AUTOMATIC CODE GENERATION FOR MULTIRATE SIMULINK MODELS WITH SUPPORT FOR THE OSEK REAL-TIME OPERATING SYSTEM

C. Homburg, U. Kiffmeier, L. Köster
dSPACE GmbH, Paderborn, Germany

This paper presents a block-diagram based approach for automatic code generation for embedded applications. The basis is a multirate Simulink/Stateflow model that is extended with implementation-specific information provided by special blocks. The OSEK standard is chosen as the target real-time operating system. The individual tasks and the inter-task communication have to be implemented efficiently by the code generator, the main mechanisms are discussed. The presented approach will be implemented in release 2.0 of the production code generator TargetLink.

Introduction

The software development for embedded systems is increasingly being done with the help of simulation tools and block diagram specifications. MATLAB, Simulink and Stateflow are well-accepted products in this area. Production code generators like TargetLink (dSPACE, 2000) are applied to transfer such graphical specifications of real-time control algorithms into a highly efficient, readable and reliable C code that fits into an Electronic Control Unit (ECU).

When this technique was first introduced, users started with pilot projects to gain experience with automatic code generation. The Simulink model covered only relatively small parts (features) of the whole control system, for example, the idle speed control of an engine. Typically, there was only a single sample rate, making life easier for the code generator. Code was generated for each RTOS task separately. Many projects have been carried out successfully with this pragmatic approach, but the problem remained, that users had to integrate the auto-generated code manually into the existing real-time kernel on their ECU.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35409-5_23

B. Kleinjohann (ed.), *Architecture and Design of Distributed Embedded Systems*

With the increasing confidence in automatic code generation, there is a trend to develop larger models of complete ECUs. Such models contain many features and sample rates, which have to be executed as different tasks of the ECU RTOS.

This contribution presents the TargetLink approach to automatic code generation from multirate Simulink models: The function developer creates a Simulink model containing all the features that have to be implemented. This model concentrates on the control algorithms, implementation aspects are not taken into account. In the next step the control algorithms have to be implemented. The functional model is automatically converted into a TargetLink implementation model. The conversion process is bidirectional. Implementation-specific model properties are preserved and stored when the implementation model is converted back to a pure Simulink model. TargetLink provides a special block set for the specification of implementation aspects. Within these blocks the software engineer can make all the settings for the implementation. Finally production-ready code including the RTOS configuration for the OSEK real-time operating system family is automatically generated out of the TargetLink implementation model.

The approach presented here is currently under development and will be part of the next major release of TargetLink.

OSEK Real-Time Operating System

The presented approach is based on the OSEK real-time operating system family, which was established as a standard for automotive ECUs. Many European automotive companies have committed themselves to this standard for new development projects, because OSEK's high scalability and flexibility perfectly supports portability and reusability of the embedded application software.

The OSEK specification describes a static real-time operating system (OSEK, 2000). All operating system objects such as tasks, events, messages and resources are created at compile time. Their attributes are described offline with the help of the OSEK Implementation Language (OIL). OSEK defines several standard attributes – like the *priority* of the object "TASK" – which must be supported by all OSEK operating system implementations. Some attributes are defined to ensure scalability of the operating system. However, the OSEK operating system developer is allowed to define additional implementation-specific attributes for OSEK objects. Based on the OIL description a highly efficient real-time kernel is generated that fits the user's needs while eliminating all overhead for features not required by the current application.

An OSEK operating system uses priority-based scheduling. In addition, the priority ceiling protocol is included for resource handling. The use of Resources is similar to the use of Semaphores in other operating systems, except that the priority ceiling protocol avoids deadlocks during resource occupation. Resources can be used to manage concurrent accesses of tasks and interrupt service routines with different priorities to shared resources like memory or hardware units.

As an extension of the OSEK standard, the *Communication Services* (OSEK/COM) and the *Network Management* (OSEK/NM) are provided to support distributed embedded systems.

Modeling of Multirate Systems

A typical controller model contains dozens of features each implementing a specific control function. Each feature may have parts which have to be executed at different sample rates, for example, in a 10 ms and a 100 ms time frame. Such parts are modeled as separate Simulink subsystems. Each of the Simulink subsystems contains only blocks with the same sample rate. The sample rate is specified at the Inport blocks or at discrete-time blocks within the subsystem. Many such subsystems may be wired to a feature and to the whole control system.

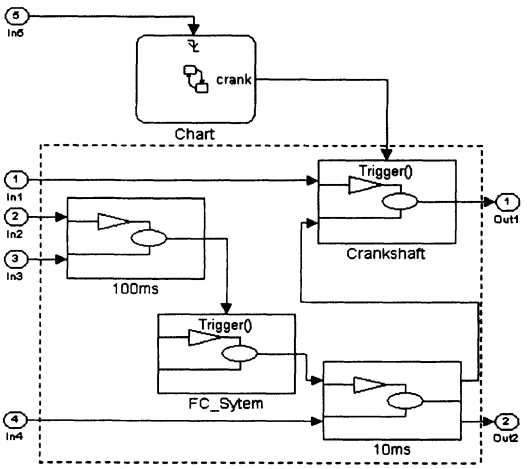

Figure 1. A typical Simulink/Stateflow diagram.

Figure 1 shows a typical Simulink/Stateflow diagram that contains cyclic and event-driven parts. In an engine control application, for example, the electric throttle control algorithm executes periodically and the ignition/injection part is driven by crankshaft events. In Simulink, asynchronous functions are modeled as triggered subsystems.

TargetLink provides a TASK block to declare a Simulink subsystem as a RTOS task and to specify its properties, for example,

- if it is preemptive or not.
- the events, which are owned by the task,
- the maximum number of activations,
- the task priority, etc.

If a Simulink subsystem is expected to become part of a specific task, a TASK block has to be placed into the subsystem (Figure 2). If the user makes no specific input, TargetLink uses a default partitioning. By default, all the model's periodic subsystems that have the same sample rate are grouped together in a common task.

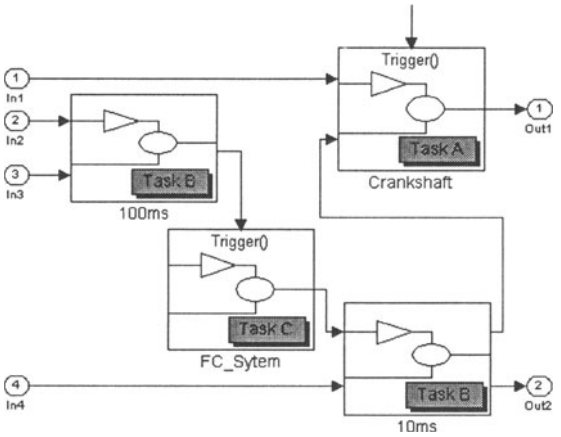

Figure 2. Specification of tasks with the Task block.

Within this task, the code for all subsystems with the same sample rate is executed. Triggered subsystems with common trigger sources are combined and their code is assigned either to the trigger source's task where this is also within the code generators scope (*FC_System* in Figure 1 is associated to task *100ms*) or to a separate task (*Crankshaft* in Figure 1).

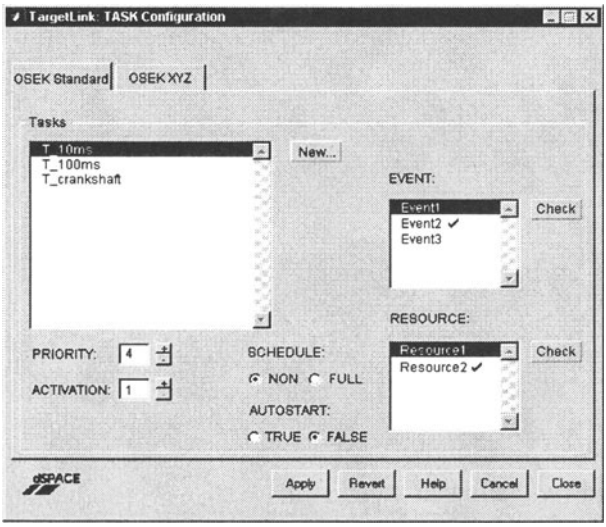

Figure 3. The Task block graphical user interface.

Figure 3 shows a screenshot of the TASK block dialog. In this case the Simulink subsystem is declared as a part of the *T_10ms* task, which is activated by *Event2* with

priority 4. Only one instance of the task can be activated at a time, and it is not activated automatically after startup.

If there are several subsystems in a model with the same sample time, they can be combined to a single RTOS task. This leads to a more efficient real-time implementation by reducing the scheduler overhead for managing many tasks separately. Alternatively, there is an option to define the execution sequence of tasks explicitly using the OSEK *ChainTask()* command. Additionally, even multirate subsystems can be assigned to a single RTOS task. The only restriction is that all their sample times must be multiples of a common divisor. In this case a counter variable is used to control the execution of slower sample rates.

As an alternative to the TASK block, the ISR block can be used to implement a subsystem as an interrupt service routine. This is typically used for short time-critical actions only and avoids scheduler overhead.

Inter-Task Communication

In Simulink, data exchange is modeled by signal lines. In a task-driven preemptive environment, data exchange becomes a more complex issue. In a RTOS environment, data exchange can be implemented by communication via operating system services, e.g. OSEK Messages. Alternatively, data exchange can be implemented via variables, with and without RTOS support for resource handling.

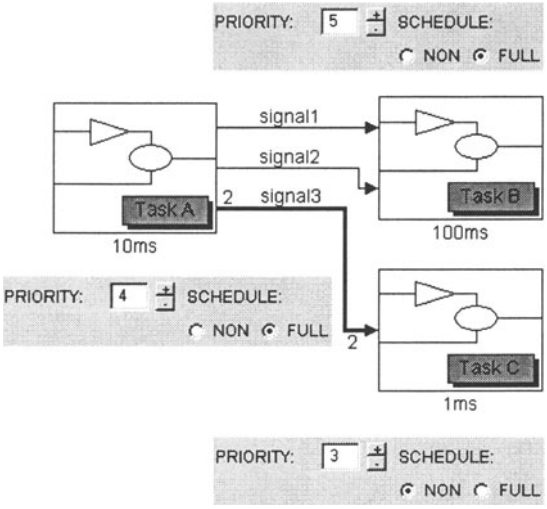

Figure 4. Communication between tasks.

The signal lines between the tasks establish a data flow. By default, TargetLink analyzes the data flow to determine the optimal implementation. For example, the sample rates and execution order are evaluated to determine if double-buffering is

necessary on the sender or receiver side of a message. Thus RAM space can be saved.

Figure 4 shows a scenario where tasks with different sample rates, priorities and preemptibilities exchange data. *Task A* sends *signal1* and *signal2* to *Task B*. Global variables that can be accessed by both tasks are usually used for this data exchange. Since *Task A* is preemptive (in OSEK terms SCHEDULE=FULL) and *Task B* has a higher priority, it may occur that *Task A* is interrupted after *signal1* has already been updated but while *signal2* still has its old value. Consequently, *Task B* would work with values of *signal1* and *signal2* from different time steps. This inconsistency can lead to unwanted behavior and normally has to be avoided. In this example, *Task A* is responsible for implementing a mechanism that is suited to protect against inconsistencies. One possible solution is to use local copies for *signal1* and *signal2*. At the end of *Task A* there is one compact block of code that copies this local representation to the corresponding global variables. It still has to be ensured that *Task A* is not interrupted during this copying procedure. This can be done by disabling interrupts or by using the OSEK service *resource*. The introduction of local working copies ensures that this critical section is as short as possible.

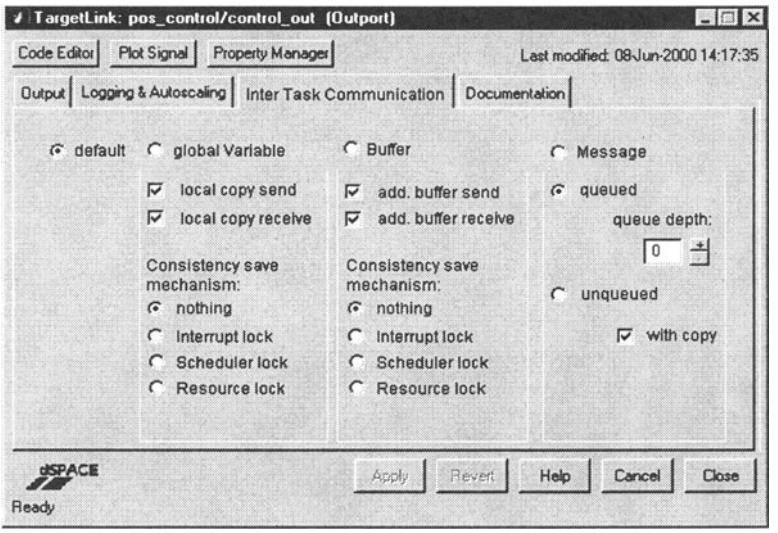

Figure 5. The specification of inter-task-communication.

It depends on the task settings and the possible interruption scenarios to determine if the sender or the receiver is responsible for the implementation of inconsistency protection. In Figure 4, *signal3* is also potentially exposed to inconsistency. Although it is a single data line it represents a vector and access to it could be interrupted. However, in the particular case of Figure 4, the sender cannot be interrupted by the receiver because *Task A* has a higher priority than *Task C*. Vice versa, *Task C* cannot be interrupted at all because it is not preemptive, i.e. SCHEDULE=NON. In this case, both tasks can use the global variable for *signal3* directly, which is of course the most efficicient way of data exchange.

Operating systems usually provide a means to exchange protected information between tasks. OSEK defines messages as a common interface for protected intra-ECU communication and for inter-ECU communication based on a common bus. For data exchange between tasks on the same ECU, these messages are not always the most efficient implementation since the automatically implemented inconsistency protection is sometimes not necessary, as shown in the last example. By default, TargetLink therefore automatically selects the most efficient implementation (global variables with or without copies, interrupt lock, resources etc.) depending on the task attributes priority and preemptibility.

The default behavior can be overruled by explicit user settings. For instance, this might become necessary if the user wants to ensure that these task settings will remain unchanged after code generation or if a signal goes outside the scope of the code generator. An example of the possible settings for data exchange is shown in Figure 5.

Special OSEK Blocks

Special blocks that are not included in the normal Simulink blockset provide a specification interface to specific OSEK features and reproduce their behavior during simulation. The Alarm/Counter block is given as an example.

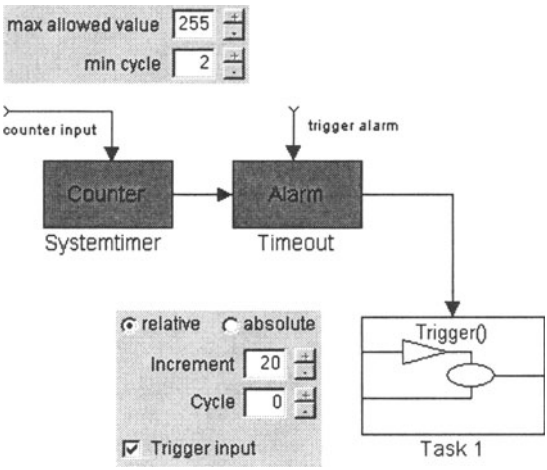

Figure 6. Alarm and Counter blocks.

In OSEK, alarms are automatically used to set up time-periodic tasks. In addition, TargetLink provides a special block that makes the general alarm functionality of OSEK available to the user. Since each OSEK alarm is assigned to a counter, the TargetLink Alarm block is always combined with a special Counter block that was designed especially for this purpose (Figure 6). Although the alarm and the counter always form a unit, they are divided into two blocks because one counter can have

several associated alarms. Code for the counter itself will usually not be generated because a hardware counter/timer or an operating system counter will be used. In this context the TargetLink counter will be used for offline simulation only.

Alarms are set up at system initialization time – e.g. for periodic tasks – or when a certain event occurs at runtime. The second case is modeled by a special input for a trigger that can be released by any source within the model. Absolute or relative alarms can be specified. The former expire at absolute counter values, whereas the latter expire at a certain number of (time-)ticks after the alarm was triggered.

OSEK Configuration and Tool Interaction

Normally, not all parts of an ECU application are auto-generated. Some tasks may be handcoded and have to be integrated with tasks generated by TargetLink. In this case there are two sources of OIL descriptions which have to be merged.

In order to ensure consistency, TargetLink maintains an OIL database that contains the complete RTOS configuration. An existing OIL description of legacy code can be imported into this database. When new code is generated, the RTOS configuration is written to an OIL file, which is loaded by the OSEK builder to generate the OSEK kernel for the ECU (see Figure 7). Simulink components can be assigned to OSEK objects that are in the OIL database already, or new entries can be created.

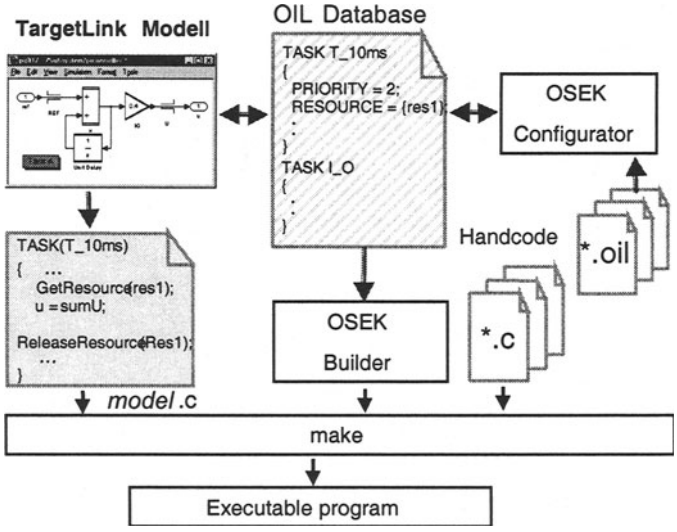

Figure 7. Tool interaction.

Target Specific Code Generation

As previously mentioned, the OSEK specification leaves a certain amount of flexibility to ensure optimal adaptation to different microcontrollers. As a consequence, different OSEK vendors implement some functionality in different ways, even if the implementation is for the same microcontroller. An example is the API function for counter access. There is no standard API function to read the system time. Therefore, the code for the same application appears slightly different for different OSEK implementations. Wherever possible, TargetLink uses standard OSEK API functions in the generated code. But when necessary, implementation-specific API functions are used to ensure efficient application code

Another area of incompatibility is the possible OIL attributes for OSEK objects. Here the OS vendors usually provide a lot more options than described in the standard. For example, tasks have several additional attributes, and the user might want to specify these attributes at the same place where the standard attributes are specified, i.e. in the TargetLink Task block. TargetLink therefore provides a mechanism to identify these extensions to the standard and let the user enter the desired options. The key for this feature is the OIL file. OSEK dictates that each vendor describes even the non-standard attributes by name and possible values in OIL syntax. TargetLink analyzes this description and provides the necessary GUI where the user can, for example, specify the stack mechanism for a task.

When the user wants to port an application from one OSEK implementation to another, TargetLink automatically replaces the implementation-specific API calls and OIL attributes. This increases portability. Although OSEK was specified mainly to facilitate the reuse of software, these jobs would have to be done manually if no automatic code generation were available.

Simulation and Test of the Generated Code

One of the major strengths of a block-diagram based approach is that it provides an executable specification of a control system. The behavior of the controller to be designed can be simulated offline on the PC, often using plant models to close the loop. Besides this Floating-Point Simulation Mode of the standard Simulink environment, TargetLink further provides the Production Code Host Simulation Mode and the Production Code Target Simulation Mode.

In Production Code Host Simulation Mode , the TargetLink-generated (often fixed-point) production code replaces the original Simulink blocks during simulation (software-in-the-loop). Using this simulation mode, the generated code can be validated and fixed-point arithmetic effects can be tested.

In Production Code Target Simulation Mode, the TargetLink-generated control algorithm is computed on a microcontroller that is identical to the processor on the production ECU (processor-in-the-loop). Simulink still provides the input signals for the controller. The signals are sent to and received from the target microcontroller by the serial interface of the PC. Using this simulation mode, final validation of the generated production code can be made under realistic closed-loop operating

conditions. The output of the target compiler can be validated because it is part of the test loop.

These features must still be supported when code is generated for OSEK. The original production code, now including OSEK API calls shall be used for simulation. To achieve this, TargetLink provides mechanisms for simulation with and without an OSEK operating system.

If no OSEK operating system is available on the target, TargetLink creates additional code that implements –in a simple way – all OSEK macros and API functions used by the generated application code. A simple scheduler substitute calls the tasks by sending 'activate task' commands. While a task executes on the target, the host waits for a response and the simulation time is halted. A preemptive scheduler is not needed.

If an OSEK operating system is available on the target, the simulation works as follows: The plant model on the host determines the simulation time. For a simulation step on the microcontroller, the system timer is disconnected from its hardware timebase. When the simulation time in Simulink increases, the host forces the target to set the system timer to the corresponding value, and the original OSEK task activation mechanism starts. After the task has finished, the computation results are sent back to the host. The communication with the host is accomplished by a low-priority idle task.

Conclusion

This paper has presented an automatic production-ready code generation approach for multirate block diagrams with real-time operating system support. The approach is based on quasi-standards in the automotive industry (i.e. Simulink, OSEK, C). The Simulink block diagram as the modeling basis holds all necessary information from the top level function development down to the ECU implementation, providing a system specification that is executable in all development phases. The presented approach fully integrates the OSEK configuration tools and the Simulink/TargetLink model. It will be available as a commercial product in TargetLink release 2.0.

Still open issues are the full support of ECU networks and WCET (worst case execution time) calculations in order to satisfy timing constraints.

References

dSPACE (2000). *Production Code Generation Guide; TargetLink 1.1.* dSPACE GmbH, Paderborn, Germany.

OSEK (2000). *OSEK Specification Vs. 2.1.* http://www-iiit.etec.uni-karlsruhe.de/~osek/.