

Ford Motor Company


SmartVehicle Baseline Report

In support of the University of California Open
Experimental Platform for DARPA – MoBIES,
Contract F33615-01-C-1841.

Trademark and copyright Notices:

- TargetLink is a trademark of dSPACE Inc.
- Embedded Coder is a trademark of The MathWorks, Inc.
- MATLAB, Simulink, Stateflow, and Real-Time Workshop are registered trademarks of The MathWorks, Inc. Embedded Coder is a brand of The MathWorks, Inc.

Conventions used within this document:

- Informational messages (visible on command line, pop-up windows etc.) and input command messages (user input strings, selection items, etc.) will be indicated by `courier font`.
 - **Bold font** will indicate Simulink, Stateflow, and Targetlink block names, signal names, system, and subsystem names.
 - Parameter values, variables, and constant values will be indicated by *an italicized font*.
- 

Overview

This report is a companion to the previously released SmartVehicle Challenge Problem Report [\[1\]](#). The challenge document defined automotive powertrain problems to be used to assess the effectiveness of new tools and methods for model-based embedded software development. The purpose of this report is to assess current tool capabilities with respect to challenge problem areas of interest. Specifically, baseline evaluations have been carried out for hybrid system analysis and automatic 'C' code generation tools.

Methodology

For each of the baseline tools reviewed, we have developed an example problem derived from the original University of California at Berkeley models [\[2\]](#). The UCB models are annotated with a data dictionary that defines the input and output signals and then processed by the appropriate tool.

For each tool reviewed we will answer the following questions:

- 1) Why was it chosen?
- 2) What does it do well?
- 3) What does it do poorly?
- 4) What does it not do at all?

Each tool is measured against the challenge problem criteria to assess the breakthrough opportunities.

The original models were developed in MATLAB Simulink/Stateflow, and the baseline tools we have chosen to evaluate are integrated, to a greater or lesser degree with the at environment. This is in keeping with the MoBIES goals to demonstrate both breakthrough tools and tool integration..

Evaluations are based on MATLAB 6.0 version as released in September, 2000. All the companion tools are compatible with this version. It is recognized that new versions of the MathWorks toolset will be released during the course of this evaluation. Nonetheless, the intention is to maintain consistency of the baseline by assessing the MoBIES-developed tools against this standard.

Data Dictionary for MOBIES models

In order to support the various baseline experiments, a dictionary function was required to provide more information regarding signal attributes than is normally imparted by a Simulink diagram. For example, many parameters used in the models, tables of values or single multipliers, need additional information for code generation purposes. To resolve that problem, a simple data dictionary function was developed based on the Simulink data

object. The MathWorks has indicated that a data dictionary function will be implemented for Simulink, so this is a simple interim solution to the problem.

The data dictionary supports two data types: one is the signal type and the other is the parameter type. Signals would include all data passed by a signal line on a Simulink diagram between blocks, and can be either control information, (such as triggers), or data. The parameter type is typically a data value stored in the MATLAB workspace and used in the model. These data tend to be static and do not change during the course of a simulation.

The signal data object has the following key attributes:

Name: same as name displayed on the Simulink diagram

Description: a brief text description

Unit: the units for the signal, e.g. kPa, Degrees, revolutions per minute.

Floating Point Target: data type for floating point processors, single or double precision

Fixed Point Target Type Signed: 1/0 to indicate if signed number

Fixed Point Target Type Bytes: number of bytes required to store number

Fixed Point Target Type BinPoint: number of bits used to store fractional portion of number.

Size: size of Matlab workspace used to store number

Min – Max: the minimum and maximum values allowed for this number

The parameter data object has the following key attributes in addition to the signal object attributes:

DefaultValue: the value of the parameter at startup.

If the parameter is a 2- or 3-dimensional table, the following are available for x, y, and z values:

FloatingPointTargetType.(x,y,z)

FixedPointTargetType.Signed.(x,y,z): 0 or 1 for unsigned or signed integer

FixedPointTargetType.Bytes.(x,y,z): number of bytes needed to store element

FixedPointTargetType.BinPoint.(x,y,z): number of bits used to represent fractional value

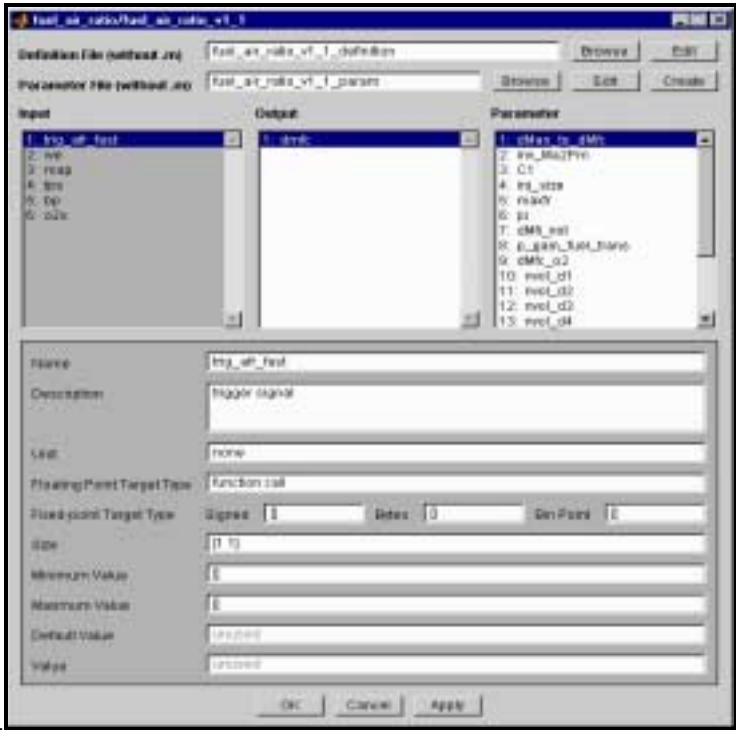
Min.(x,y,z): minimum value allowed

Max.(x,y,z): maximum value allowed

DefaultValue.(x,y,z): default values

The data dictionary information is stored in M-script files and can be modified using the MATLAB editor. For an example of how the data dictionary might be used, we can look at the Air Fuel Ratio control model released as part of our challenge problem update and used later in the code generation section.

Along with the model file, fuel_air_ratio.mdl, there are two other files for the signal and parameter definitions. They are fuel_air_ratio_v1_2_definition.m, for the data dictionary definitions for signals and parameters, and fuel_air_ratio_v1_1_param.m for the parameter data. The actual parameter values may be something other than the default values and will likely change during the course of the model's development. When the model is loaded in the Simulink environment, these files are loaded into the workspace. To access the loaded definitions there are numerous utilities which are documented in Appendix A. To see that the definitions are loaded, the user can right click on the subsystem mask and select Mask Parameters. A GUI display of the information will appear as seen in Figure 1.



• Figure 1 Data Dictionary User Interface

Dynamic analysis of hybrid system models for design validation

The report [Dynamic Analysis of Hybrid Systems Models for Design Validation](#) states the automotive powertrain requirements and briefly reviews candidate tools for hybrid system analysis, resulting in the selection of CheckMate for further evaluation. In this baseline report, issues of conversion to the CheckMate modeling language and fidelity of the

converted model with the original challenge model are addressed. A subsequent evaluation report will address problem analysis capabilities of CheckMate.

From the challenge problem document, tool requirements for hybrid system analysis are:

1. Simulation capability
2. Admissibility of general hybrid system models
3. User friendliness
4. Automation of analysis
5. Computational efficiency

The CheckMate tool was evaluated against these requirements by application to an automotive automatic transmission shift control problem defined in [1]. Specifically:

Given the powertrain model with constant throttle position and road grade inputs, will the system oscillate between first and second gears as the vehicle accelerates from zero to 100 km/hour?

The main difficulties encountered with CheckMate are associated with not satisfying requirement number 2; that is, the analysis requires conversion of the original Stateflow / Simulink model into the CheckMate environment. Nonetheless, a simulation comparison between the original and the CheckMate model (derived after not inconsiderable effort), showed good agreement. Specific modeling difficulties include:

1. Complex guard conditions as a result of CheckMate's insistence that they be functions only of continuous variables
2. Increased system order
3. Loss of modeling hierarchy

Automatic Generation of Code from Models

The report [Embedded Software Analysis and Generation](#) evaluates the dSPACE TargetLink and MathWorks Real Time Workshop with Embedded Coder automatic code generation tools against challenge problem requirements for two models: the fuel-air ratio and transmission shift control problems represent a substantial portion of the Simulink and StateFlow modeling environments.

The criteria for code generation as described in [1] are:

1. ANSI C (stated compliance)
2. Preserved Functionality (Simulink)
3. Preserved Functionality (Stateflow)
4. ROM / RAM / CPU Efficiency
5. Traceable / Readable
6. System Independent Functions
7. Automated Usage
8. Discrete Time Support
9. Permits Calls to/ from User Code
10. Variable Name Control
11. Variable Type Control
12. Type Qualifiers
13. Storage Class Control
14. Variable Scoping Control
15. Variable Initialization
16. Variable Declaration
17. Function Partitioning
18. Function Prototyping
19. File Partitioning
20. C Structures
21. User Comments
22. Processor Optimization

Neither tool satisfied all requirements for both problems. Nonetheless, the generated code accurately represented the system model in all cases, and both tools represent an exacting standard. It may be noted that automatic code generation using these and other commercially available tools has been a reality for over a decade. The challenge remains to generate efficient, embedded code that is configurable and linkable to legacy code in a production environment.

Appendix A – Data Dictionary

A.1 - Data Dictionary entries for the Fuel-Air Ratio model:

This is not a complete data dictionary. We have included only a single example of each major entry type that would be encountered.

```
% Definition for the air fuel ratio controller.
clear Definition

% Input definitions.
Definition.Input = struct([]);
Definition.Input(1).Name = 'trig afr_fast';
Definition.Input(1).Description = 'trigger signal';
Definition.Input(1).Unit = 'none';
Definition.Input(1).FloatingPointTargetType = 'function call';
Definition.Input(1).FixedPointTargetType.Signed = [];
Definition.Input(1).FixedPointTargetType.Bytes = [];
Definition.Input(1).FixedPointTargetType.BinPoint = [];
Definition.Input(1).Size = [1 1];
Definition.Input(1).Min = [];
Definition.Input(1).Max = [];
Definition.Input(2).Name = 'we';
Definition.Input(2).Description = 'engine speed';
Definition.Input(2).Unit = 'rad/s';
Definition.Input(2).FloatingPointTargetType = 'double';
Definition.Input(2).FixedPointTargetType.Signed = 0;
Definition.Input(2).FixedPointTargetType.Bytes = 2;
Definition.Input(2).FixedPointTargetType.BinPoint = 2;
Definition.Input(2).Size = [1 1];
Definition.Input(2).Min = 0;
Definition.Input(2).Max = 1048;

% Output definitions.
Definition.Output = struct([]);
Definition.Output = struct([]);
Definition.Output(1).Name = 'dmfc';
Definition.Output(1).Description = 'fuel mass flow rate command';
Definition.Output(1).Unit = 'kg/s';
Definition.Output(1).FloatingPointTargetType = 'double';
Definition.Output(1).FixedPointTargetType.Signed = 0;
Definition.Output(1).FixedPointTargetType.Bytes = 2;
Definition.Output(1).FixedPointTargetType.BinPoint = 2;
Definition.Output(1).Size = [1 1];
Definition.Output(1).Min = 0;
Definition.Output(1).Max = 0.02;

% Parameter definitions.
Definition.Parameter = struct([]);
Definition.Parameter(1).Name = 'dMao_to_dMfc';
```

```

Definition.Parameter(1).Description = 'dMao_to_dMfc = 1/AFdes where AFdes = 14.7. This should probably get a
better name like inv_afr_desired';
Definition.Parameter(1).Unit = '';
Definition.Parameter(1).FloatingPointTargetType = 'double';
Definition.Parameter(1).FixedPointTargetType.Signed = 0;
Definition.Parameter(1).FixedPointTargetType.Bytes = 2;
Definition.Parameter(1).FixedPointTargetType.BinPoint = 10;
Definition.Parameter(1).Size = [1 1];
Definition.Parameter(1).Min = 1/14.7;
Definition.Parameter(1).Max = 1/14.7;
Definition.Parameter(1).DefaultValue = 1/14.7;

Definition.Parameter(2).Name = 'inv_Ma2Pm';
Definition.Parameter(2).Description = sprintf(['1/Ma2Pm where\n' ...
' o Ma2Pm = Rair*Tm_init./Vm, manifold mass to pressure conversion\n' ...
' o Rair = 0.287, gas constant for air (kJ/(kg*K))\n' ...
' o Tm_init = 303, manifold temperature (K)\n' ...
' o Vm = 2.7e-3, manifold volume (m^3)']);
Definition.Parameter(2).Unit = 'kPa/kg';
Definition.Parameter(2).FloatingPointTargetType = 'double';
Definition.Parameter(2).FixedPointTargetType.Signed = 0;
Definition.Parameter(2).FixedPointTargetType.Bytes = 2;
Definition.Parameter(2).FixedPointTargetType.BinPoint = 27;
Definition.Parameter(2).Size = [1 1];
Definition.Parameter(2).Min = 0.287*303/(2.7e-3);
Definition.Parameter(2).Max = 0.287*303/(2.7e-3);
Definition.Parameter(2).DefaultValue = 0.287*303/(2.7e-3);

```

A.2 – Data Dictionary access functions

[MoBIES Parameter Management Documentation](#)

The documentation for the Data Dictionary is in HTML form and can be read from any web browser.

References

- [1] [Challenge Problem Report](#)
- [2] http://vehicle.me.berkeley.edu/mobies/powertrain/powertrain_v_2_0.zip
- [3] CheckMate, Carnegie Mellon University, <http://www.ece.cmu.edu/~webk/checkmate/>