I/O driver S-functions EQADC.mexw32
  - A library of I/O driver blocks MPC5644A_Driver_Library.mdl
- codewarrior – Contains files associated with the specific compiler
- demo – Contains demos
- mpc5644a – Contains control files central to the target
  - System target file mpc5644a.tlc
  - Template makefile
  - File customization template that produces a main mpc5644a _main.c
  - Hook file mpc5644a _make_hook.m that intervene the build process to dispatch target specific actions
  - More files

Fig. 2 MPC5644A directory structure



System Target File Browser: test1206_AD

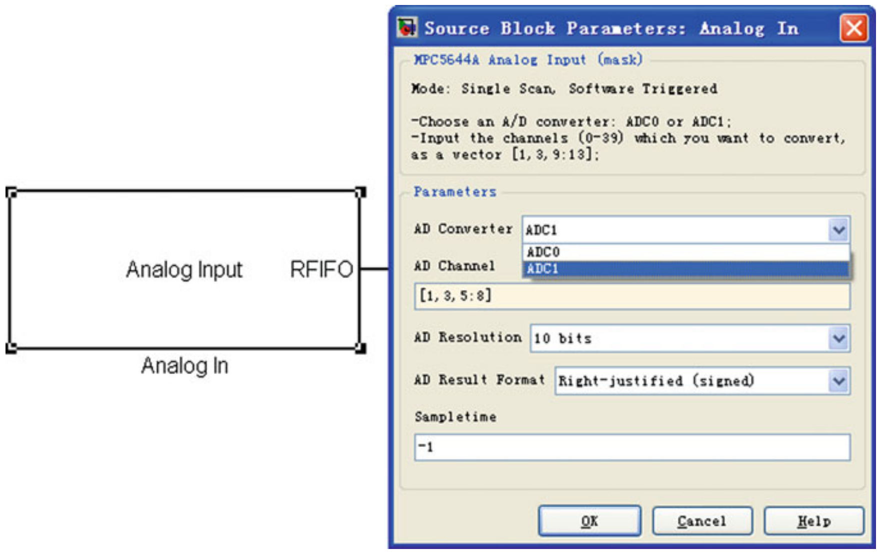
```
System Target File:      Description:
mpc555exp.tlc            Target Support Package (for use with Freescale MPC5xx) (algorithm
mpc555pil.tlc            Target Support Package (for use with Freescale MPC5xx) (processor-
mpc555rt.tlc             Target Support Package (for use with Freescale MPC5xx) (real-time
mpc555rt_grt.tlc         Target Support Package (for use with Freescale MPC5xx) (real-time
mpc5644a.tlc             Target for MPC5644A (for Freescale CodeWarrior Development Tools)
multilink_ert.tlc        Embedded IDE Link (for use with Green Hills(R) MULTI(R)) ERT code
multilink_grt.tlc        Embedded IDE Link (for use with Green Hills(R) MULTI(R)) GRT code
rappid.tlc               RAppID Toolbox MPC5554 Code Generation Target (rappid)
rsim.tlc                 Rapid Simulation Target

Full Name:          C:\Program Files\MATLAB\R2009b\toolbox\rtw\targets\mpc5644a\mpc5644a\mpc5644a.tlc
Template Makefile:  none
Make Command:       make_rtw
```

Fig. 3 STF mpc5644a.tlc in system target file browser

can modify this file to display custom target information in the STF browser. Moreover, STF also provides the main entry point to the top-level TLC, the definition of target-specific code generation, inheriting options from another target, and so on. "mpc5644a.tlc" is an ERT-based STF created for processor MPC5644A, see Fig. 3.

Hook file can customize building process and invoke target-specific actions at specified points. In order to compile, link, download the generated code automatically, hook file needs to interface with development tool. Take MPC5644A for example, solutions are given to support the complete build process which interfaces with the Freescale CodeWarrior IDE. To implement this function, developer should create an eXtensible Markup Language (XML) file, which indicates CodeWarrior project to add the generated source/header files. Build process automation depends



Source Block Parameters: Analog In

MPC5644A Analog Input (mask)

Mode: Single Scan, Software Triggered

-Choose an A/D converter: ADC0 or ADC1;
-Input the channels (0-39) which you want to convert, as a vector [1, 3, 9:13];

Parameters

AD Converter: ADC1 (ADC0, ADC1)

AD Channel: [1, 3, 5:8]

AD Resolution: 10 bits

AD Result Format: Right-justified (signed)

Sampletime: -1

Fig. 4 Dialog box for eQADC driver block

on Matlab COM automation functions. A series of CodeWarrior API functions such as “CreateCWComObject”, “OpenCW”, “OpenMCP”, and “BuildCW” are invoked in hook file, after the code generation process is completed. Then Code-warrior starts automatically, compiles and downloads the code into controller through BDM. This example helps developers to deal with similar interfacing problems with particular integrated development environment.

2.4 Developing Device Drivers for Embedded Target

To create device drivers for MPC5644A, a C MEX S-function is built primarily for use in simulation, and a driver TLC is created for use in code generation. C MEX S-Function is a C code file that implements specific functions, such as “mdlInitializeSizes”, “mdlOutputs”, “mdlRTW”, to initialize driver block structure, validate block parameter data input by end users and pass these data to rtw file. Command “mex” is used to build a MEX-file from custom driver source code and designer can add icons, dialog boxes, initialization commands to an S-Function block by masking it, see Fig. 4.

Driver TLC is used to create hardware support files including header files, macro definitions and driver code libraries. For example, the purpose of the “start” function in eQADC driver TLC “eqadc.tlc” is to generate code that initializes 32-bit control registers of MPC5644A Analog-to-Digital driver; while the “output” function generates code that repeats the "convert" operations for all selected AD channels. Code related to hardware drivers are all generated to a source file named "MPC5644A_drivers.c"

3 Simulink Fixed Point and Custom Storage Classes

3.1 Writing Fixed-Point S-Functions

Within microprocessor like MPC5644A, numbers are represented as either floating-point or fixed-point data types. Floating-point data types contain three parts: sign bit, fraction field, and exponent field; while fixed-point numbers are characterized by word size in bits and binary point, whether signed or unsigned. If a MCU support fixed-point, it always means the chip size can be smaller with less power consumption. Furthermore, fixed-point calculation needs less time, memory and cost compared with floating-point. So, when write C S-functions, how to support fixed-point data types is an inevitable question.

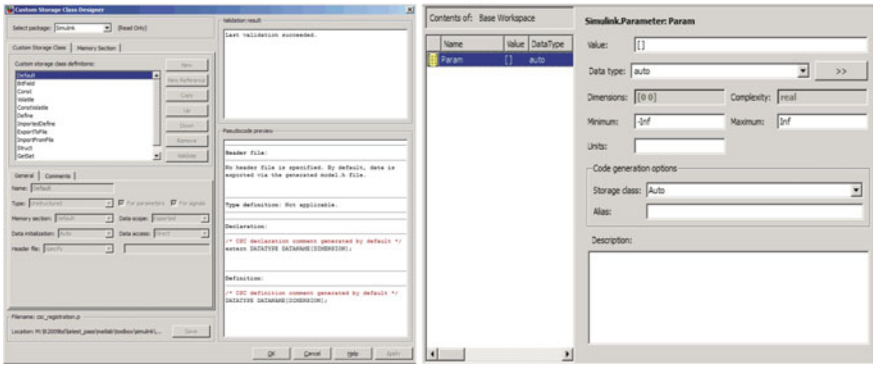
User-written S-functions supports many kinds of fixed-point data types, such as "sfix16_En13", "ufix32_En11", and "sfix35_S3_B4", provided that "fixed-point.h" and "fixedpoint.c" have been included at appropriate places in the S-function. Each data type used in S-function has a unique data type ID, which programmers use to get and set information about data types. The assignment of data type IDs follow "first come first served" rules. To set fixed-point data types, developer should register a data type first using data type registration functions. And then use the returned data type ID for input and output ports, run-time parameters.

3.2 Creating and Using Custom Storage Classes

Storage classes are widely used in the code generation process. They decide how signals, parameters are declared and stored in generated files. There are four built-in types that can satisfy users demand most of the time. For example, storage class "Auto" put all data into one single structure while "ExportedGlobal" variables are declared and stored in unstructured way.

In the development of embedded system, programmers often need flexible control over the representation of data. Thus, custom storage classes (CSC) are provided as complement of built-in storage class to control the structure of generated data. Another advantage is that developer can create and edit memory section definition using CSC.

Before creating CSC, a data object package should be built to support CSC definition using data class designer, see Fig. 5. CSC designer is a tool for creating and managing custom storage classes and memory sections. Developer can choose



Select package: Simulink (Read Only)

Custom storage class definitions: Default, BitField, Const, Volatile, ConstVolatile, Define, ImportedDefine, ExportToFile, ImportFromFile, Struct, GetSet

Name: Default
Type: Unstructured ☑ For parameters ☑ For signals
Memory section: Default Data scope: Exported
Data initialization: Auto Data access: Direct
Header file: Specify

Filename: csc_registration.p
Location: M:\R2009b\Latest_pass\matlab\toolbox\simulink\...

Validation result

```
Last validation succeeded.
```

Pseudocode preview

```
Header file:
No header file is specified. By default, data is
exported via the generated model.h file.

Type definition: Not applicable.

Declaration:
/* CSC declaration comment generated by default */
extern DATATYPE DATANAME[DIMENSION];

Definition:
/* CSC definition comment generated by default */
DATATYPE DATANAME[DIMENSION];
```

Contents of: Base Workspace

	Name	Value	DataType
	Param	[]	auto

Simulink.Parameter: Param

Value: []
Data type: auto
Dimensions: [0 0] Complexity: real
Minimum: -Inf Maximum: Inf
Units:

Code generation options
Storage class: Auto
Alias:

Description:

Fig. 5 Data class designer

either inheriting properties from existing CSC or creating a new CSC. The former choice configure CSC designer step by step, while the later choice requires knowledge of TLC programming. To make it east, template TLC files are provided to help developer writing custom TLC file for CSC. After having created the TLC file, register this file CSC designer and apply CSCs to parameters and signals, so it can generate data structures as the TLC file describes.

4 Code Analysis and Verification

4.1 Designing Real-Time Scheduler

Matlab code generation structure can be divided into three main parts—core algorithm, hardware support and real-time scheduler. The core algorithm is composed of self-carried Simulink blocks and TLC controls how to translate user's configuration into core algorithm code. Hardware support works in the similar way but using custom GUI blocks and TLC files. Having developed driver blocks in Simulink library, designer can build vehicle control algorithm and configure peripheral device drivers all in one model without any handwritten work. The middle block in Fig. 6 is a subsystem describing electric vehicle controller core algorithm and the sideward blocks consist of MPC5644A CAN, AD, I/O blocks. The eQADC blocks are used to collect analog signals, such as accelerator pedal position, brake pedal position. I/O blocks gather digital signals like gear state, warning level and so on. The CAN Receive block receives can messages from Auxiliary Power Unit (APU) controller, battery controller, motor controller. These data are calculated in vehicle control algorithm and new control commands are sent back to each controller through CAN Transmit block. Time Trigger Controller Area Network (TTCAN) protocol plays an important role in this distributed

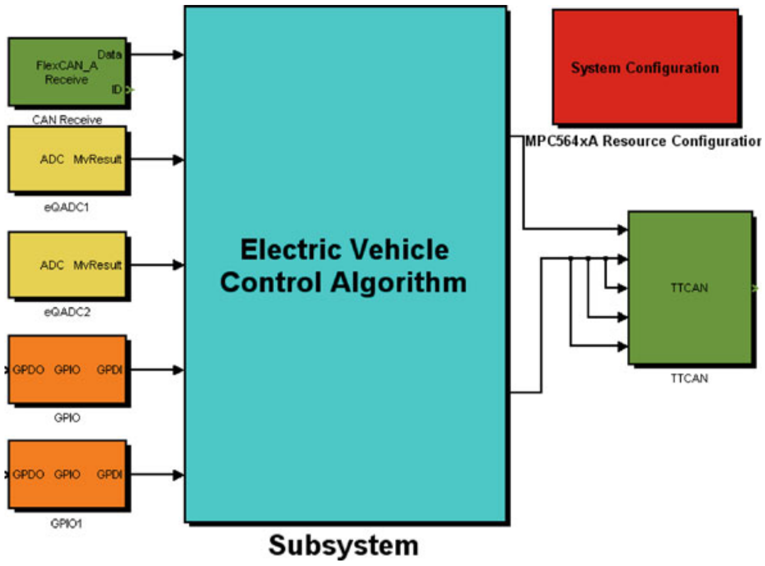


Fig. 6 Model of electric vehicle control unit

communication network. TTCAN improves real-time and accuracy, but the design of the network are always complicated projects, let alone automated code generation process.

In the application of electric vehicle controller, in order to guarantee TTCAN transmission delay less than 100 μs, two Periodic Interrupt Timer (PIT) channels are used in this model to fulfil this scheduling mechanism. The PIT with lower priority is attached to model's base sample time step function "rt_OneStep". ADs or other drivers' sample times are integral multiple of base sample time so they run at each sub-rate. To do this, a function called "rate_scheduler" is created to compute which sub-rates should run during the next base time step. Sub-rates are an integer multiple of the base rate counter. Therefore, the subtask counter is reset when it reaches its limit and subtask starts running. Sample time offsets are handled by priming the counter with the appropriate non-zero value in the model's initialization function. For example, the electric vehicle algorithm here is executed every 0.01 s, eQADC channel one samples every 0.04 s, and eQADC channel two samples every 0.08 s after 0.3 s time delay. The PIT with higher is assigned to CAN transmission blocks. It can interrupt model algorithm to transmit CAN messages at once which guarantee network real-time. In this model, three messages "VCU_C_1", "VCU_D_1", "VCU_S_1" are transmitted at different rates following TTCAN protocol, with transmission delay less than 10 μs.

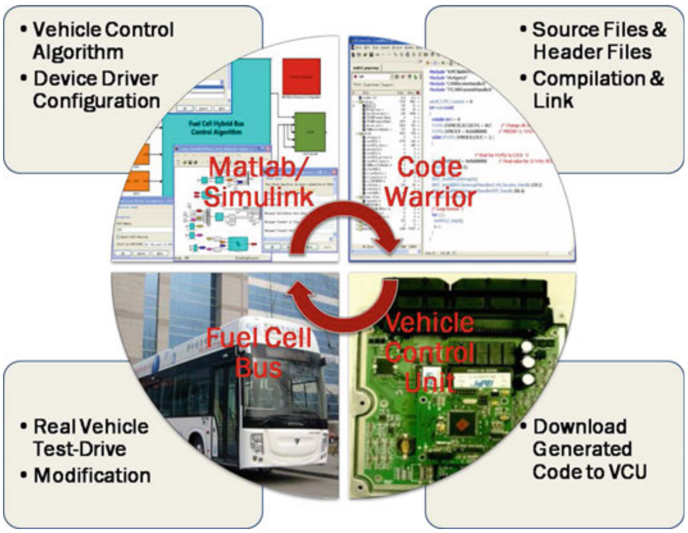

• Vehicle Control Algorithm
• Device Driver Configuration
• Source Files & Header Files
• Compilation & Link
Matlab/ Simulink
Code Warrior
Fuel Cell Bus
Vehicle Control Unit
• Real Vehicle Test-Drive
• Modification
• Download Generated Code to VCU

Fig. 7 Model of electric vehicle control unit

4.2 Analyzing Generated Code

Click "generate code" button, RTWEC transforms electric vehicle model into intermediate rtw file first and then generates several source files. [Model].c file is closely related to the control algorithm and hardware drivers. Every segment code is related to a block and controlled by block TLC. This modularity characteristic makes the code easy to read and modify. In Simulink model, blocks are connected through signal lines. Each block has input and output ports, so there are corresponding block I/O variables' definitions in the generated code. A local block I/O variable can be an input to one block but an output to next block following the signal line. Initialization function in this file initializes what the model needs at the beginning and hardware registers as well. Step functions describe electric vehicle control algorithm block by block and are generated according to block sample times. Besides, there are special functions such as interpolation function because this model supports fixed-point operation. [Model]_main.c file contains model main function called "rt_OneStep". This key function runs base rate task and then call "rate scheduler" function to judge which step function should be executed. Generally, step zero function is attached to base rate sample time, while block TLCs generate code into other step functions if their sample times are integral multiple of base rate. Rate scheduler function is called by PIT interrupt every base sample time to decide which sub-rate task should run. [Model]_data.c file is used to store all blocks signals and parameters. How these data declared and stored are controlled by custom storage classes mentioned above. MPC5644A_drivers.c file includes hardware related functions such as "AD_convert", "CAN_transmit", "CAN_receive".

5 Conclusion

This paper presents a technical route to develop an embedded target—MPC5644A and use it in the development of electric vehicle controller, see Fig. 7. Automated code generation frees programmers from handwriting driver code and lets them focus on the design of vehicle control algorithm. Experimental results show that the R&D time and cost can be drastically reduced. Besides, automated generated codes are easy to maintain and transplant, but the memory space it takes is a little bit bigger. The fuel economy and the vehicle dynamic performance of electric vehicle are kept in the similar level compared to traditional methods because the control algorithm does not change.

Acknowledgments Supported by the NSFC (National Natural Science Foundation) of China under the contract of No. 61004075, the MOST (Ministry of Science and Technology) of China under the contract of No. 2010DFA72760 and No. 2011AA11A269, and the Tsinghua University Initiative Scientific Research Program (Grand No. 2010THZ08) is greatly acknowledged.

References

1. The Mathworks, Inc., Supported hardware, http://www.mathworks.org
2. Microchip Technology, Inc., MATLAB device blocks for MATLAB IDE, http://www.microchip.com
3. Hercog D, Curkovic M, Jezernik K (2006) DSP based rapid control prototyping systems for engineering education and research. In: Proceedings of IEEE international symposium on computer aided control systems design, Munich, Germany
4. Rebeschiess S (1999) MIRCOS—microcontroller-based real time control system toolbox for use with Matlab/Simulink. In: Proceedings of IEEE international symposium on computer aided control system design, pp 267–272
5. Duma R, Petreus D, Sita VI, Dobra P, Rusu A (2010) Rapid control prototyping toolbox for Renesas M32C87 microcontroller. In: Proceedings of IEEE international symposium on automation quality and testing robotics (AQTR), pp 1–6
6. Rusu C, Radulesvu M, Balan H (2007) Embedded toolbox for F24X DSK. International Aegean conference on electrical machines and power electronics, Aegean, Greek, pp 556–559
7. Bartosinski R, Hanzalek Z, Struzka P, Waszniowski L (2007) Integrated environment for embedded control systems design. In: Proceedings of IEEE international symposium on parallel and distributed processing (IPDPS), pp 1–8
8. Feng L, Zhihui H (2010) Embedded C code generation and embedded target development based on RTW-EC. In: Proceedings of IEEE international symposium on computer science and information technology (ICCSIT), vol 5, pp 532–536