Proc. of the Intl. Conf. on Advances In Engineering And Technology - ICAET-2014

ISBN: 978-1-63248-028-6 doi: 10.15224/ 978-1-63248-028-6-01-101

Model Based Development of Control Strategy for Fuel Injection Management and Production code Generation

VP. Arvind Raj
M.Tech Automotive Electronics
Veltech Technical University
Chennai, India

M. Sivakumar
R&D
UCALFuel Systems Ltd.
Chennai, India

R. Sridhar
R&D
UCALFuel Systems Ltd.
Chennai, India

G. Sasikala
Department of Electrical Engineering
Veltech Technical University
Chennai, India

Abstract— ***Conventionally, the control strategy of a Fuel Injection Management system is created in the form of hand written Embedded C codes; this cannot be understood by the personnel of other disciplines working on the system. In this paper, the methods to implement model based design for the same control strategy along with production code generation has been discussed, along with the benefits of the model based design adaptation.***

Keywords- ***Model Based Design; Auto Code Generation; Legacy Code Tool; Fuel Injection Management.***

I. INTRODUCTION

In Automobile Sector, Control System for applications plays a major role in performing the intended task of those applications. Basic architecture of all these control systems consists of hand written embedded C codes. On the contrary, in an automobile sector control system alone can't perform everything; it requires support from various other factors engineered by different departments, i.e. calibration, data acquisition, measurement etc..,. So construction of control strategy that governs the working of the control system should never be a standalone process, which it is at present.

Vast inputs and feedbacks from various departments along with the predefined requirements, intends to create a large complex control strategy and more space for human errors. To avoid this we need to create a common environment/platform where all these departments can work, and communicate effectively between the different departments to avoid the errors at a much early stage of the project. Hence the high error rectification costs during the final phases of the project will be reduced.

Model Based Design using Simulink provides that common platform, to work and implement the same control strategy in the form of mathematical models using the blocks available in the library of Simulink. Once the modeling is completed, it can be configured to generate C code with the help of Real Time Workshop Embedded Coder available in the Simulink. Lack of Application Programming Interfaces and their supporting files can be covered, by using the legacy code tool and custom code addition.

II. CONTROL STRATEGY FOR FUEL INJECTION MANAGEMENT

A. Engine State Definition

It determines the manner in which the various system strategies operate at any particular time. This defines the engine states available within the engine management system and defines the rules for the transition between each of these states. Different engine states are; power-off, stall, cranking, idle, running, wide-open throttle and over-run cut as in Fig.1. Generally the transitions between these states are influenced by the engine rpm and Throttle position, which is obtained from the crankshaft position sensor and Throttle position sensor.

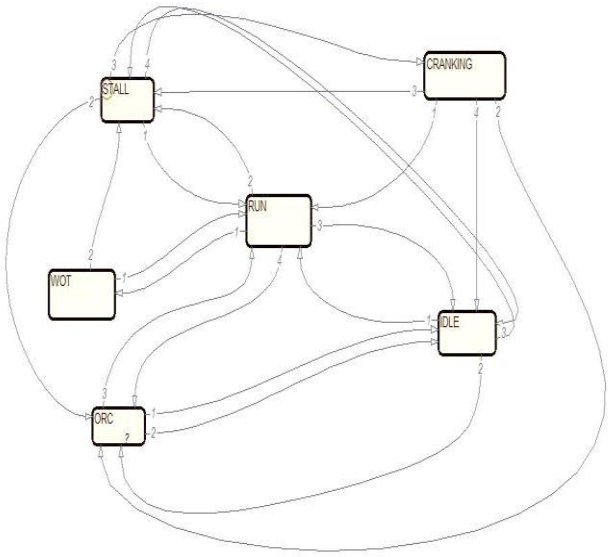

Fig.1 Engine State Definition

Proc. of the Intl. Conf. on Advances In Engineering And Technology - ICAET-2014

ISBN: 978-1-63248-028-6 doi: 10.15224/ 978-1-63248-028-6-01-101

B. Phase Recognition

It determines which phase of the engine is on for a four stroke single cylinder engine i.e. induction/compression or expansion/exhaust. This information is used to allow the engine management system to run the fuel injector; ignition and idle air control valve sequentially i.e. one event per cycle. This strategy examines the encoder tooth period over a 90 degree crank angle segment prior to and immediately after physical engine Top Dead Centre on a revolution-by-revolution basis. Phase of the engine is determined using the speed fluctuations that occur due to the firing and non-firing revolution of each cycle.

C. Determination of Airflow

Intake air flow is calculated, based on the temperature and pressure of the Intake Air, which is measured by the sensors. From those values Air per cylinder per cycle value can be calculated by using the lookup mechanism.

Airflow = (Manifold Pressure * Airflow Volume Correction)/ ((273+Intake Air Temp.)*Atmospheric Pressure) (1)

Air per Cylinder per Volume = (Volumetric Efficiency(in %) * Cylinder Volume(in ml)) / R (2)

Where, R = Universal Gas Constant = 0.287 (KJ/Kg °K)
Temp. = Temperature

D. Determination of Fuelling

The base fuelling value is determined based on the requested Air Fuel ratio and the Air per cylinder per cycle value (determination of Airflow). Trims are applied to the base fuelling based on the different engine states to calculate the optimum amount of fuel required. Closed loop fuelling offsets were applied to perturbate the Air Fuel Ratio about the stoichiometric Point (14.7) and a fuelling adaption value is applied so that this perturbation occurs around the zero value.

III. MODEL BASED DESIGN

Model-Based Design is a process that enables faster, more cost-effective development of dynamic systems; including control systems, signal processing and communications systems. In Model Based Design, a system-model is at the center of the development process, from requirements development, through design, implementation, and testing. The model is an executable specification that is continually refined throughout the development process.

A. Modeling

The process consists of six steps to model any system, they are: *Defining the system, Identifying system components, modeling the system with equations, building the Simulink block diagram, running the simulation, validating the simulation results.* The pseudo code of the above mentioned control strategies are converted into system components and are represented using simple mathematical equations and then modeled with the help of Simulink blocks. For example, the following equation used for filtering,

$SampFilt_n = Sample + (((SampFilt_{n-1} – Sample) * SampFiltC)/256)$. (3)

Is modeled as in Fig.2,

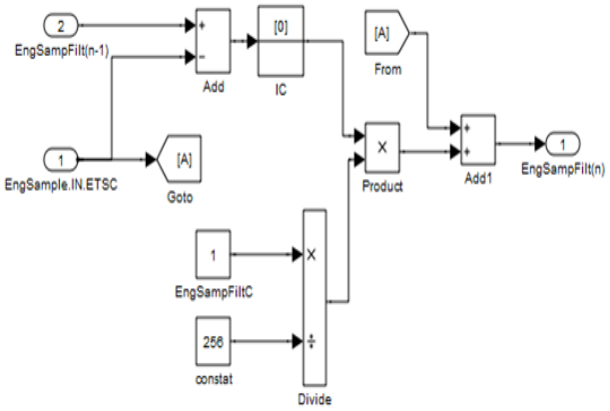

Fig. 2 Modeling of Equation

Sample Time: 0.01 and data types generally used were double, uint16 and int16.

B. Stateflow

It is an interactive, graphical design tool for developing and simulating event-driven systems based on finite-state machine theory. Fig. 3 is an example of a Stateflow chart. Stateflow has been used in order to determine the engine states at various scenarios. It is an event-driven process, hence Stateflow has been adopted. It has two types of states, Exclusive state and parallel state, in this paper I have used exclusive states, based on the exclusivity of the events that are controlled. “en:” in the Fig.3 denotes the Entry condition that is assigned as soon as the control moves to “EngineState”. And the arrows are used to denote the transition from one state to another, which occurs if the condition mentioned inside the [ ] is satisfied. Likewise, the entire enginestate has been modeled using the same methodology.

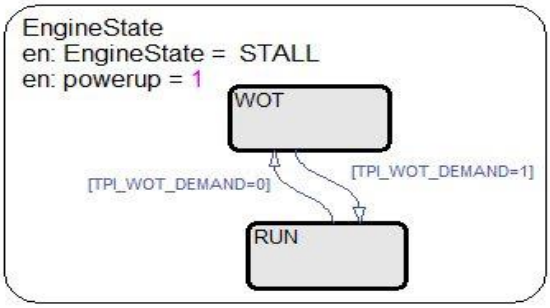

Fig.3 Portion of Stateflow Chart

C. Legacy Code Tool

It is used to migrate a certain C codes like APIs and I/O interfaces to Simulink Environment via MATLAB. It is a set of commands as in Fig.4, which has to be typed in the

Proc. of the Intl. Conf. on Advances In Engineering And Technology - ICAET-2014

ISBN: 978-1-63248-028-6 doi: 10.15224/ 978-1-63248-028-6-01-101

command window of MATLAB and finally a S-Function block[4] will be created inside the Simulink window.

```
legacy_code('help')
specs = legacy_code('initialize')
legacy_code('sfcn_cmex_generate', specs)
legacy_code('compile', specs, compilerOptions)
legacy_code('generate_for_sim', specs, modelname)
legacy_code('slblock_generate', specs, modelname)
legacy_code('sfcn_tlc_generate', specs)
legacy_code('rtwmakecfg_generate', specs)
legacy_code('backward_compatibility')
```

Fig.4 Legacy code Tool Commands

For creating the s-function block, the following commands where typed in the command window of the MATLAB.

```
def = legacy_code('initialize')
def.SourceFiles = {'doubleIt.c'}
def.HeaderFiles = {'doubleIt.h'};
def.SFunctionName = 'ex_sfun_doubleIt';
def.OutputFcnSpec = 'double y1 = doubleIt(double u1)';
legacy_code('sfcn_cmex_generate',def)
legacy_code('compile',def);
legacy_code('slblock_generate',def);
```

Typing the above command in the MATLAB command window will generate a S-function block as in Fig.5, the block performs a Multiply by 2 operation, to the input given.

IV. AUTO CODE GENERATION

A. *Real Time Workshop Embedded Coder*

It is an add-on product of Simulink used for embedded software development. After completing the modeling of the control system, we have to configure some parameters in order to generate an efficient C code. In the model window, click on the simulation tab, and then click the "configuration parameters" option. The configuration window opens, in this window go to solver menu and select "Fixed-step solver" in the "Solver Options > Type" option. In the side window, choose "Hardware Implementation" and select the target specific microcontroller family from the given options.

B. *Code Generation Advisor*

In the Real-Time Workshop window in configuration parameters, select "ert.tlc" under the system target file option. Then set the Objective in the "Select objective:" option. You get to have 4 different objectives, they are: efficiency, traceability, debugging and safety precaution. Out of these four options, choose "efficiency" in order to generate a memory efficient code. Once you have selected the Objective. Click the "Check model" button, without changing any other options. And this opens the code generation advisor, based on the objective the code generation advisor provides the options, which are all to be used. And also provides an option for modifying all parameters at the click of a button.

C. *Custom Code*

It is an option available inside the configuration parameters of the model. Under the Real-Time Workshop dropdown menu, we have a tab called as "custom code". Include the path of all the necessary source and header files in the "Include list of additional" option and no need to put any extension, the Real time Workshop embedded coder derives the file type and memory allocation based on the initial statements of a code file. Thus we could include the required source and header files inside the auto generated code.

Now after configuring all the options in the configuration parameters window, now go back to the system model move away the source and sink blocks from the calculation blocks. Now select all the blocks in the model except the source and sink blocks by clicking and dragging the cursor. Now right click on any of the selected blocks, in the drop down menu, click on the option named "Real-Time workshop", a side menu opens up in that menu click on the "Build" option. It generates the auto-code with necessary c functions and supporting source and header files linked to the code.

V. TESTING

The Auto Code Generation creates a set of source and header files inside the present working directory, which has to be added into the corresponding Header and Source file folders of the Fuel Management System along with the other files. These files have to be added inside the Keil project as well. And a new function has to be created known as Placeholder Function [3], inside the legacy code. And it has to be called from the Scheduler, the placeholder function acts as an interface between the Legacy code and Auto-generated code. In order to understand about the placeholder function, refer to [3]. Once the process of creating a placeholder function is done, we can continue with the testing of the code. We used the Keil Ulink Debugger and Vector CANape along with CANcase XL in order to monitor the proper working of the generated auto-code. Result obtained during the PIL testing is depicted in Fig.6

Proc. of the Intl. Conf. on Advances In Engineering And Technology - ICAET-2014

ISBN: 978-1-63248-028-6 doi: 10.15224/ 978-1-63248-028-6-01-101

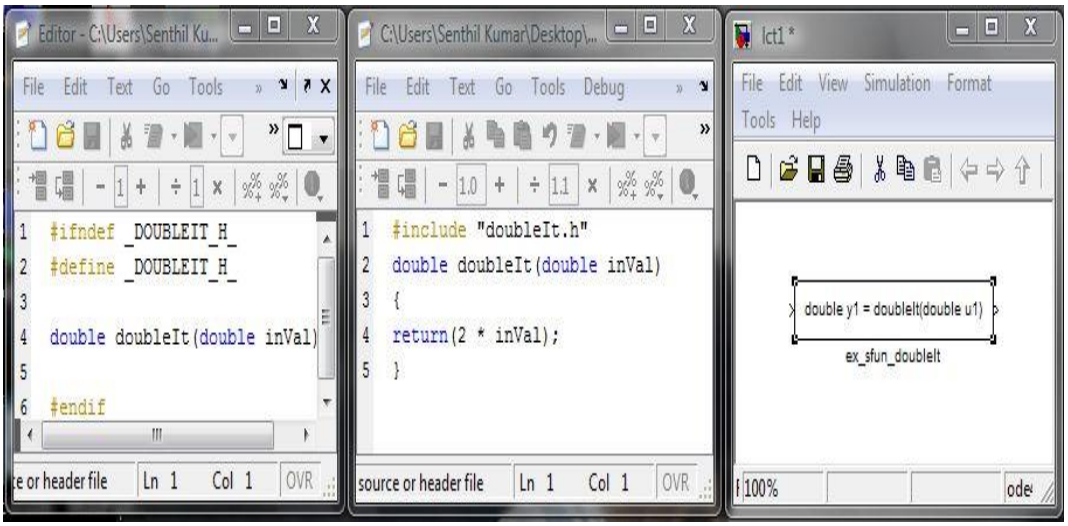

Fig.5 Legacy code tool example

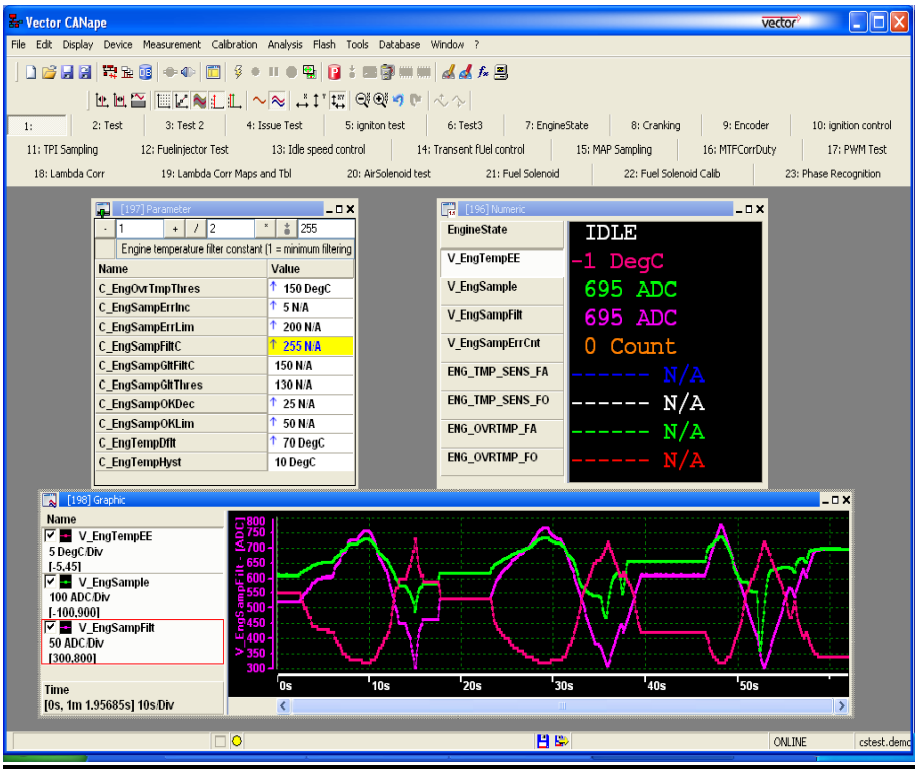

Fig.6 Vector CANape result for Engine Oil Temperature Sensor.

VI. CONCLUSION

Thus by creating a Model Based Design of the control strategy for fuel injection management, a common design environment has been created and configured for generating production ready codes. The Level of Interdepartmental dependency was reduced. And the overall time required to complete a project has been considerably reduced.

VII. ACKNOLEDGMENT

I would like to thank Mr. M. Sivakumar, R. Sridhar and G. Sasikala for their valuable guidance and for their support throughout the project. I would like to express my gratitude to Mr. Nandu K. Sreevalsan, Mr S.Bharanitharan and Mr M.B. Prem Anand for spending their valuable time and sharing their knowledge, which helped me to complete the work.

ISBN: 978-1-63248-028-6 doi: 10.15224/ 978-1-63248-028-6-01-101

REFERENCES

[1] Jeffrey M. Thate, Robert A. Kagy, Robyn A. Jackey, Roger Theyyunni, Jagadish Gattu. *Methods for Interfacing Common Utility Services in Simulink Models used for Production Code Generation*. SAE International 2009; 2009-01-0155.

[2] Tom Erkkinen, Scott Breiner. *Automatic code generation – Technology Adoption lessons learned from commercial vehicle case studies.* SAE International 2009: 2009-01-4249.

[3] Jinming Yang, Sumithra Krishnan, Jason Bauman, Al Beydoun. *Implementation of Auto-code generation in legacy code for body control software applications,* SAE Internation 2008: 2008-01-0749

[4] Information on http://www.matlabcentral.com.

[5] Information on MATLAB Software Help Search Engine.