Ascola: A Tool for Importing Dymola Code into Ascet

Clemens Schlegel
Schlegel Simulation GmbH
Meichelbeckstr. 8b
D-85356 Freising
cs@schlegel-simulation.de

Reinhard Finsterwalder
University of the Federal
Armed Forces Munich
D-85577 Neubiberg
reinhard.finsterwalder@unibw.de

Abstract

A new tool, Ascola™, is presented for import of Dymola generated C-code of a Modelica simulation model into Ascet-MD. All variable names, types, default values, units and comments of the original Modelica model along with the naming structure are retained during import.

The main focus of Ascola is on delivering a user-friendly tool to aid embedded control system design using Ascet and Modelica. An example will be discussed to illustrate the import process and the use of imported models in Ascet.

1. Motivation

The tools of the Ascet product family [1] support the development of embedded software by model-based design of control and diagnostic functions, simulation, rapid prototyping, and automatic code generation. Ascet-MD provides functions to support the modeling and (offline-) simulation tasks in a structured, object oriented way. Functions like the visualization of the interdependencies in a model or the possibility to edit all implementation details of a variable help the user to focus on the main algorithms. Ascet tools are widely used in the automotive sector.

To check whether the software and the control algorithms under development fulfill all functional requirements of a design project is a core task in all stages of the development process. In the first stages this is mostly done by testing the control algorithms and the control software against a simulation model of the device to be controlled (model-in-te-loop and software-in-the-loop). In later stages hardware-in-the-loop simulations come in. For the Ascet environment Ascet-MIP (Matlab Integration Package) [1] facilitates the import of Simulink [2] models to perform such functional tests. It requires the component Realtime Workshop [2] to generate C-code of the Simulink model, which is then imported in Ascet.

Due to its unique features (most outstanding the object oriented, acausal approach) Modelica [3] is used more and more as a modeling language for mechatronic devices which are typically controlled by an embedded controller. Naturally, developers of embedded control software wish to use also Modelica models in Ascet for functional testing. Using Dymola [4] for processing a Modelica model this can be done with some tweaking via export of the Modelica / Dymola model to Simulink and import of the resulting S-function block in Ascet via MIP [5]. However, this requires a long chain of tools, what makes the development process less robust and more expensive. In this paper we introduce Ascola as a tool for direct import of Dymola generated C-code of a Modelica model into Ascet.

2. Basic Structure of Ascet

A controller is specified (modeled) in Ascet by a set of interdependent elements, modules, classes, processes, tasks, and messages. The algorithms and the configuration of the operating system used are maintained in a project. Projects consist of modules and tasks. Modules contain several processes which are specified within the module and scheduled for execution in the tasks. Modules encapsulate several class instances as objects. Each instance belongs to a single class which contains the related methods. Each module occurs only once in a project, whereas classes may have several instances. Methods differ

from processes by containing arguments and return values. Inter-process communication is done using messages. Messages from different modules are globally available, they are automatically routed according to matching names. All elements are kept in a database to facilitate consistent handling and reuse.

Both classes and modules may be specified graphically using a block diagram editor or textually using the built-in language ESDL (Embedded Software Description Language), or using ANSI-C. All elements of Ascet contain implementation information in order to handle target- or project-specific variants. The implementation consists of the respective data types, default values, ranges and quantization of values and memory-location information (RAM, ROM, flash, etc.). Therefore C-classes and C-modules are always implementation specific and treated accordingly in Ascet. The implementation information is used for the automatic code generation. It's indispensable especially for generating code for specialized embedded targets with fixed-point arithmetic.

An important part of the controller specification is the interface definition of the controller's classes and modules. The interface of a class consists of its public methods, their arguments and return values. The interface of a module consists of its processes. Processes determine the activation of the module's functionalities, but unlike C-classes they do not define inputs, outputs or parameters. Modules communicate and interact via messages and global elements which have to be specified explicitly.

In Ascet's project editor the different processes, tasks and the runtime environment (the built in real-time operating system Ercosek) can be configured. There is also an experiment environment to run, stimulate, calibrate and monitor the generated controller code. The control algorithm may run offline on a PC, in real-time on a dedicated experiment hardware and on several target processors used in today's automotive control units.

3. Ascola

3.1 Overview

According to the described structure of Ascet an imported C-code simulation model matches the properties of a C-module inserted in a project (not in a library), because it is unique, has an encapsulated functionality, is based on full scale floating point arithmetic (what's a target specific implementation detail from Ascet's point of view), and needs a specific task structure and a specific sampling rate, which are defined in a project, not in a module.

Started in an Ascet project editor Ascola first reads in the model specific C code generated by Dymola (dsmodel.c), parses the variable definitions (input, output, parameters, and initial states), and prepares corresponding elements according to the Ascet conventions (data type, unit, comment, etc.). It then displays the basic model properties (number and type of parameters, etc.), and asks the user for model and parameter-update sampling rates and the integration algorithm to choose (if not included in the model [6]).

From this information an Ascet C-module with body, header, and interface definition is automatically generated and inserted in the actual project. In addition the necessary make information (compiler settings, symbol definition, linking of model independent utilities, etc.) is generated. The C-module is organized in four processes (initialization, simulation, parameter update and termination) which are assigned to four dedicated tasks. Now the imported Dymola model code can be used like any other module in an Ascet experiment environment.

After having generated C-code from a Modelica model Dymola is not needed any more neither for the import procedure nor at runtime of the corresponding Ascet module, what's a major advantage compared with cosimulation approaches. More over, Dymola generated code can easily be passed to an Ascet user without disclosure of model details because that code is very hard to read. If the Dymola code is used in an offline experiment in Ascet even a variable step integrator including event detection may be used in that module, avoiding the necessity to tune the simulation model for fixed step integration, what may be a major task if the model is stiff, contains nonlinearities, state events, and / or causality changes (e.g. stick slip, hard stop, etc). This technique of encapsulating the integration algorithm along with the simulation model (as a separate or an inlined algorithm) is beneficial also in other simulation environments like Simulink (see e.g. [7]).

3.2 Mapping of Modelica Models to Ascet

Mapping of Modelica model properties to Ascet comprises mainly two aspects: handling of data types and plugging into the operating system environment.

Modelica's predefined data types (real, integer, boolean) containing values, units and comments fit quite well to according Ascet's data types (Ascet is limited to scalars and arrays in one and two dimensions). Therefore Ascola preserves the Modelica names, units and comments. Ascet's tabulated data types (1D and 2D, several interpolation methods) containing axis and dependent values have no corresponding data type built in Modelica (version 2.2.1, [3]). A user defined Modelica data type would not help, because it's structure can't be traced down in the C-code generated by Dymola.

Input and output are implemented as messages in Ascet. Modelica variables with fixed causality are mapped correspondingly. Because Ascet messages are limited to scalars, fixed causality arrays in Modelica have to be serialized.

Since Ascet does not provide integrators which can be used for C-modules they have to be added by Ascola, if not already included in the model (inline integration [4]). Despite the fixed I/O schedule (performed at a user defined rate) a variable step solver may be used as mentioned in paragraph 3.1. In the current version Ascola provides Euler and DASSL (stiff solver, variable step [8]) as integrators.

Apart form initial and terminal sections Modelica doesn't provide any keyword influencing the execution sequence of the generated code. Therefore there are code parts executed once and other parts executed in a fixed schedule imposed by the controller under test. This scheduling requirements end up with the following task scheme:

- initial task: model initialization (single shot)
- step task: input, output, model evaluation (high sampling rate)
- parameter update (low sampling rate)
- terminal task: model terminal section (single shot)

3.3 Implementation Aspects

Ascola is implemented in C++. In the current version only the PC target is supported which is based on the Borland C++-compiler.

The complete model specific Dymola code is stored in the Ascet database in order to avoid consistency problems. Ascola supports several Dymola versions (6.0 and higher) by providing version-specific header files and corresponding function libraries.

4 Example

Figure 1 shows a simple vehicle model which is suitable for checking e.g. cruise control algorithms. Ascola is invoked from an Ascet project editor (menu Dymola / Import C-code, see figure 2). After having chosen the file to import Ascola shows the main model properties (number and type of parameters, inputs, and outputs) and import defaults (task names and task timings for model code and parameter update) which may be overwritten by the user (figure 3). The different steps of the import procedure are logged in Ascet's monitor window (figure 4). After the code import Ascets main window shows the interface elements of the generated C-code module: names of the Modelica variables (with prefixes in_, out_, and p_ for inputs, outputs and parameters) and the corresponding Ascet properties (type, scope, kind, dependency, etc.), see figure 5. In the OS tab of the project editor the generated task configuration settings are shown (figure 6). From the project editor a simulation experiment may be invoked, figue 7 shows simulation results.

5 Future

For a future version it's intended to support also Ascet's rapid prototyping targets (based on PowerPC [1]). This will facilitate model based control and realtime simulations using Modelica models. Based on user demand it's further planned to make Dymola generated C-code import also available for Intecrio, Etas' universal simulation tool [1].

6 References

[1] www.etasgroup.com

[2] www.mathworks.com

[3] www.modelica.org

[4] www.dynasim.se

[5] C. Schlegel, Kopplung Dymola – Ascet, Technical report for BMW AG. Munich, 2003

[6] Dymola 5.3 User's Manual, Dynasim AB, Lund, Sweden. 2004

[7] J. Köhler, A. Banerjee, Usage of Modelica for Transmission simulation in ZF. Proceedings Modelica 2005, pp. 587-592

[8] Brown, Hindmarsh, Petzold, BDF methods with direct and preconditioned Krylov linear solvers. SIAM J. Sci. Comp.: 15, 6, 1467 (1994) and 19, 5, 1495 (1998)

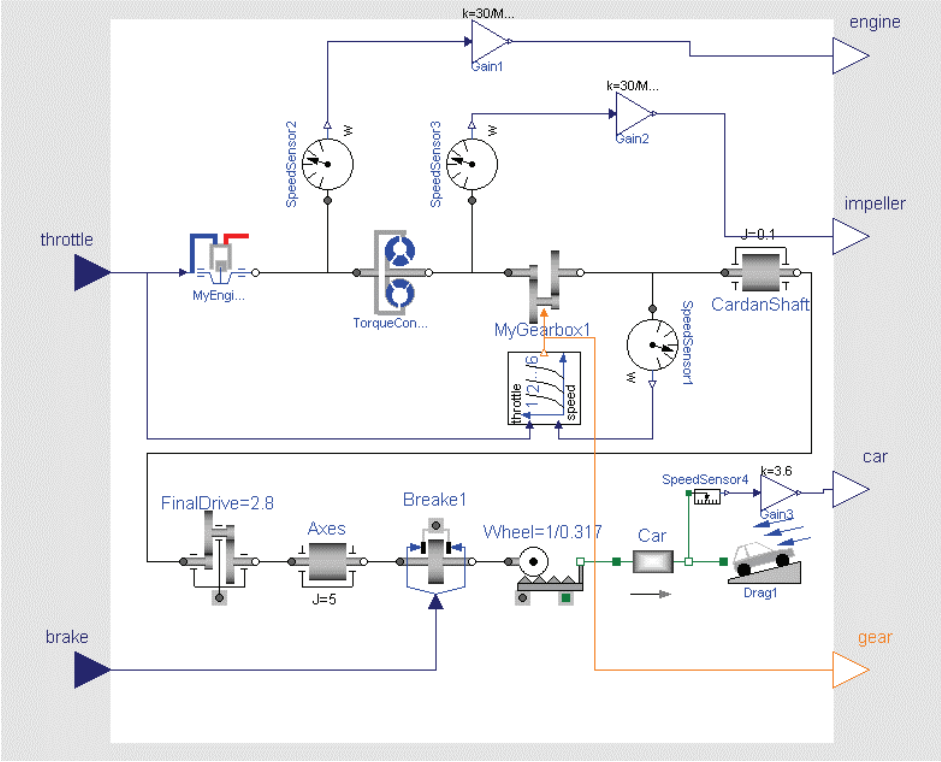

Figure 1: Simple vehicle model for checking cruise control algorithms

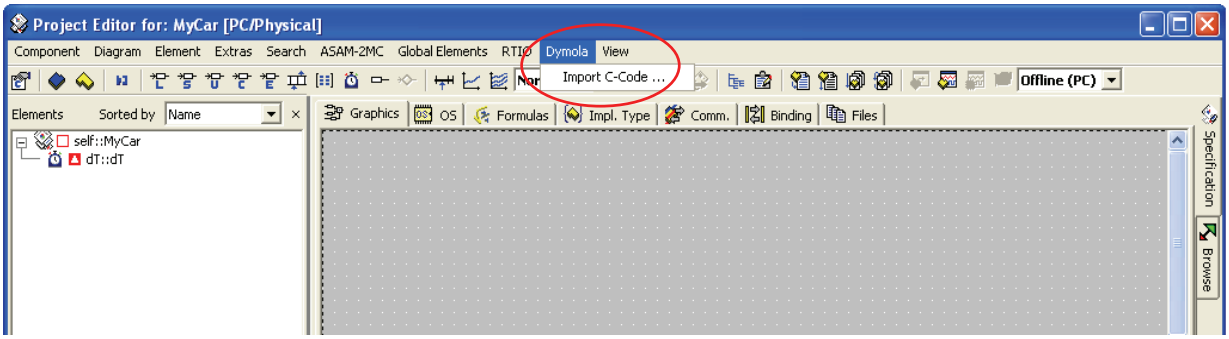

Figure 2: Invoking Ascola from an Ascet project editor

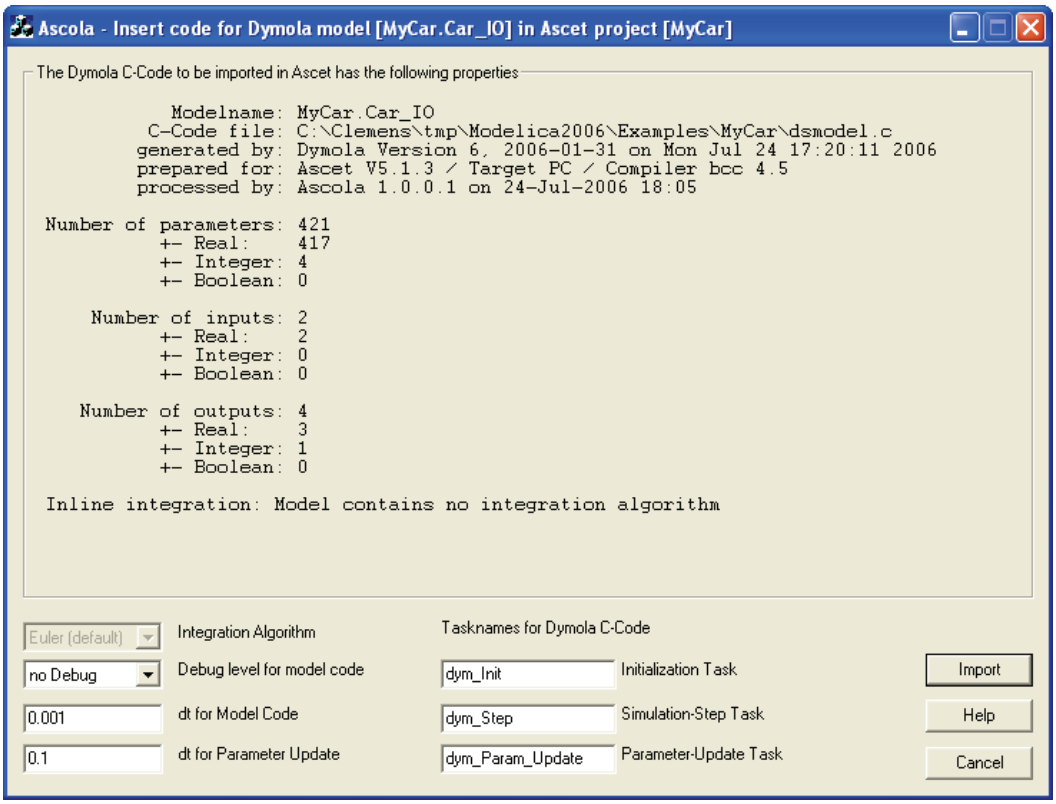

Figure 3: Graphical user interface of Ascola

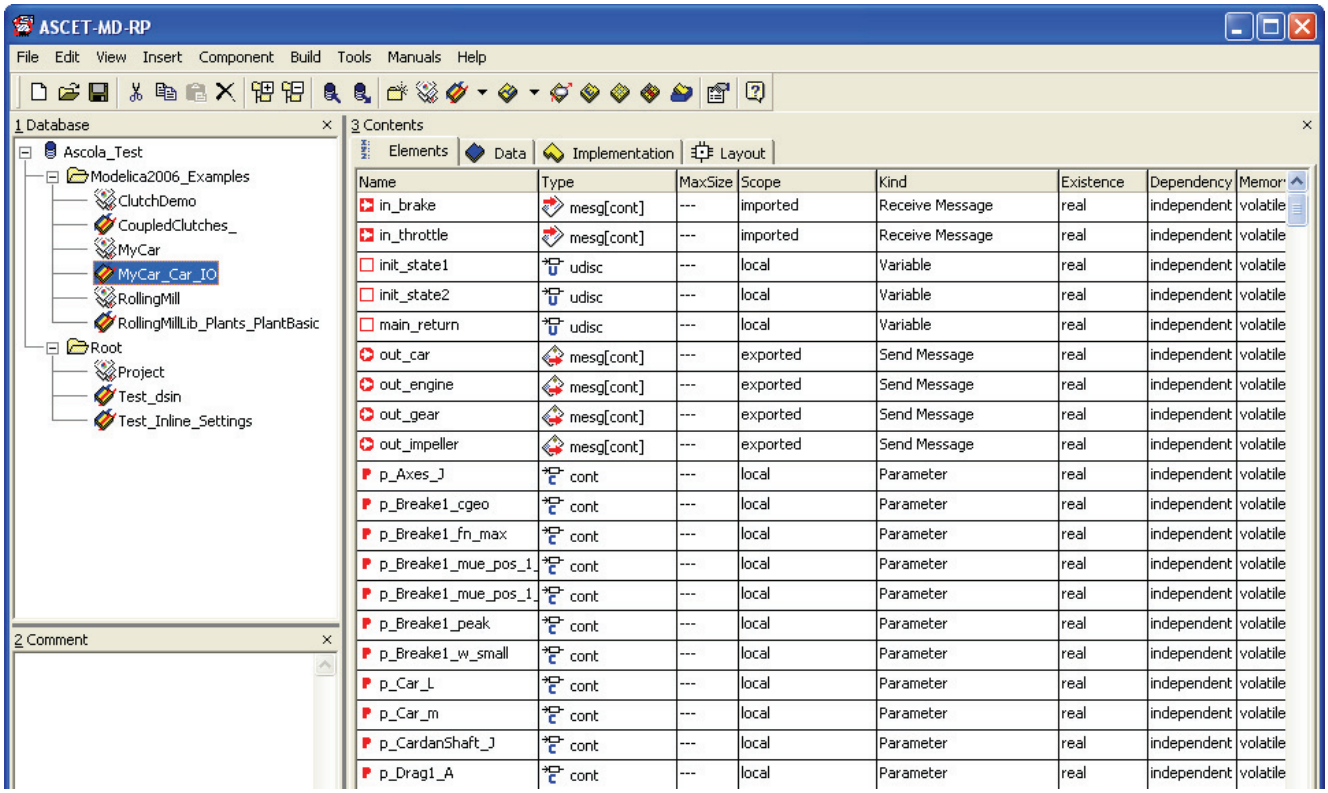

Figure 5: Ascet main window after model import

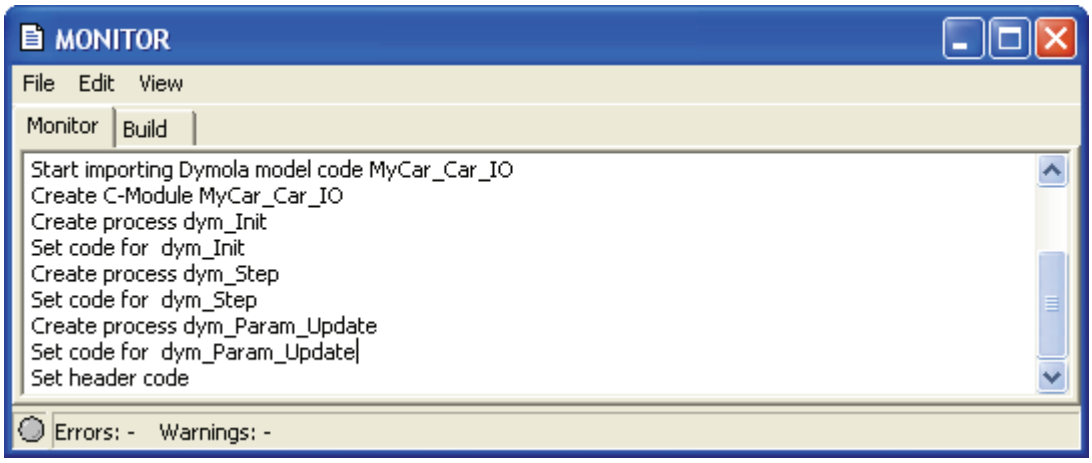

Figure 4: Ascola import messages in Ascet monitor

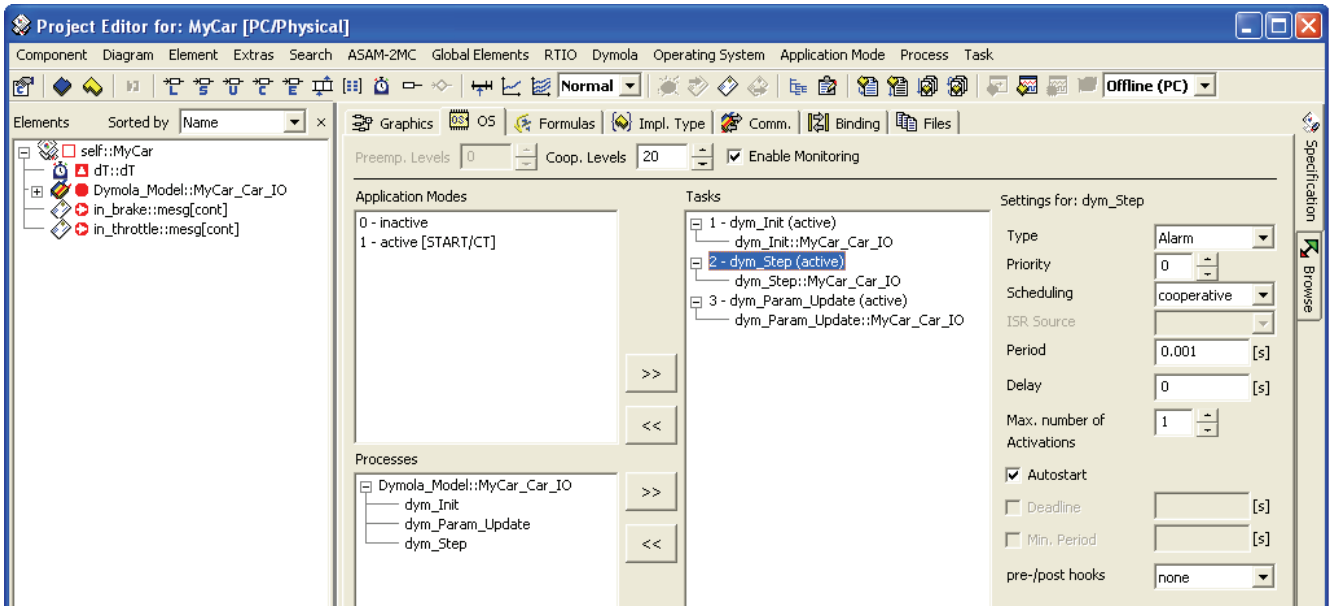

Figure 6: Operating system configuration for imported model

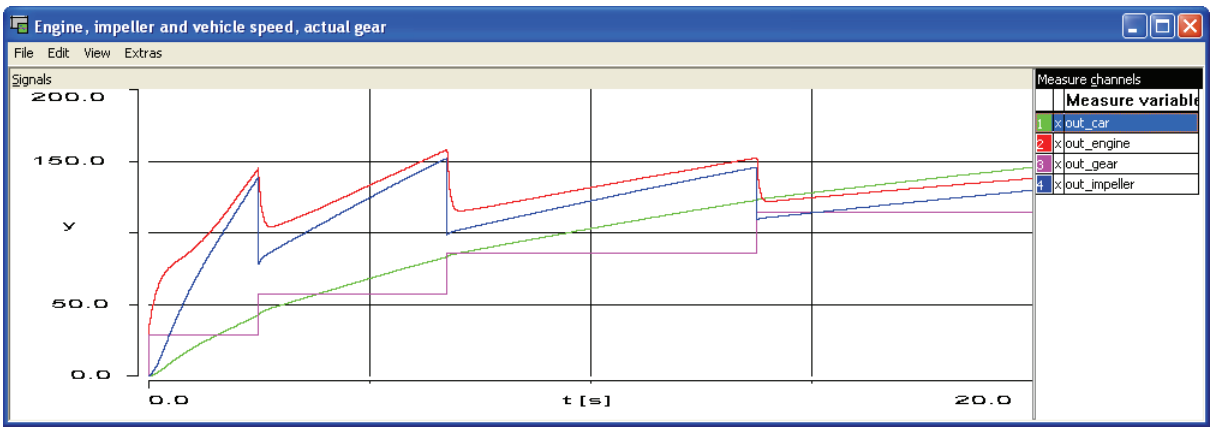

Figure 7: Vehicle acceleration simulation in Ascet (upper: engine and impeller speed, lower: vehicle speed and actual gear)