

Experience with an Advanced Design Flow with OSEK Compliant Code Generation for Automotive ECU's

In this paper we describe our experience with an advanced development process for Automotive ECU's running OSEK. We have used this development process to develop a demo ECU for a car tail light. Running Motorola OSEK-OS/08 on a M68HC08 target with generated application code and an early HW/SW co-verification. In this advanced development process we identified five main stages. Some stages are performed in parallel to shorten significantly development time. One of the key advantages is the verification along the V-model, at each level of abstraction, while moving from the specification down to the final integration. In the traditional, flow the verification can only be performed on the right branch of the V-model of an ECU.

INTRODUCTION

ECU development is becoming more complex with the additional requirements and shorter time to market. One of the ways to meet these needs is to change the way ECU's are developed, through integrating and automating several disciplines, such as simulation, hardware design, software design and implementation.

The tail light module described here is a design flow example that demonstrates the path of a seamless integrated process, from high level functional specifications to a final software and hardware implementation. The ECU is based on a Motorola M68HC08 8-bit MCU.

This process focuses on two key areas. One is automatic code generation out of a software design model. The design model has been derived directly out of a formal functional requirements model. The other is the simulation of a virtual ECU that enables early integra-

tion, of the software and the simulated hardware, with zero waiting time for the real prototypes.

The developed tail light module demo is a fault tolerant controller. In case of failures in one of the lamps, another lamp takes over and compensates for the broken one and further PWM for dimming. (The module specification can be found in appendix A).

The functional requirements of the application are specified in StateMate MAGNUM[®] from I-Logix. The input to this phase was the textual requirements, where the output is a formal functional specification model that can be tested and verified through simulation, as well as rapid prototyping, and is directly used in the next development stages.

The next development stage is to import the functional model, which is design independent, into an I-Logix design tool, Rhapsody in MicroC. In this tool the software design and architecture are defined as well as specific HW details, like mapping signals to HW ports,

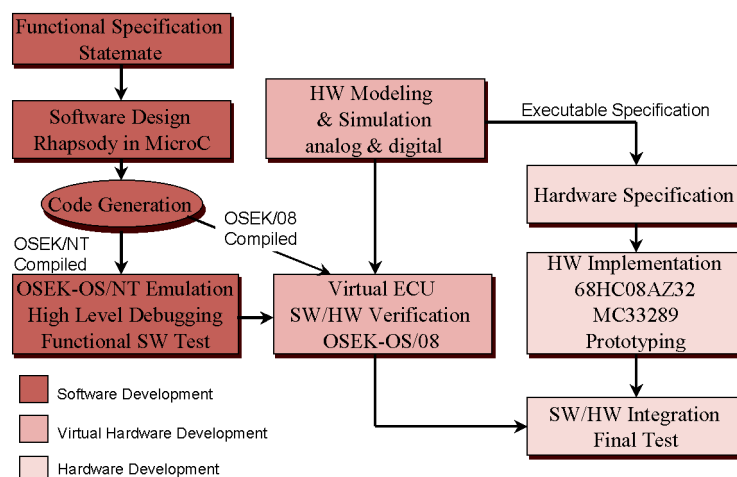


Figure 1. Tail Light Demo Development Flow.

messages and timing issues. The model is then synthesized automatically, using the Rhapsody in MicroC tool, into a production OSEK-OS compliant ANSI C code, that is optimized for the OSEK-OS realtime operating system. This generated code can then be tested either on the Motorola's OSEK-OS/NT, or directly on the target HW OSEK-OS machine, in this module the HC08, since the behavior on the OSEK-OS/NT is the same as on the target, OSEK-OS/08. The Rhapsody in MicroC Graphical Back Animation (GBA) is being used to verify and debug the functionality of the application model at the graphical level, while interacting with the OSEK environment, either the OSEK-OS /NT or the OSEK-OS/08.

The Virtual ECU is the simulation of the analog/digital hardware, software (generated application code and OSEK-OS/08) and the environment. The behavior model of the digital and the analog components provides the platform on which the application software is verified, via a debugger interface. The simulation environment consists of a VHDL behavioral model of the microprocessor, with a debugger and the co-simulation with the analog behavioral models in MASTTM. This is an early integration stage of HW and SW before a real prototype is made. Modification in the hardware architecture and implementation should be done at this level, as this is the first integration of the software with the virtual target hardware. At this early stage of the design process, the turn around time to solve system problems is significantly short. The modeling of the HW and the environment can be done in parallel to the software design. The HW models of the semi-conductors can then be used as living specifications for the chip designers.

The final integration of the generated code and the OSEK-OS on a hardware prototype consists of a M68HC08AZ32 EVB, a daughter board with the high side drivers MC33286, and a tail light. By using the HIWAVE debugger environment with a communication link to the GBA, it is possible to debug the designed software in the state and activity chart level where it was initially designed. This approach a vertical transparency of the software is achieved from the design level down to the running code on the target.

HIGH LEVEL SPECIFICATION

At this stage we define the functionality required from the application. This is done in Statemate MAGNUM.

The input to this phase was the textual requirements, as given in appendix A. While translating those requirements into a formal model, few unclear points were raised and resolved, in a relatively minor effort.

Throughout the process of completing and testing this model, we were able to "run the spec" using Statemate MAGNUM Simulator and Prototyper tools. At the end, we have compiled the model and ran a prototype of this specification model, the "Living Functional Specification". This functional specification model was later used to test and verify the application that we have made.

The top-level chart of the functional specification model is given in figure 2. Five control algorithms are

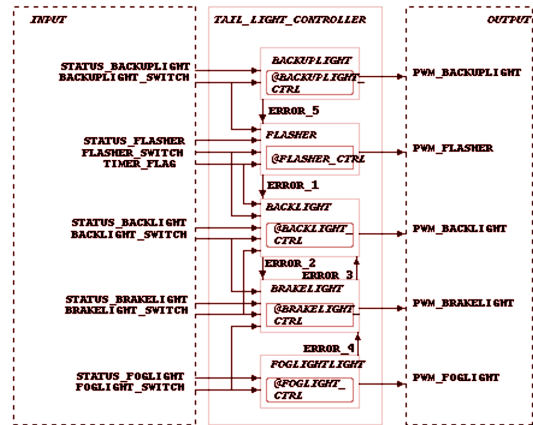


Figure 2. Top Level Activity Chart, in Statemate MAGNUM©. This picture shows the main 5 algorithms as well as the inputs and outputs of the controller.

defined here, according to the specification. Those control both normal operation mode and failure mode.

SOFTWARE DESIGN AND CODE GENERATION FOR OSEK2.0 (NT/08)

Here we have used an I-Logix design tool, Rhapsody in MicroC, to make the transition from the functional requirement model, that we have completed as described above, to the software design and implementation model. First we imported the functional model, which is HW-design independent, into this design tool. Then we went through a design stage, determining the execution frame (tasks) for each function, i.e., control algorithms, as well as handling the different PWM modes (Slow and Fast). Here we have defined 2 EXTENDED tasks, one EXTENDED task, named "TAIL_LIGHT_CONTROLLER", which handles both the inputs from the switches and error signals, as well as the slow PWM mode (1.5 Hz), and another EXTENDED task, named "PWM_FILTER", that handles the fast PWM mode, the dimming mode. Both of those tasks are being activated by an OSEK alarm that is defined on the system timer. We have linked the 10 input signals, 5 commands and 5 error signals with inverses logic, to the memory-mapped input ports that have been defined for them. We have also linked the 5 output signals with the appropriate memory-mapped port pins that have been defined for them.

Now we have synthesized automatically the software design model into ANSI-C OSEK 2.0 compliant code. We went through the stage of modifying the model, and then synthesizing the design model to code several times while testing and debugging the controller. We have found the combination of the Motorola's OSEK/NT OS environment with the Rhapsody in MicroC Graphical Back Animation (GBA) most useful to detect algorithmic errors. Also, the combination of the Motorola's OSEK-OS/08 environment and the HI-WAVE / Rhapsody in MicroC IDE that supports the Rhapsody in MicroC GBA from the target, was found to be very efficient in debugging on target errors that could not be found in the NT environment.

Turning off a lamp power caused one such error,

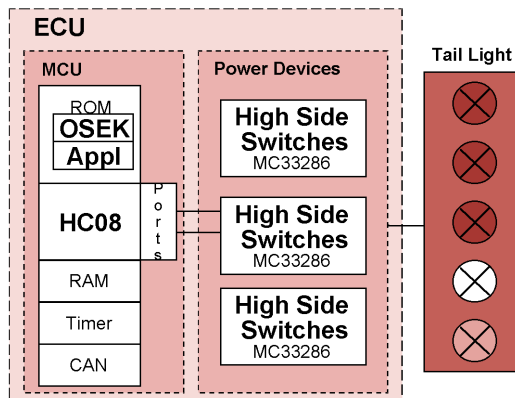


Figure 4. Tail Light Block Diagram.

described here, when detecting failure in this lamp. On the target a special silicon device, a high side driver (MC33286), is responsible to detect failures. Upon such a failure, the device set low an error bit. However, in the NT environment, this device was not used. Instead we have set "logical" errors. Initially, upon failure in one of the lines, the software has reset the line power. As soon as we got to the target environment, we have noticed that the controller is behaving strange. In spite of repeated trails, we could not reproduce the error in the NT environment. We have turned on the GBA instrumentation mode, and re-run a compilation cycle for the 68HC08. Now we have traced the code execution on the target and noticed the error. The problem was caused because we have turned off the lamp power. As soon as the power went off, the high side driver no longer identified an error, so it reset its error bit, thus causing the controller to back up from failure (compensation) mode to normal mode, and back again to failure as soon as the power was set again... Of course, when we keep the power open, the controller behaves as expected.

VIRTUAL ECU

Stage four is the simulation (hardware/software simulation) of the Virtual ECU which verifies the application code on the simulated target micro-controller and the analog environment (electrical, thermal, mechanical, etc.) on a behavior level. The behavior modeling of the digital and the analog components provides the platform on which the application software is tested.

Modification in the hardware architecture and implementation should be done at this level, as this is the first integration of the software with the target hardware.

This enables the user to have the complete analog/digital signal flow which is beneficial to the system designer, as he is able to look at the behavior of the system before the real hardware is designed. As this is done at an early stage of the design process, the turn around time to solve system problems is significantly reduced. In the traditional approach, this functional test is not executed until the real hardware is available.

The virtual ECU is based on a digital model of a cycle accurate MCU model, running on a logic simulator with the co-simulation to an analog simulator with the behavioral model of the power stages and the environment.

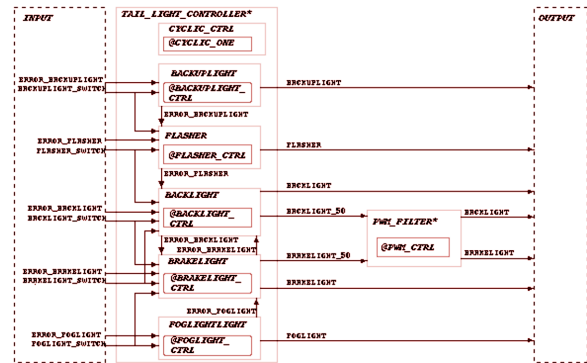


Figure 3. Top Level Activity Chart, in Rhapsody in MicroC™. This picture shows the 2 OSEK-TASKS, marked with "*", the main algorithms as well as the inputs and outputs of the controller.

The digital model is a behavior VHDL model of the 68HC08AS20. The model includes the core and the peripherals (timer, interrupt, RAM, ROM, SPI, SCI, etc). The simulation environment consist of a logic simulator (Leapfrog) with a Motorola internal tool extension (SiMez), which provides a debugger interface, and the capability to control the logic simulator via this interface. The debugger provides access to the model internal registers, and to the code running on the processor model.

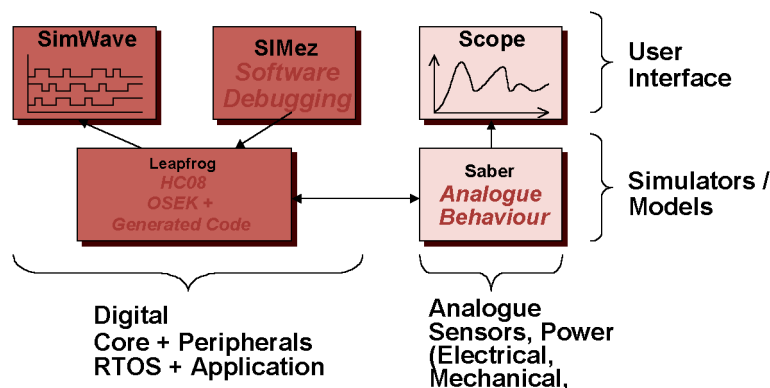


Figure 5. Virtual ECU Tools.

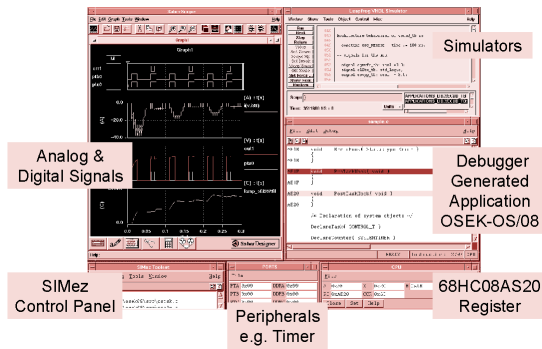


Figure 6. Virtual ECU Screenshot.

The analog part of the application (high side driver MC33286, heat sink, lamps, etc.) is modeled in MAST. These models demonstrate the multi domain simulation capability of analog behavior models.

The behavior model of the dual high side switch (MC33286) includes functions like the open load detection, over temperature shut down, current limitation, etc. These functions are modeled on the behavior level to gain simulation speed. Component features like the over temperature shutdown can be simulated due to the multi domain capability of SABER, as it allows to create an interaction between thermal and electrical circuits. The lamps are also multi domain models, in order to see electro-thermal effects like the inrush current which depends on the temperature of the filament. See figure 6.

Figure 5 shows a block diagram of the coupled simulators and the provided user interfaces. The signals and the debugger interface of the virtual ECU give access to the digital signals of the microprocessor and the analog ones of the power stages, heat sinks and lamps. The generated application software and the OSEK-OS/08 running on the microprocessor can be observed with the debugger interface. The 68HC08 core and the 68HC08AS20 peripherals with their registers can be accessed via the SIMeZ control panel when the simulation is stopped. The signal waves can be viewed by either the SABER Scope™ or the SimWave tool.

Through this simulation setup, it is possible to simulate the entire system behavior before even a prototype is built. The interactions between the application code and the operating system with the analog environment can be tested. For such a co-simulation 30k clock cycles/second has been achieved which is highly dependent on the interactions of the analog simulator with the digital simulator.

Figure 6 shows a screen shot of the simulation environment.

We see the need to simulate the entire system analog and digital especially for continuous systems, or where the feedback to the software is determined by time constants or events in the analog environment to verify the HW/SW design. The mixed signal simulation slows the simulation speed. However, for a virtual system design/check it is essential to have behavior models of all involved blocks. When the system was mod-

eled, the number of signals between the simulators had to be cut to the minimum, to avoid any simulation overhead. The simulation speed is very much influenced by events which are exchanged by the simulators. This means that the simulators have to synchronize when events are exchanged.

In a new design, the behavioral models, digital and analog, can then be passed on to the hardware designers as living specifications for the chip design. This ensures the compliance of the design to the agreed specification. This virtual prototype is used as a repository for the HW and SW development.

The software design and the code generation with emulation on OSEK-OS/NT can be done in parallel with the development of the hardware behavioral models, and the simulation on the target hardware. The development of the SW and HW models were done in parallel and the generated code was first tested on the model. The figure 7 shows the design flow.

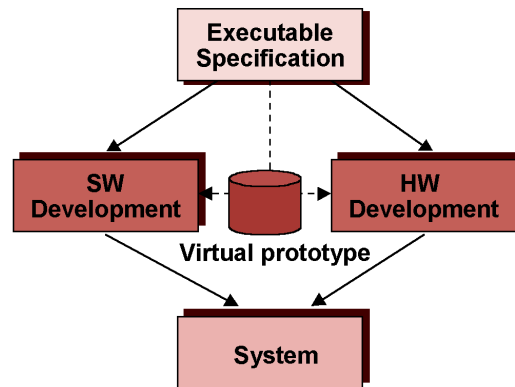


Figure 7. Parallel HW and SW development.

INTEGRATED DEVELOPMENT ENVIRONMENTS

We have used two main configurations regarding the application software. One is the Motorola's OSEK-OS/NT, the other is OSEK-OS/08.

The generated code was tested in both, while the Rhapsody in MicroC Graphical Back Animation (GBA) is being used to verify and debug the functionality of the application model at the graphical level, while interacting with the OSEK-OS environment.

OSEK/NT Environment

The Motorola OSEK-OS implementation provides an emulation on NT. Through that feature it is possible to test the behavior of the application code with the OSEK-OS on a hardware independent platform. With the graphical back animation a link is provided to the software design model. This creates a vertical transparency of the software design towards the target implementation. The internal state of the software is sent back to the design model, in Rhapsody in MicroC, via a TCP/IP link. Additionally the normal C-level debugging (e.g. MS Developer Studio) was used for testing.

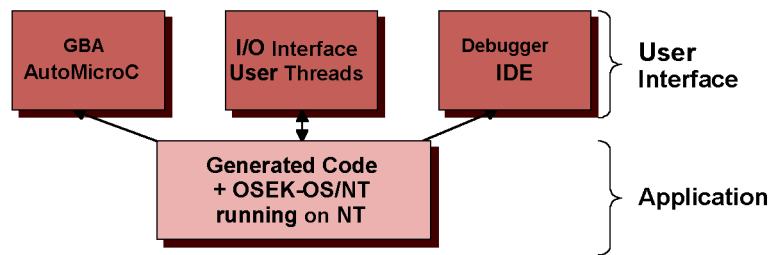


Figure 8. Generated code + OSEK environment.

The three main inputs are:

1. **Basic OS Configuration:** Rhapsody in MicroC uses this OS Basic Configuration as input for the CPU, system timer, and the initialization task settings of the OSEK. It then adds the object declarations (e.g. tasks, alarms, events, etc.) to generate the complete application configuration file (OIL-file) for the System Generator.
2. **Generated Application Code:** This part is generated by Rhapsody in MicroC and contains the tasks and the functionality of the application. This code is directly used and compiled for the target system.
3. **OSEK-OS Source Code:** The OSEK-OS source is selected depending on the target implementation (NT, HC08, HC12, M Core,...). Compiler switches in the system definition files, generated by the System Generator, are used to compile just the necessary source code and therefore optimize the code size of the OSEK-OS.

Figure 9 shows the tool and data flow for building an OSEK-OS application with code generation from Rhapsody in MicroC.

For porting the application to the 68HC08AZ32 target hardware, the Basic OS Configuration for the OSEK-OS/08 configuration is selected as input to Rhapsody in MicroC. The application is then rebuilt by starting the System Generator and re-compiled with the OSEK-OS/08 source code and linked. No manual changes have to be made to the generated application code.

Only the OS related blocks have to be modified (see figure below).

OSEK/OS Environment

In this environment, the previously shown, figure 9, application building environment is very much the same. However, here the HIWAVE environment forms the link between the application software and the Graphical Back Animation (GBA) instead of the direct TCP/IP on the NT.

CONCLUSIONS

We are certain that such methods, as described above, will eventually replace the traditional flow of independent hardware/software development, hand coding, and the late verification of algorithms on prototype hardware. The need of shorter cycle times leads to the development of software and hardware in parallel. Virtual ECU's provide this potential for earlier time to market. This approach also delivers a higher quality design, as the generated code has a higher maintainability due to its graphical representation, as well as the formal trace to the specification.

Using a living specification between car manufacturer and system supplier leads to using the same "language" and reduces the errors of transferring paper specifications into implementation. Since the code is generated automatically from the design model, the interaction between the car manufacturer and the sup-

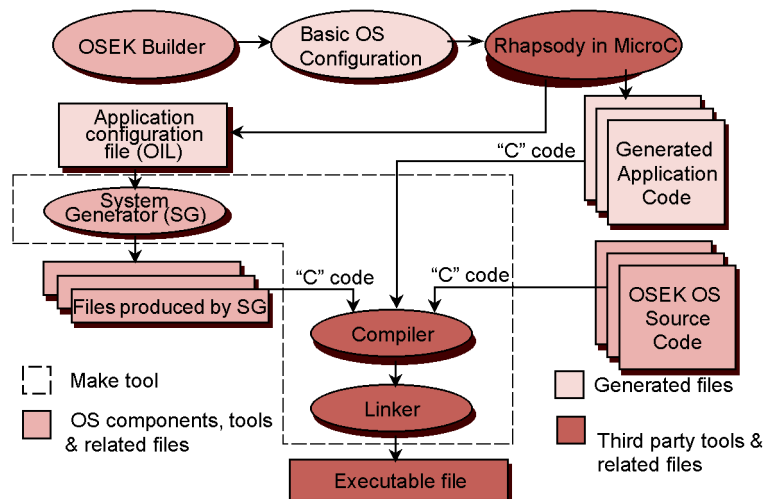


Figure 9. Building an OSEK application with generated code.

Light	Normal	Error 1	Error 2	Error 3	Error 4	Error 5
Flasher (orange)	1.5Hz ON/OFF	x				ON
Back light (red)	PWM 50%	1.5Hz ON/OFF	x	ON		
Break Light (red)	ON		PWM 50%	x	ON	
Fog Light (red)	ON				x	
Back Up Light (white)	ON					x

Table appendix.

plier is not limited to the first and second step but goes all the way to the virtual ECU and hardware implementation. This new process will significantly improve the quality of the end products, reduce cost and development time, and provide a strategic process advantage over those that continue to develop hardware and software sequentially, validate designs late in the development flow and use the traditional way of hand writing code ■

This article was composed by Raz Yerushalmi, M.Sc from I-Logix and by Manfred Thanner Dipl-Ing (FH), BEng(Hon) from Motorola GmbH.

REFERENCES

- 1 "Progress in Semiconductor Technology and its Impact on the Development Process", Dr.Ing. K.T. Neumann, Munich, A. Krueger, Genf, VDI Report Nr. 1415, 1998
- 2 "SW Synthesis for embedded Automotive Applications", Authors: S. Steinhauer, B. France, R. Yerushalmi, VDI Reports 1374, 1997
- 3 "OSEK/VDX V2.0 Specification"
- 4 "68HC08AZ32 Microcontroller" Motorola Technical Summary Rev 1.1
- 5 "68HC08AS20 Microcontroller" Motorola Technical Summary Rev 3.0, 1997
- 6 "MC33286 Automotive Dual High Side Driver", Motorola Technical Data, 1997
- 7 "OSEK-OS for NT Manual", Motorola, 1998
- 8 "OSEK-OS HC08 Manual", Motorola, 1998
- 9 "SIMEZ/HC08 User's Manual", Internal Motorola Document, 1997
- 10 "Analogy User's Manual", Analogy, 1997
- 11 "Leapfrog Manual", Cadence

APPENDIX A

Light Module Specification / Functional description

The aim is to demonstrate the entire design flow of an embedded automotive application. To demonstrate the design flow a tail light with relay replacement is chosen. Using smart power semiconductor devices for switching the load a fault handling can be achieved to ensure the functionality of a tail light in error cases.

For example, if the bulb of the backlight fails the functionality can be taken over by the dimmed brake light. The five bulbs have the following functions: Flasher, Back Light, Break Light, Fog, Back light, Backup indication. The input for the states are 5 signals from switches. The feedback about the status of the lights are accordingly 5 the inputs.

Error 1: Defect at Flasher

Error 2: Defect at Backlight

Error 3: Defect at Break Light

Error 4: Defect at Fog Light

Error 5: Defect at Back Up Light

The table shows which lamp takes over which functionality in which error. In case of the overlap of the backup functionality and the original functionality the original functionality has the higher priority. e.g. Error 1: Flasher lamp defect. The Back light takes over the flasher function. If now the back light is turned on the lamp will be dimmed at PWM 50%. The flashing will then be between 50% and 100% PWM.

Port Mapping

Output Port E[1:5] Lamp on / off

Input Port A[0:4] Driver diagnostics

Input Port B[0:4] Switches for input



We welcome news from your company through press-releases that can be submitted on our website at <http://www.dedicated-systems.com/VPR>

For more information contact Nico Van Wijmeersch.

Phone. 32-2-520.55.77 or Fax. 32-2-520.83.09 or Email. info@dedicated-systems.com