MODELING AND REALTIME SIMULATION OF AN AUTOMATIC GEARBOX USING MODELICA

Martin Otter and Clemens Schlegel
DLR Oberpfaffenhofen
D–82230 Wessling
Germany
Email: Martin.Otter@DLR.de
Email: Clemens.Schlegel@DLR.de

Hilding Elmqvist
Dynasim AB
Research Park Ideon
S–223 70 Lund
Sweden
Email: Elmqvist@Dynasim.se

KEYWORDS

Variable structure, object oriented modeling, automatic gearbox, hardware-in-the-loop simulation, Modelica.

ABSTRACT

To speed up the development cycle of electronically controlled mechanical devices, hardware-in-the-loop simulation is used more and more. Applying this method to an automatic gearbox of a car, the electronic transmission control hardware is tested against a model of the gearbox, which is simulated in realtime. Modeling such a system is a demanding task, because its structure varies during gearshift. Since the gearbox investigated has six switching elements (clutches and freewheels) the actual structure has to be found amongst $2^6 = 64$ possible configurations. It is shown how such a variable structure system can be modeled with Modelica and how a consistent configuration can be found in realtime after a gearshift occured. The results presented show that it is possible to simulate variable structure systems in realtime using standard realtime simulation hardware and appropriate software tools.

INTRODUCTION

Comfort standards regarding to gearshift of automatic gearboxes of vehicles increase permanently. Gearshift comfort is mainly influenced by control of the switching elements of a gearbox: Clutches and freewheels. Control is performed mostly by an electronic control unit (ECU). Fine tuning of the ECU and the switching elements is therefore essential to optimize gearshift comfort.

In order to speed up the development cycle, hardware-in-the-loop (HIL) simulation is used more and more. For the problem treated, such a setup consists of the ECU-hardware and a realtime simulation of all other components interacting dynamically with it: The whole drive train of engine, hydrodynamic torque converter, gearbox, differential gearbox and longitudinal dynamics of the vehicle.

Modeling such a system is difficult since the structure of the system varies during each gearshift: Depending on the actual control, different wheels and freewheels are engaged or disengaged. The integrator has to be stopped and the new structure has to be determined in accordance to the actual forces imposed on the gearwheels by the switching elements. In the example treated, the automatic gearbox has 6 switching elements leading to $2^6 = 64$ possible configurations of the system.

It is shown how such a variable structure system can be modeled with Modelica (Modelica 1997), a novel unified object-oriented language for physical systems modeling. The Modelica-translator of the simulation environment Dymola (Elmqvist et al. 1996), together with Dymola's symbolic engine, is used to generate efficient code suited for realtime simulation which e.g. requires a state space form of the model. A new technique, which meets the realtime requirements like fixed sampling time, is used to find a consistent configuration after a gearshift. Therefore a hardware-in-the-loop (HIL) simulation of automatic gearboxes becomes possible. The efficiency of the generated code and the sampling frequency necessary to reproduce offline simulation results enables the usage of digital signal processors.

MODEL OF AUTOMATIC GEARBOX

Figure 1 shows an outlined sketch of a typical 4 speed automatic gearbox which is investigated in this paper. It consists of a hydrodynamic torque converter, a standard planetary wheelset, a Ravigneaux wheelset, 6 clutches/brakes, and 2 freewheels (Förster 1991).

Gearshift dynamics can only be simulated if the input- and output-torques of the gearbox represent a real-life vehicle maneuver. Therefore, at least the engine and the longitudinal dynamics of the vehicle has to be modeled. A Modelica model of such a system is given in figure 2 as a composition or object diagram (Otter and Elmqvist 1997) using Dymola's object diagram editor. Every icon in the composition diagram is an instance of a Modelica class and represents a component of the drive train. Mechanical flanges of the components are modelled by Modelica connector classes and are characterized by small square

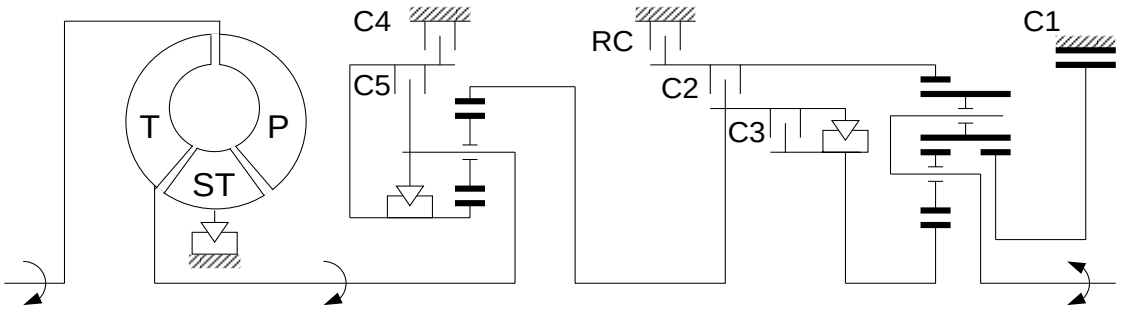


Figure 1: Structure of a typical automatic gearbox.

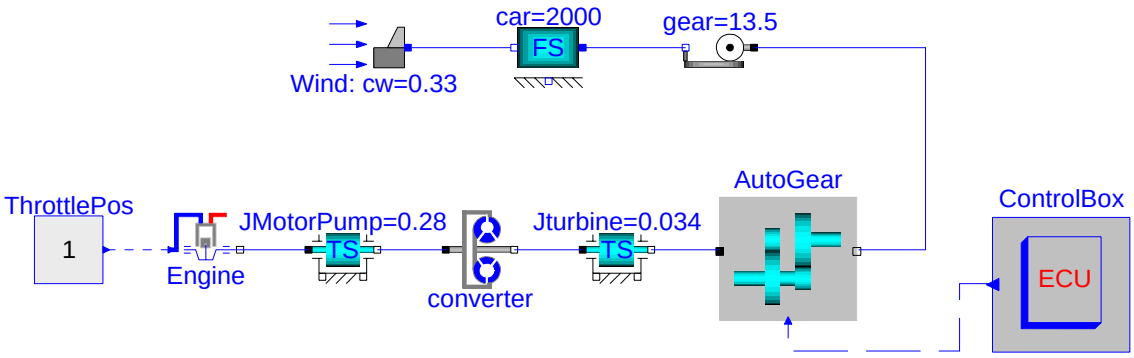


Figure 2: Composition diagram of vehicle drive train.

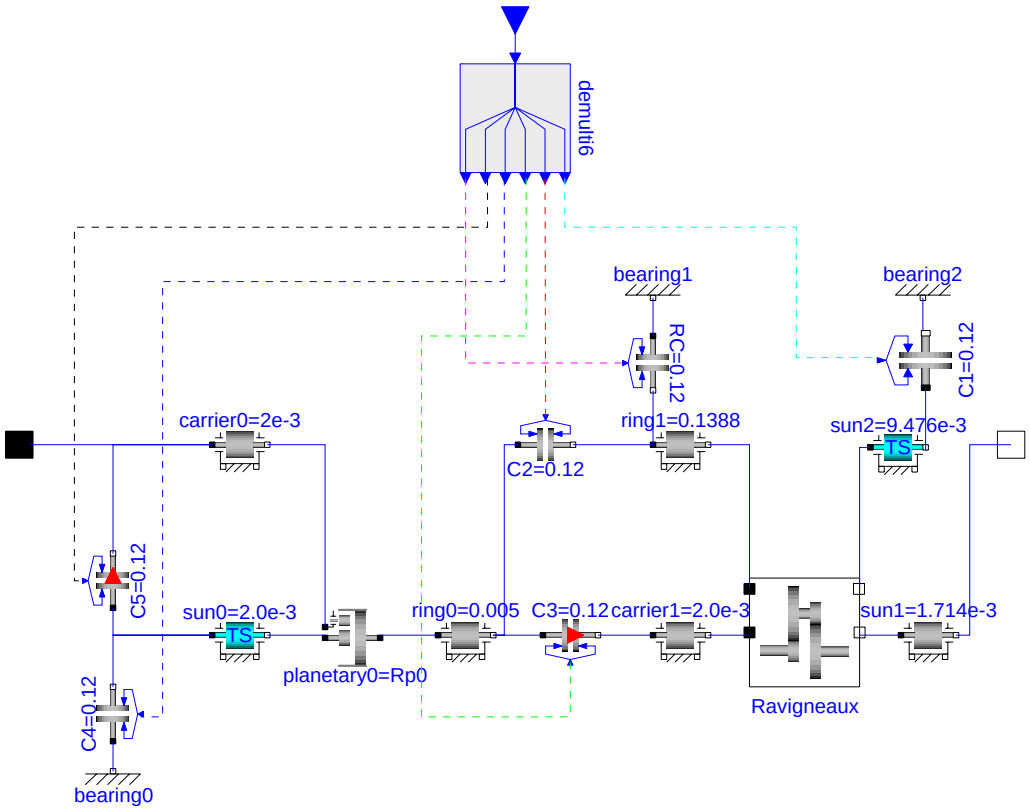


Figure 3: Composition diagram of automatic gear box “AutoGear”.

boxes in the icons. A line between two connectors defines a rigid, mechanical coupling between the respective components. Dashed lines respresent directed signal flow connections.

The driving part of the system in figure 2 consists of the engine, the torque converter and appropriate inertias. Engine and converter are modeled by tables. The driven part incorporates the automatic gearbox, the mass of the car, and aerodynamic forces. Component "ControlBox" represents the ECU which generates the gearshift signals. For offline tests of the simulation model simple ramp functions are used. During HIL simulations these signals are generated by the (hardware) ECU and fed to the simulation processor via appropriate interfaces like digital IO and CAN bus.

In figure 3 the Modelica model of the automatic gearbox (component "AutoGear") is shown. It is set up by a standard planetary wheelset and a Ravigneaux set, by shafts to model wheel inertias, by clutches and by combined clutch/freewheel elements (= one-way clutches). The dashed lines represent the signals of the ECU which are proportional to the desired pressure forces of the clutches. Component "Ravigneaux" is an instance of the Modelica class shown in figure 4 which builds up the Ravigneaux set by basic kinematic wheel elements.

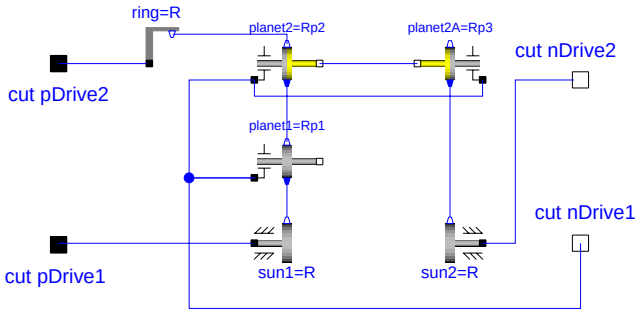


Figure 4: Composition diagram of Ravigneaux set.

MODELICA MODELS OF BASIC CONTINUOUS COMPONENTS

In the previous section the vehicle drive train model was built up by the connection of basic components, such as shafts and clutches, using composition diagrams. The most important continuous base components are now discussed in more detail, based on ideas of (Otter 1994): All drive train components have mechanical flanges which are used to connect components rigidly together and which are defined by the following Modelica connector class (a connector class is a class usable in connections):

```
connector DriveCut
    AngularVelocity     w  "velocity of cut";
    AngularAcceleration a  "acceleration of cut";
    flow Torque         t  "cut-torque";
end DriveCut;
```

Connector class DriveCut defines the three variables "w, a, t" using basic type classes defined in the Modelica standard library (type classes are classes which do not have equations):

```
type AngularVelocity     = Real(Unit ="1/s");
type AngularAcceleration = Real(Unit ="1/s.s");
type Torque              = Real(Unit ="N.m");
```

That is, all the three variables are real variables with a defined unit. The *Torque* type has the prefix **flow** in the connector definition, meaning that the sum of all torques at a connection point is zero. In other words, *Torque*s are through variables. Some typical components using this connector class are shown in table 1.

Model class *ShaftS* describes a shaft with inertia and two mechanical flanges p and n. The first two equations contain the relationship between the kinematic variables of the two flanges. The third equation defines that the angular acceleration is the derivative of the angular velocity and the last equation finally states Newtons law. Model class *Gear* is an ideal gearbox without inertia and finally model class *DriveSpringS* is a rotational spring where the spring torque is used as state variable.

MODELICA MODEL OF CLUTCHES

Clutches and freewheels lead to variable structure systems because two shafts can stick or slip relative to each other. The number of states is changing during a transition from stick to slip and vice versa. This complicates the modeling because a clutch would need the possibility to remove states from the shaft, which is a component connected to the clutch, i.e., a clutch can no longer be described by a local model. It is common practice to use the additional constraint equation that the relative angular acceleration vanishes while the clutch is stuck. Since the relative angular velocity is zero when reaching the stuck state, it remains zero due to this constraint equation. The graphical representation of a clutch within Dymola, is shown in figure 5.

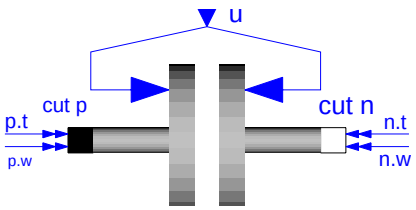
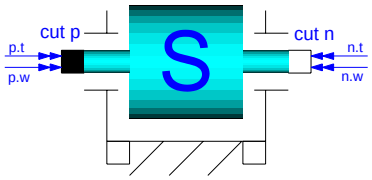


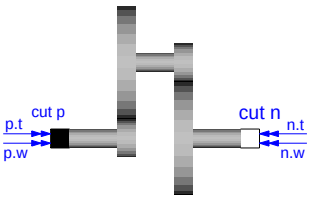
Figure 5: Composition diagram of clutch.

The Modelica model of this clutch is defined as:

```
model Clutch
    DriveCut p, n;
    Real     u  "control input u = 0..1";
protected
    AngularVelocity wrel "relative speed";
    Torque    tc     "constraint torque, if stuck";
    Boolean   locked "true if stuck";
```

Graphical Representation	Modelica description

	**model** *ShaftS*
	  parameter *Inertia J* "shaft inertia";
	  DriveCut p, n;
	equation
	  $p.w + n.w = 0$;
	  $p.a + n.a = 0$;
	  der$(p.w) = p.a$;
	  $J * p.a = p.t - n.t$;
	end *ShaftS*;

	**model** *Gear*
	  parameter *Real ratio* "gear ratio";
	  DriveCut p, n;
	equation
	  $p.w + ratio * n.w = 0$;
	  $p.a + ratio * n.a = 0$;
	  $ratio * p.t = n.t$;
	end *Gear*;

	**model** *DriveSpringS*
	  parameter *Real c* "spring constant";
	  DriveCut p, n;
	equation
	  der$(p.t) = c * (p.w + n.w)$;
	  $p.t = n.t$;
	end *DriveSpringS*;

Table 1: Components of Dymola's drive train library.

equation
  $wrel = p.w + n.w$;
  $0 \quad = \textbf{if } locked \textbf{ then } p.a + n.a \textbf{ else } tc$;
  $-p.t = \textbf{if } locked \textbf{ then } tc \textbf{ else } ta(wrel, u)$;
  $p.t \quad = n.t$;
  ...
end *Clutch*;

The clutch is controlled by the input signal u. For $u = 0$ the clutch is open, for $u = 1$ the maximum pressure force is applied. The first equation determines the relative angular velocity between the elements connected by the clutch component. The second equation is essential: it states that in the stuck mode ($locked = true$) the relative acceleration and in the sliding mode the constraint torque ($= tc$) vanishes, respectively. This is an equation with variable causality and it is not possible to solve for one of the two if-branches. It turns out that such equations are always part of an algebraic loop, i.e., the unknown variables in this equation are determined by the solution of a system of a algebraic equations. The third equation is used to compute the cut-torque in the mechanical flange p: In stuck mode, the cut-torque is equal to the contrainst torque tc. In sliding mode, the cut-torque is an applied torque and is calculated as function of the relative angular velocity and of the input signal. The model class for a freewheel is nearly identical to that of a clutch. The only difference is that for a freewheel there is no applied torque ta during the slip phase.

Having set up the equations for stick and slip it remains to be defined how to switch between these configurations, i.e., how the Boolean variable *locked* is set. This is done using a library for finite automata and petri nets (the details of this library are outside the scope of this paper). The finite automaton of a freewheel is given in figure 6.

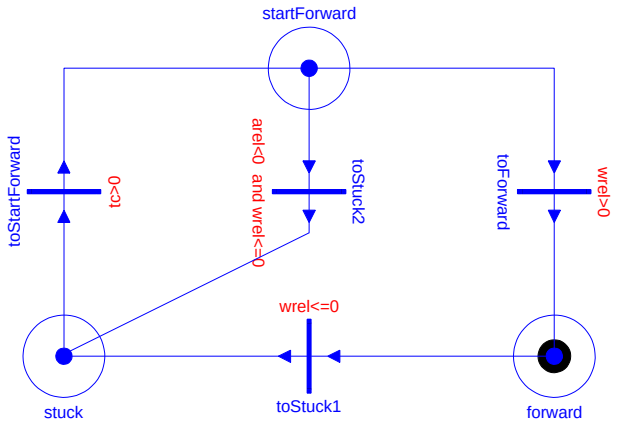


Figure 6: Switching structure of a freewheel.

It consists of the 3 discrete states *stuck, forward* and *startForward* representing stick, slip and start of slip. These states are instances of class *place* which has one connector with the Boolean variable *state*. Only *one* place object is active at a time, meaning that the corresponding Boolean

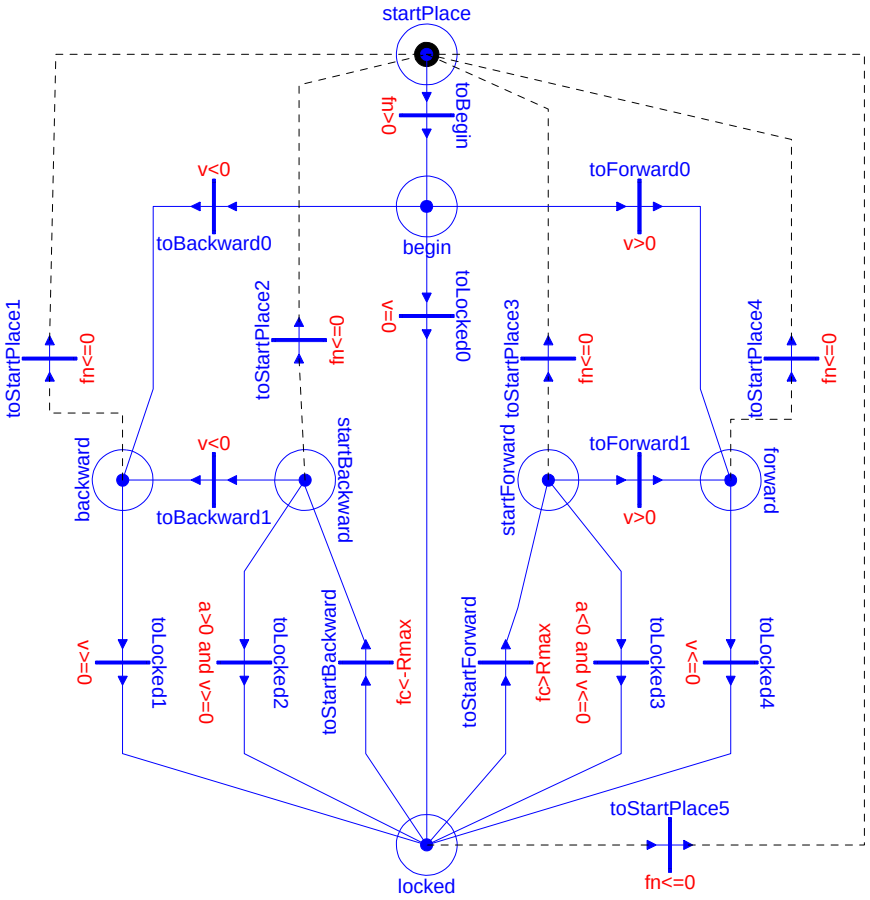


Figure 7: Switching structure of a clutch.

variable is *true*. The transitions between the states are described by objects of class transition (e.g. *toStuck1, toStartForward*). The conditions which fire a transition are defined at the transition objects (e.g. $wrel \leq 0$). Firing a transition means that the active state variable is set to false, and the state of the object connected at the other side of the transition component is set to true.

During integration, the state of the finite automaton does not change. If a switching condition becomes true the exact time of that event is determined by an iterative procedure (rootfinding) and the integration is stopped. The state of the automaton is changed and the whole model is evaluated based on the value of the new active state. As a result, another transition condition may become true. Therefore, the model is evaluated (and the finite automaton switched) until no transition condition is true anymore. Finally the integration is restarted. In order to connect the finite automaton of figure 6 to the dynamic model of the freewheel, just an equation of the form "locked = stuck.state" has to be added in the class of the freewheel, i.e., whenever the stuck state becomes active, the Boolean variable *locked* is *true*. In all other cases it is *false*.

Mathematically, the whole procedure can be seen as a fix-point iteration scheme to find a consistent configuration for the restart after an event. Alternatively, a consistent configuration could be determined by transforming the equations into a *linear complementary problem* (LCP) and by solving the LCP-equations. This approach is described in detail in (Pfeiffer and Glocker 1996). The LCP-formulation has the disadvantage that it is less suited for realtime applications since it is not possible to have just one function evaluation at every sample interval. In the next section it is discussed how the fix-point iteration scheme can be modified to fulfill this need.

The finite automaton of the freewheel in figure 6 is simple: The automaton switches from *forward* to *stuck* when the relative speed *wrel* becomes zero or less than zero. When the constraint torque *tc* becomes greater than zero, sliding occurs again. It is not possible to switch at once from *stuck* to *forward* because the relative speed is zero at the switching time and the automaton would switch back to *stuck* at once since the transition condition of *toStuck1* is fulfilled. Therefore, it is only switched to the intermediate state *startForward*. The switching to *forward* takes place when the relative speed becomes greater than zero (or more precise: greater than an epsilon-bound defined in the simulator). When several freewheels or clutches switch at the same time, it is also possible that the relative acceleration becomes less than zero in the *startForward* state. In such a case the automaton switches back into the *stuck* state.

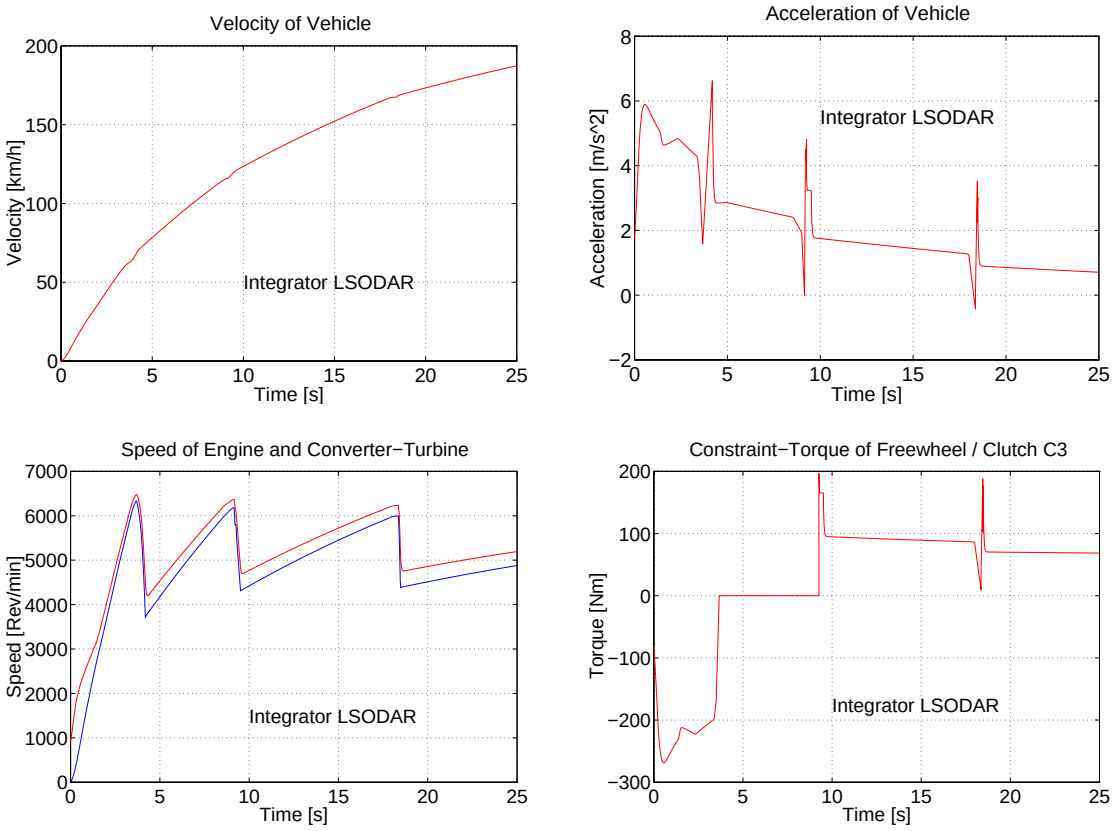


Figure 8: Simulation results of offline simulation.

The finite automaton for a clutch is more complicated because both forward and backward sliding can occur, see figure 7. Whenever, the normal force *fn* of the clutch vanishes, the clutch becomes in-active and the automaton switches in the *startPlace* state, independent of the previous state of the automaton. This starting state is only left, when the clutch becomes again active, i.e., when the normal force becomes positive.

REALTIME SIMULATION OF AUTOMATIC GEARBOX

The Modelica translator of the Dymola modelling and simulation environment takes the described Modelica model of the vehicle drive train and the automatic gearbox as an input, generates a differential-algebraic equation system (DAE) and transforms the DAE to state space form by symbolic manipulation and graph theoretical algorithms. Algorithmic details are described in (Elmqvist 1978, Duff et al. 1986). The original sorted equations contain a linear system of 66 equations, due to the shafts connected via rigid gearboxes and due to the clutch equations. Via tearing (Elmqvist and Otter 1994, Otter 1994) this system of equations is reduced to 10 linear equations which are solved by standard numerical procedures whenever the model is evaluated, i.e., the generated code is mixed symobolic/numeric. The final equations are stored as C-Code which is compiled and linked to the Dymola simulator for offline simulation. For realtime simulation Matlab CMEX code can be generated, used in Mathworks Realtime Workshop and downloaded to dSPACE hardware using Realtime Interface.

In figure 8 some simulation results for a drive shift through all 4 gears at full engine load are shown, using LSODAR, a variable step size, variable order integrator with rootfinder (Hindmarsh 1983). There are strong peaks in the acceleration because the engine speed is not reduced during gearshift and the simple ramp functions used are not well suited for clutch control. In HIL simulations a well tuned ECU provides better signals resulting in better gearshift comfort and less stress of the drive train components. The constraint torque of the combined clutch/freewheel element C3 shows that the freewheel is locked in the first gear (negative constraint torque) and that it is free in the second gear. The clutch is engaged in the third and forth gear.

During offline simulations iterations occur in two situations: First, for event detection (e.g. zero crossing of the relative velocity in a clutch) the event instant is determined by an iterative root finding algorithm. Second, for the event restart a new consistent configuration is found by the discussed fix-point iteration scheme. In realtime simulations fixed step integrators have to be used to enable IO synchronization. Iterations are carried out at fixed time instants and every iteration requires one model evaluation. This is unsuited for realtime simulation since a maximum number n_{iter} of iterations has to be estimated (say 5-10 iterations) and the sample time has to be big

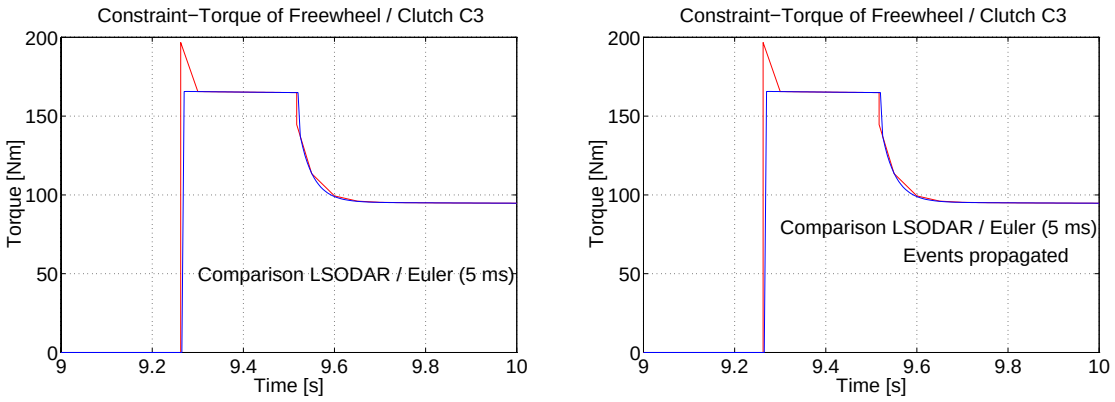


Figure 9: Comparision LSODAR/Euler with standard/time-propagated event iteration.

enough for n_{iter} model evaluations.

Therefore, a new simple technique was developed to overcome this difficulty: Conditions for state events are tested only at the sampling times of the realtime simulation. This is justified for realtime simulation, since the integration step size has to be small compared with model dynamics to avoid numerical oscillations. The essential new idea is to process only one state transition of a finite automaton at an event instant. If this is not yet a consistent model configuration, an event will occur after the next integrator step and one other state transition will take place. In other words, the iterations are propagated in time throughout the next steps. This kind of iteration is driven by representations of physical torques and forces and therefore only a few iterations are to be expected. For the automatic gearbox simulation it turns out that usually only 3-4 iterations are needed. This introduces only neglegable additional errors when these iterations are propagated in time as can be seen from figure 9 where a typical simulation result *around one event instant* is shown. In the left part of this figure an offline simulation with approximate root finding and usual event iteration is shown, whereas in the right part of the figure the event iteration is replaced by the discussed scheme.

For the evaluation of one Euler step we measured 4.7 ms on a Texas Instruments C40 DSP with 50 MHz and 0.65 ms on a DEC alpha processor with 300 MHz (both PC plugin cards by dSPACE). A typical gearbox ECU sampling time is 10 ms. This offers enough margin for model refinement and I/O using todays typical HIL simulation processors.

OUTLOOK

It is comparatively easy to find clutch control functions in terms of forces which fulfill the requirements regarding to gearshift comfort. Since the hydraulic clutch actuators used in typical automatic gearboxes introduce additional dynamics, it is more difficult to implement these clutch control functions. Therefore the hydraulic control of the clutches of the gearbox has also to be modeled to set up a complete HIL simulator which is useful for ECU tuning. That can be done using Modelicas hydraulic library which is under development. Finally, simulation results have to be compared with in-vehicle measurements to validate the models used.

REFERENCES

Duff, I.S.; A.M. Erismann; and J.K. Reid. 1986. *Direct Methods for Sparse Matrices.* Oxford Science Publications.

Elmqvist H.; D. Brück; and M. Otter. 1996. *Dymola – User's Manual*, Version 3.0, Dynasim AB, Lund, Sweden.

Elmqvist H. 1978. *A Structured Model Language for Large Continuous Systems*, Ph.D. Dissertation, Report CODEN: LUTFD2/(TFRT–1015), Dept. of Automatic Control, Lund Inst. of Technology, Lund, Sweden.

Elmqvist H. and M. Otter. 1994. "Methods for Tearing Systems of Equations in Object-Oriented Modeling". *Proc. ESM'94 European Simulation Multiconference*, Barcelona, Spain, June 1.-3., pp. 326-332.

Förster H.J. 1991. *Automatische Fahrzeuggetriebe.* Springer Verlag.

Hindmarsh A.C. 1983. "ODEPACK, a systematized collection of ODE solvers". *Scientific Computing*, edited by R.S. Stepleman et. al., North-Holland, Amsterdam.

Modelica. 1997. *A Unified Object-Oriented Language for Physical Systems Modeling.* Modelica homepage: http://www.Dynasim.se/Modelica/.

Otter M. 1994. "Object-Oriented Modeling of Drive Trains with Friction, Clutches and Brakes". *Proc. ESM'94 European Simulation Multiconference*, Barcelona, Spain, June 1.-3., pp. 335-340.

Otter M. and H. Elmqvist. 1997. "Energy Flow Modeling of Mechatronic Systems via Object Diagrams", *2nd International Symposium on Mathematical Modelling*, MATHMOD'97, Vienna, Feb. 5.-7.

Pfeiffer F. and C. Glocker. 1996. *Multibody Dynamics with Unilateral Contacts.* John Wiley Inc., New York.