

Anwendung des OSEK-Betriebssystems am Beispiel der Betriebsstrategie eines Hybridfahrzeugs

Application of an OSEK Operating System for a Hybrid Vehicle Control Strategy

Dipl.-Ing. Christian **Amsel**, Dipl.-Ing. Christian **Renner**, Dipl.-Ing. Karsten **Wittek**
Institut für Kraftfahrwesen Aachen, Aachen

Dr.-Ing. Helmut **Brock**
Vector Informatik GmbH, Stuttgart

Zusammenfassung

Am Institut für Kraftfahrwesen Aachen (ika) werden seit vielen Jahren im Bereich der unkonventionellen Antriebe neuartige Antriebssysteme konzipiert, entwickelt und erprobt. Das von der Europäischen Union geförderte Projekt INMOVE hat das Ziel, ein wettbewerbsfähiges und optimiertes Antriebskonzept eines Parallelhybridfahrzeugs zu etablieren. Die hierzu erforderliche komplexe Steuer- und Regelstrategie wurde zunächst entwickelt und durch Simulation optimiert. Im Anschluß erfolgt die Implementation dieser Strategie auf den Fahrzeugsteuerrechner (Vehicle Management Unit - VMU).

In diesem Vortrag werden die Vorgehensweise und die Vorteile beim Einsatz eines OSEK Betriebssystems erläutert. Beispielhaft wird die Projekterstellung anhand des Parallelhybridfahrzeugs INMOVE beschrieben. Mittels einer Standardaufgabe werden darüberhinaus die strukturellen Verbesserungen gegenüber der herkömmlichen Programmierweise gemessen.

Summary

The Institut für Kraftfahrwesen Aachen (ika) is well known in conception, development and testing of unconventional drivetrain systems since many years. The project INMOVE, as an abbreviation of the working title 'Integrated Modular Electric Propulsion System for Parallel Hybrid Vehicles', is funded by the European Commission. It's aim is to establish an competitive and optimized approach of the drivetrains architecture. The complex control strategy has been developed with the help of simulation tools and has to be implemented in a Vehicle Management Unit (VMU) now.

The procedures and advantages of software design with an OSEK operating system are discussed on the example of the project INMOVE. In addition the improvements are

demonstrated on a standard application.

1 Einführung

Der Anteil elektronischer Bauelemente in modernen Kraftfahrzeugen hat in den vergangenen Jahren stetig zugenommen. Das führte zu einer erhöhten Zahl an Steuergeräten. Ziel sollte es hingegen sein, die Anzahl der Steuergeräte überschaubar zu halten und mehrere Funktionsgruppen in einem größeren Steuergerät zu vereinen, um deren Anzahl zu minimieren. Das bedeutet eine hohe Komplexität der Steuerungssoftware, die nur zu bewältigen ist durch eine strukturierte Programmierung in Modulen, so daß die Software leicht erweitert oder angepaßt werden kann.

Ebenso ist es besonders wichtig, den Quellcode so übersichtlich zu halten, daß in der nächsten Generation des Steuergerätes möglichst viele Funktionsgruppen beibehalten werden können und nur notwendige Ergänzungen und Abänderungen vollzogen werden müssen. Auch die Portabilität der Software zu anderen Baureihen des Fahrzeugs sollte gewährleistet sein. Diese verschiedenen Forderungen führen zu einem Zwang nach einer Standardisierung im Bereich der Softwareentwicklung für Steuergeräte.

Ein Projekt zur Standardisierung der Systemfunktionen Kommunikation, Netzmanagement und Echtzeitbetriebssystem in einem vernetzten System stellt OSEK (**O**ffene **S**ysteme und deren Schnittstellen für die **E**lektronik im **K**raftfahrzeug) dar, welches von einer großen Anzahl an Fahrzeugherstellern und Zulieferern vorangetrieben wird. Es ermöglicht die vereinfachte Kommunikation innerhalb und zwischen Steuergeräten unterschiedlicher Hersteller, führt zu weitreichenden Vereinfachungen für die Softwareimplementierung und geht einher mit hohen Kosteneinsparungen.

Am Beispiel des EU-Projektes INMOVE (**I**ntegrated **M**odular Electric Propulsion System for Parallel Hybrid **V**ehicles) wird in diesem Beitrag anhand der Vehicle Management Unit (VMU) des am Institut für Kraftfahrwesen Aachen (ika) aufgebauten parallelen Hybridfahrzeugs der Einsatz einer Echtzeitanwendung auf Basis des OSEK-Betriebssystems erläutert. Die VMU ist Bestandteil eines CAN-Busses und kommuniziert mit den Steuergeräten für das Batteriemangement (BMU), die elektronische Kupplungseinheit (ECS) und die Motorsteuerung des Elektromotors (MCU). Die VMU spielt dabei eine systemübergreifende Rolle.

2 Wahl eines Betriebssystems

Software-Implementationen lassen sich grundsätzlich auf zwei Arten realisieren. Der überwiegende Teil der Software-Implementation wurde bisher durch einfache Zeitscheiben realisiert. Durch einen streng zyklischen Aufruf der einzelnen Funktionen konnte nicht unterschieden werden, ob die jeweilige Funktion gerade aktiv oder inaktiv ist. Funktionale Anforderungen und der Zwang nach einer gleichmäßigen Verteilung der Laufzeiten auf die Zeitscheiben führte zu einer Aufteilung logisch zusammenhängender

Funktionalitäten in unterschiedliche Zeitscheiben.

Wenn neue Funktionen realisiert werden, muß zum Teil eine Veränderung des Zeitscheibentimings vorgenommen werden. Dies führt unter Umständen dazu, daß bestehende Funktionen in andere Zeitscheiben verlegt werden müssen. Die Unabhängigkeit der Softwaremodule geht auf diese Weise weiter verloren.

Daraus ergeben sich folgende Probleme herkömmlicher Softwareimplementierung :

- Die Softwarepflege wird durch komplexe, verknüpfte Strukturen erschwert.
- Die Erweiterung und Portierbarkeit ist wegen der schlechten Überschaubarkeit der Software und geringer Modularität stark eingeschränkt.

Der Einsatz eines Betriebssystems kann helfen, komplexe Steuerungs- und Regelungsaufgaben zu realisieren. Hierzu muß das Betriebssystem eine Struktur bieten, durch die dem Programmierer alle organisatorischen Tätigkeiten abgenommen werden. Unterschiedliche Aufgaben, z.B. das zyklische Versenden bestimmter Nachrichten aber auch das schnelle Reagieren auf Ereignisse, müssen unsichtbar für den Programmierer vom Betriebssystem durchgeführt werden.

Auf diesen Gründen ergeben sich klare Vorteile gegenüber den klassischen Zeitscheibenmodellen. Komplexere Aufgaben können in eine Vielzahl von einzelnen, voneinander unabhängigen Problemen aufgegliedert werden, die für sich betrachtet leicht zu programmieren sind. Eine Erweiterbarkeit einzelner Funktionen ist sehr gut möglich, da logische zusammenhängende Funktionen erhalten bleiben. Eine Wiederverwendung für weitere Anwendungen ist aufgrund der Modularität sehr gut möglich.

Am Institut für Kraftfahrwesen Aachen (ika) fiel im Jahre 1998 die Entscheidung, zukünftige komplexe Anwendungen auf Basis des OSEK Standards durchzuführen. Aus diesem Grund wurde am ika das Produkt osCAN167 der Fa. Vector Informatik angeschafft.

3 OSEK

Im Mai 1993 wurde OSEK in der Kraftfahrzeugindustrie gegründet mit dem Ziel, einen Industriestandard für offene Fahrzeugarchitekturen mit verteilten Steuergeräten zu schaffen. OSEK ist die Abkürzung für Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug. Gegründet wurde dieser Arbeitskreis von den „Initial Partners“ BMW, Bosch, Daimler-Benz, Opel, Siemens, VW und der Universität Karlsruhe. Die französischen Fahrzeughersteller PSA und Renault entwickelten zunächst einen eigenen aber vergleichbaren Ansatz VDX (Vehicle Distributed eXecutive) und schlossen sich OSEK 1994 an. Hieraus entwickelte sich der Name OSEK/VDX.

Beim ersten Workshop im Oktober 1995 präsentierte die OSEK/VDX Gruppe die Ergebnisse der Harmonisierung zwischen OSEK und VDX. Nach dem 2. Internationalen

Workshop im Oktober 1997 wurde die 2. Version der OSEK Spezifikation veröffentlicht.

In den letzten Jahren fand das Projekt zunehmendes Interesse innerhalb Europas. Mittlerweile sind ca. 40 europäische Firmen Mitglied im Technical Committee.

3.1 Motivation und Ziele

In der Softwareentwicklung von Steuergeräten gab es in der Vergangenheit hohe Kosten für Entwicklung und Variantenmanagement von immer wiederkehrenden nicht-anwendungsrelevanten Aspekten der Steuergerätesoftware. Zusätzlich verminderte eine Inkompatibilität der Steuergeräte verschiedener Hersteller bedingt durch abweichende Protokolle und Schnittstellen eine erneute Nutzung bereits entwickelter Software.

Mit OSEK wird die Erweiterbarkeit, die Portierbarkeit und die Wiederverwendbarkeit der Applikationssoftware unterstützt. Erweiterbarkeit bedeutet die Integration neuer Funktionen in eine vorhandene Hardwareplattform. Unter Portierbarkeit versteht man den Transfer von Applikationsfunktionen von einer Hardwareplattform zu einer anderen. Zu diesem Zweck wurden abstrakte und anwendungsunabhängige Schnittstellen definiert. Zusätzlich mußte eine Nutzerschnittstelle spezifiziert werden, die unabhängig von Hardware und Netzwerk ist. Auf diese Weise wird eine Wiederverwendbarkeit von Softwaremodulen optimal unterstützt.

Das System OSEK ist konfigurierbar und skalierbar, so daß eine optimale Anpassung an die jeweilige Anwendung vorgenommen werden kann.

3.2 OSEK Architektur

Innerhalb von OSEK wurden drei firmenübergreifende und nicht wettbewerbsrelevante Softwarefunktionen realisiert. Auf diese Funktionen kann die Anwendung über standardisierte Schnittstellen (APIs = **A**pplication **P**rogramming **I**nterface) zugreifen.

Für jede dieser Funktionen wurde innerhalb von Arbeitskreisen jeweils eine Spezifikation erarbeitet. Nachdem im Oktober 1995 die Spezifikation OSEK 1.0 veröffentlicht wurde, konnte zwei Jahre später eine grundsätzlich überarbeitete Version im Oktober 1997 vorgestellt werden.

Der aktuelle Stand der Normung von OSEK 2.0 läßt sich wie folgt angeben:

- Betriebssystem (OSEK Operating System): Ver. 2.0 r1
- Kommunikation (OSEK Communication): Ver. 2.1
- Netzmanagement (OSEK Network Management): Ver. 2.5

Das OSEK Betriebssystem stellt eine Spezifikation einer betriebsmitteleffizienten und einheitlichen Laufzeitumgebung in Form eines Multitasking-Betriebssystems für

Software in Steuergeräten dar.

Die Funktion Kommunikation bietet Schnittstellen und Protokolle für den fahrzeug-internen Datenaustausch innerhalb (Intertaskkommunikation) und zwischen Steuergeräten (Transportprotokolle) an. Hierbei ist die Organisation des Datenbussystems nicht von Bedeutung.

Das Netzwerkmanagement stellt Dienste zur Überwachung der Stationen (z.B. Ausfall eines Teilnehmers) und zum geordneten Ein- und Ausschalten (z.B. Sleep-Modus) des Kommunikationsmediums zur Verfügung.

Detaillierter soll an dieser Stelle auf Funktionen des Betriebssystems eingegangen werden.

Das OSEK Betriebssystem besteht aus folgenden Elementen:

- Task
Eine Task ist aus programmierertechnischer Sicht ein Unterprogramm. Der Aufruf einer Task wird vom Betriebssystem verwaltet. Zur Compilerzeit wird einer Task eine Priorität zugewiesen, die sich zur Laufzeit nicht ändert. Dies ist der Grund, weshalb das OSEK-Betriebssystem als statisches Betriebssystem bezeichnet wird. In OSEK gibt es Basic- und Extended-Tasks, die sich nur darin unterscheiden, daß eine Extended-Task während des Durchlaufs in den Wartezustand versetzt werden kann, um damit anderen Tasks Rechenzeit einräumen zu können. Eine wartende Task wartet immer auf ein bestimmtes Ereignis. Tritt dieses ein, so wird die Task vom Betriebssystem automatisch wieder in Betrieb genommen, sofern nicht gerade eine Task mit höherer Priorität den Prozessor belegt. Weiterhin kann festgelegt werden, ob eine Task preemptiv oder nicht preemptiv ist, also während ihres Durchlaufs unterbrechbar ist oder nicht. Tasks sind nebenläufig, d.h. sie besitzen eine eigene Zeitachse.
- Event
Die Echtzeitanforderungen erfordern, daß die Bearbeitung von Aufgaben synchron zu Ereignissen gestartet und innerhalb einer definierten Frist beendet wird. Events sind die wichtigsten Synchronisationsmittel im OSEK-Betriebssystem. Eine wartende Task, somit steht dieses Mittel ausschließlich Extended Tasks zur Verfügung, kann ausschließlich auf ein bestimmtes Event warten. Wird dieses Event gesetzt (dies ist von Basic und Extended Tasks möglich), so wird die auf dieses Event wartende Task in Betrieb genommen. Sollte der Prozessor in diesem Moment von einer Task gleicher oder höherer Priorität belegt sein, so wird die wartende bei Freigabe des Prozessors durch die höherprioritäre Task aktiviert.
- Alarm und Counter
Für zeitabhängige Steuerungsaufgaben stellt das Betriebssystem Alarme zur Verfügung. Ein Alarm kann entweder zu einem bestimmten absoluten Zeitpunkt ein Event setzen oder eine Task aktivieren. Die Verwaltung der Alarme wird asynchron

zur Anwendung durch das Betriebssystem übernommen. Counter diesen als Basis für Alarme und können von der Anwendung nicht verändert werden.

- **Interruptverwaltung**
Das Reagieren auf spontane Ereignisse ist ein wichtiger Aspekt eines Echtzeitsystems. Jedem Interrupt wird eine benutzerspezifische Interrupt-Service-Routine (ISR) zugeordnet, die das laufende Programm unterbricht. Es sind drei ISR-Kategorien in OSEK erlaubt. ISR der Kategorie 1 beinhalten keinen Aufruf des Betriebssystems, während ISR der Kategorien 2 und 3 bestimmte Betriebssystemdienste nutzen.
- **Ressourcenverwaltung**
Durch die Ressourcenverwaltung wird der Zugriff mehrerer Tasks auf gemeinsam genutzte Ressourcen koordiniert. Zu diesem Zweck wird die Taskpriorität zur Laufzeit durch das Betriebssystem so verändert, daß konkurrierende Tasks nicht gleichzeitig auf eine Ressource zugreifen können.
- **Fehlerbehandlung**
Das OSEK Betriebssystem stellt Mechanismen sowohl für eine zentrale als auch dezentrale Fehlerbehandlung in Form sogenannter HOOK-Routinen zur Verfügung. Diese stellen ein Gerüst dar, das es dem Anwender ermöglicht, durch eine eigene Auswertung die Fehler zu verarbeiten.

3.3 Reduzierung der Softwarekomplexität

Ein wesentlicher Aspekt bei der technischen Realisierung ist es, die Software klar zu strukturieren. Je näher die Softwarestruktur dem technischen Modell kommt, desto verständlicher ist sie. Die Software läßt sich dann leichter implementieren und natürlich auch besser warten.

Die wichtigste Aufgabe, schon bei der technischen Modellierung, ist die Bildung von unabhängigen Funktionsgruppen (siehe obige Abbildung). Ein Hinweis auf eine solche Funktionsgruppe, die „Task“ genannt wird, sind individuelle zeitliche Anforderungen. So ist der Ablauf einer Task immer abhängig vom Eintritt eines bestimmten Ereignisses wie zum Beispiel eines zyklischen Zeitintervalls oder eines Hardwareinterrupts. Weitere Kriterien sind die Priorität und die Unterbrechbarkeit von Funktionsabläufen.

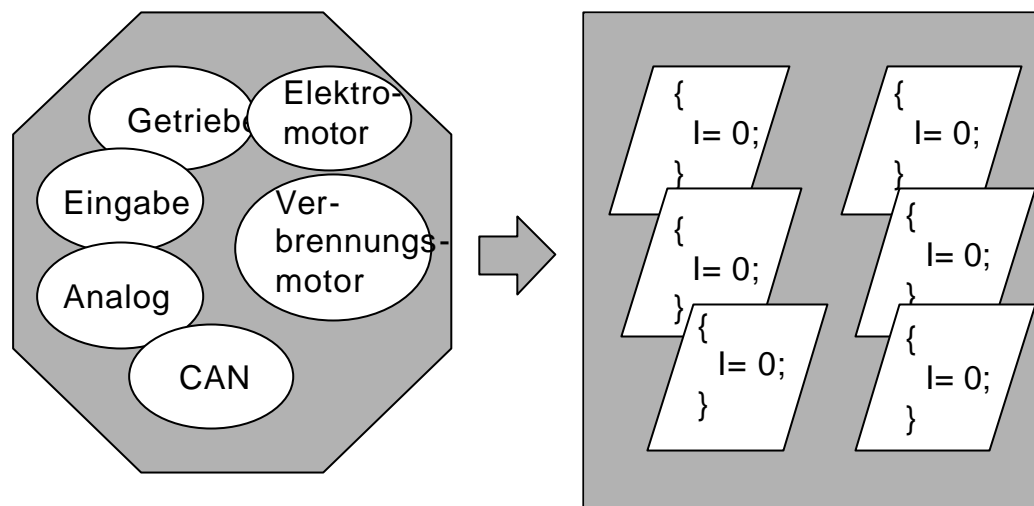


Abb. 1: Umsetzung des technischen Modells in eine Softwarestruktur

Das Betriebssystem übernimmt die Koordination dieser Tasks. Dazu gehört in erster Linie der Start des Ablaufs der Tasks, gesteuert nach deren Prioritäten. Der Softwareentwickler kann sich somit auf die inhaltlichen Aspekte der Funktionsabläufe konzentrieren. Das Betriebssystem OSEK ist optimiert für die Verwaltung von ereignisgesteuerten Tasks. Es unterscheidet sich hierin von Zeitscheibenmodellen, wie sie häufig in Multi-Usersystemen eingesetzt werden. Insbesondere zeitkritische Abläufe lassen sich auf diese Weise optimal steuern, da Rechenleistung ausschließlich im Bedarfsfall angefordert wird und kein zyklisches Durchlaufen nicht erforderlicher Programmteile stattfindet. Eine Reduzierung der Softwarekomplexität für den Anwender wird erreicht, da die Ablaufsteuerung, wie oben beschrieben, durch Routinen des Betriebssystems erledigt wird und unsichtbar für den Anwender bleibt.

Im folgenden soll das Vorgehen bei der Softwarestrukturierung unter Ausnutzung des Betriebssystems OSEK erläutert werden. Im Abschluß werden die erreichbaren Vorteile anhand einer typischen Anwendung gezeigt.

3.4 Strukturierungsmodelle

In vielen laufenden Anwendungen findet man ein Softwaremodell, in dem verschiedene Ablaufblöcke in einem festen Zeitraster aufgerufen werden (Abb. 2). Die Steuerung geschieht durch eine Zustandsmaschine von unterschiedlicher Komplexität. Für den Softwareingenieur ist dieses Modell sehr bequem, denn es legt die Bearbeitungsreihenfolge explizit fest und stellt damit die Datenkonsistenz sicher. Dies bedeutet jedoch nicht, daß es auch einfach zu warten ist. Zum einen müssen alle Module in das Zeitraster eingepaßt werden, das sich im allgemeinen an der Aufgabe mit der höchsten Bearbeitungsfrequenz orientiert. Zum anderen liegt vielen Aufgaben ein nicht zyklischer

Auslöser zugrunde wie z.B. die Betätigung des Schalthebels. Dies ist aus der Struktur der Software nicht erkennbar, und die auslösenden Faktoren müssen als globale Variablen der ausführenden Funktion übergeben werden.

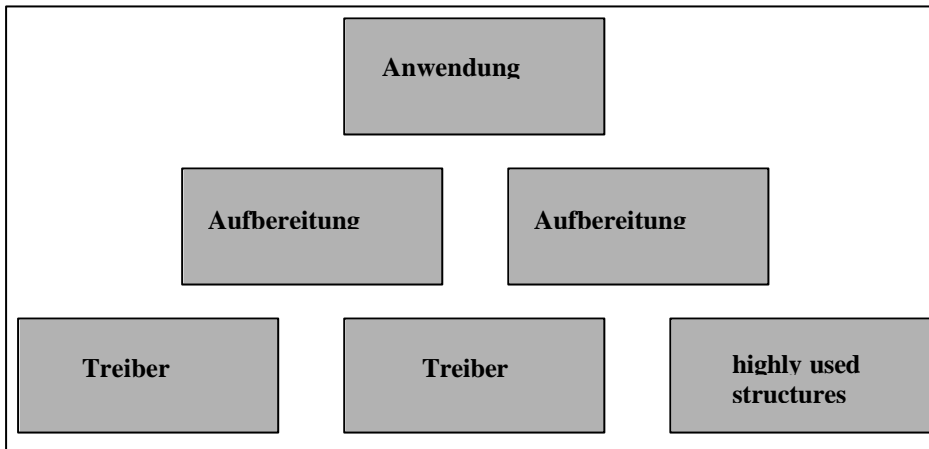


Abb. 2: Hierarchiebildung in Softwarestrukturen

Das Betriebssystem OSEK bietet neben der expliziten Taskaktivierung den Mechanismus des „Events“, um den Ablauf einer Task zu steuern. Hiermit können die Reaktionen auf unregelmäßige Signale genauso nachgebildet werden wie für zyklische Signale. Die Verwendung von globalen Zustandsvariablen läßt sich weitgehend vermeiden.

3.5 Vorgehensweise bei der Strukturierung

Der erste Schritt ist die Aufteilung der Gesamtaufgabe in eine Vielzahl von Teilaufgaben. Die Anzahl der Teilmodule hat zu diesem Zeitpunkt noch keine Bedeutung. Wichtig ist, daß Funktionsblöcke gebildet werden, die ein homogenes Zeitverhalten und einen minimalen Datenaustausch mit anderen Funktionsblöcken aufweisen. Für die Unterteilung bieten die folgenden Kriterien eine Hilfestellung:

- Wie wird der Funktionsablauf ausgelöst (z.B. zyklisch oder durch ein spezielles Ereignis bzw. durch einen Zustand)?
- Wie schnell muß auf das auslösende Ereignis reagiert werden? Hieraus leitet sich auch die Priorität des Funktionsblocks ab?
- Muß der Teilalgorithmus aus Gründen der Datenkonsistenz oder Hardwarezugriffen geschützt werden (z.B. gegen Unterbrechungen durch andere Softwaremodule)?

Die Vielzahl der Teilmodule wird im nächsten Schritt in ein hierarchisches Schema (vgl. Abb. 3) gebracht. Damit wird nicht nur eine bessere Übersichtlichkeit und Lesbarkeit erreicht, sondern auch eine zu starke Verflechtung der Teilmodule untereinander

verhindert. Der intensive Gebrauch von globalen Variablen zur Ablaufsteuerung ist häufig ein Zeichen dafür, daß Algorithmen aufgeteilt wurden, die vom Ablauf her zusammengehören, oder zumindest die Abhängigkeiten verschiedener Module nicht deutlich dargestellt werden. Da globale Variablen prinzipiell der Gefahr des fehlerhaften Beschreibens unterliegen, ist diesem Punkt unbedingt Beachtung zu schenken.

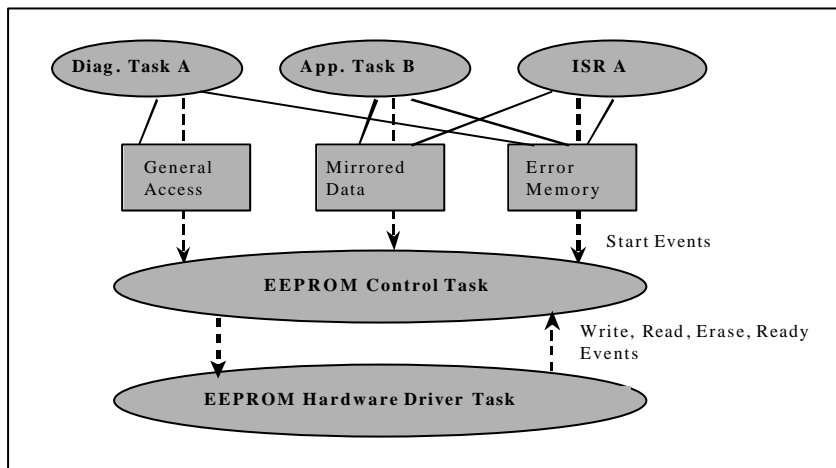


Abb. 3: Struktur einer EEPROM-Handler Task

Selbstverständlich haben die Kenngrößen Speicher- und Rechenzeitbedarf Priorität vor einer akademisch optimalen Softwarestruktur. Deshalb müssen in einem iterativen Prozeß die eingesetzten Betriebssystemobjekte, auf ein sinnvolles Maß reduziert werden. Dies sind für das Betriebssystem OSEK vor allem die Objekte „Task“, „Event“ und „Alarm“. Da die Verwaltung dieser Objekte meistens in Bitfelder organisiert ist, wird eine Optimierung immer dann erreicht, wenn der Sprung über eine Bytegrenze möglich ist. Ein Beispiel wäre die Reduzierung der Anzahl der Tasks von 17 auf 16.

3.6 Strukturvergleich

Die positiven Eigenschaften des Betriebssystem OSEK lassen sich sehr deutlich am Beispiel einer Schreibroutine für EEPROMs zeigen (Abb. 3). Solche Schreibroutinen werden z.B. für die Ablage von Wartungsinformationen benötigt. Interessant bei diesen Aufgaben sind die zeitlichen Anforderungen:

- Die Daten fallen unregelmäßig mit zum Teil sehr großen Abständen (Minuten bis Stunden) an.
- Die Reaktion auf neue Daten soll schnell erfolgen, um Datenverlust zu vermeiden.
- Der Schreibvorgang ist vergleichsweise lang (ca. 10 ms pro Byte).

In der herkömmlichen Programmierweise wird eine solche Routine als Zustands-

maschine dargestellt, die zyklisch zur Zustandsüberprüfung aufgerufen wird. Dabei richtet sich die Zykluszeit nach der kürzesten erforderlichen Reaktionszeit. Dies bedeutet aber, daß die Routine in den Wartezeiten sehr häufig aufgerufen wird, obwohl keine neuen Daten oder Zustandsänderungen vorliegen.

Im Betriebssystemstandard OSEK ist der Event-Mechanismus eine elegante Art um auf derartige asynchrone und evtl. quasi-simultane Anforderungen zu reagieren. Dabei ist der Event kein einfacher Trigger, sondern liefert zusätzlich eine unterscheidbare Information über seine Herkunft. Damit kann in jedem Zustand gezielt auf auftretende Ereignisse reagiert werden.

In dem Beispiel der EEPROM Task erfolgt eine Aktivierung der Routine erst bei Eintreffen eines neuen Schreibauftrags. Nach dem Aktivieren des EEPROM internen Schreibvorgangs startet ein Zeitgeber und die Routine wechselt in den Wartezustand. Nach Ablauf des Zeitgebers wird der Erfolg des Schreibvorgangs geprüft und die Routine beendet. Dieses Vorgehen hat folgende Vorteile:

- die Routine wird nur aufgerufen, wenn sie benötigt wird,
- die Struktur der EEPROM Programmierung wird erkennbar,
- die Routine wird nicht in das Zeitraster anderer Vorgänge gepreßt,
- es werden wenig globale Zustandsvariablen verwendet (data hiding).

Zur Beurteilung der erreichbaren Vorteile wurde eine bisher verwendete Schreibroutine umgeschrieben auf das Betriebssystem osCAN, das konform mit dem OSEK-Standard ist. Die erreichten Reduzierungen sind in Abb. 4 dargestellt.

Als primäre Größe wurde die Reduzierung des Quellcodes bestimmt. Es ergab sich eine deutliche Abnahme um 16 % im Vergleich zur zyklisch programmierten Software. Die verwendeten Betriebssystemroutinen sind hierin nicht berücksichtigt. Das obige Beispiel zeigt jedoch, daß bei größeren Applikationen mit mehreren derartigen Modulen der Mehraufwand für das Betriebssystem kompensiert, wenn nicht sogar übertroffen werden kann, da die Reduzierung des Softwarequellcodes dem Overhead durch Betriebssystemroutinen überwiegt.

Ein weiterer Aspekt mit hoher Bedeutung ist die Reduzierung der Komplexität. Weniger Code ermöglicht eine bessere Überschaubarkeit, das Programm ist dadurch leichter lesbar und schneller programmierbar. Als Indikator dient der Verschachtelungsgrad. Für diesen wurde als Maßzahl McCabe's Cyclomatic Complexity verwendet. Dabei ergab sich ein Rückgang der Komplexität um 34 %.

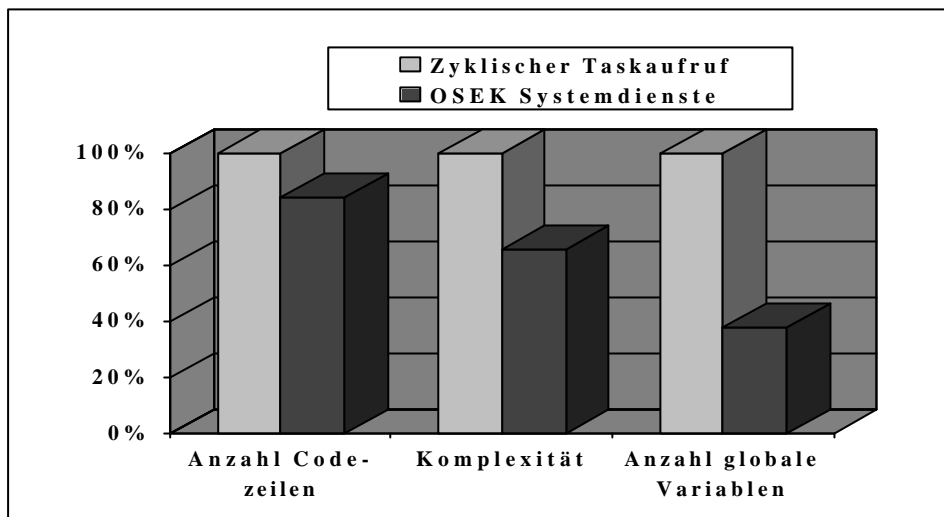


Abb. 4: Vergleich von zyklischer und Betriebssystem Implementierung der EEPROM Task

Eine Vermeidung globaler Variablen wurde nicht vollständig erreicht, da sie die effizienteste Methode für die Datenübergabe darstellen. Insgesamt konnte ihre Anzahl wegen des vollständigen Verzichts auf globale Variablen in der Ablaufsteuerung stark reduziert werden.

Ein Vergleich der CPU-Belastung wurde nicht durchgeführt, da sich keine eindeutigen Meßbedingungen angeben lassen. Der entscheidende Unterschied liegt darin, daß bei dem herkömmlichen zyklischen Verfahren eine kontinuierliche Grundlast vorhanden ist, wohingegen bei Verwendung des Betriebssystems die CPU nur im Bedarfsfall beansprucht wird.

Eine EEPROM Schreibroutine ist eine typische Aufgabe, wie sie in jeder Applikation zu finden ist. Die Vorgehensweise läßt sich auf alle Funktionen mit unregelmäßigem Zeitverhalten übertragen. Aber auch für zyklische Prozesse bietet das Betriebssystem ein Grundgerüst, das den Programmierer von Standardaufgaben entlastet.

4 Echtzeitanwendung am Beispiel eines Hybridfahrzeugs

Am Beispiel des Hybridfahrzeugs INMOVE im Rahmen eines EU-Projektes wird die Vorgehensweise der Erstellung einer Echtzeitanwendung auf Basis des OSEK-Betriebssystems erläutert. Zunächst soll die Struktur des Hybridfahrzeugs vorgestellt und anschließend auf die Realisierung der Anwendung auf Basis des OSEK-Betriebssystems eingegangen werden.

4.1 Struktur des Hybridfahrzeugs INMOVE

Bei dem erwähnten Hybridfahrzeug INMOVE handelt es sich um ein Einwellenparallelhybridfahrzeug. Das Antriebssystem besteht aus einem konventionellen Verbrennungsmotor mit einer Leistung von 55 KW und einem Elektromotor mit einer Leistung von 30 KW. Die Antriebsleistung kann sowohl vom Elektro- oder Verbrennungsmotor alleine als auch in Kombination beider Antriebssysteme erbracht werden. In der folgenden Abbildung ist der Antriebsstrang mit seinen wesentlichen Komponenten dargestellt.

Die mechanische Kupplung und das Getriebe mit Differential wurden aus den Komponenten der Serie abgeleitet, wobei die Kupplung mit einem elektrischen Aktuator ausgestattet wurde. Der Elektromotor ist drehfest mit der Getriebeeingangswelle verbunden, sein Gehäuse ist an das Getriebegehäuse angeflanscht. Der mechanische Rückwärtsgang sowie der ursprünglich vorhandene 5. Gang wurden entfernt, so daß Rückwärtsfahrt ausschließlich elektrisch möglich ist. Durch die hier beschriebene Anordnung bewirkt der sich auf der Getriebeeingangswelle befindliche Elektromotor eine Erhöhung des Trägheitsmoments um ein Vielfaches.

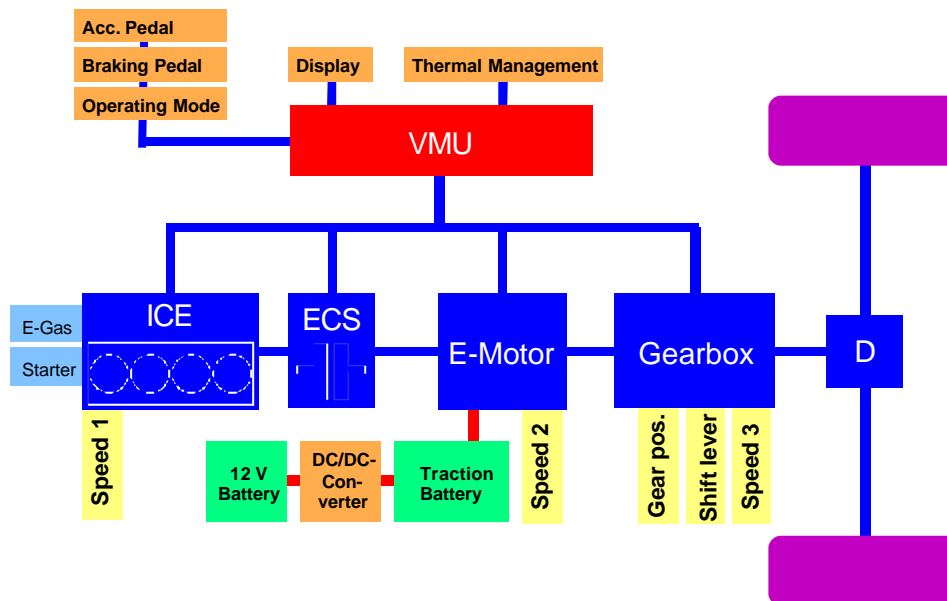


Abb. 5: Struktur des Einwellenparallelhybridfahrzeugs INMOVE

Der Startvorgang des Verbrennungsmotors erfolgt in einem ersten Schritt mit dem konventionellen Starter. In einem nächsten Schritt wird der Verbrennungsmotor durch den Elektromotor angeschleppt, so daß der Starter entfallen kann. Die Ansteuerung des Verbrennungsmotors geschieht durch ein E-Gassystem.

Die 2 Motoren sowie das elektronische Kupplungssystem und das Batteriemangement werden durch entsprechende Steuergeräte angesteuert. Mit Ausnahme des Verbrennungsmotors erfolgt die Kommunikation der Steuergeräte untereinander sowie mit dem zentralen Fahrzeugsteuerrechner (VMU = Vehicle Management Unit) über ein

CAN Hochgeschwindigkeitsnetzwerk mit 500 kBit/s.

Die VMU ist Bestandteil eines CAN-Bussystems und spielt eine systemübergreifende Rolle. Im Gegensatz zu einem konventionellen Fahrzeug hat der Fahrer keinen direkten Einfluß auf die Leistungsabgabe der Antriebsaggregate. Diese Aufgabe obliegt der VMU. Sie interpretiert aus der Gaspedalstellung den Fahrerwunsch und setzt diesen in eine adäquate Antriebsleistung um. Diese ist abhängig von der Höhe des Bedarfsmoments, der Fahrzeuggeschwindigkeit und vom Ladezustand der Traktionsbatterie.

Bei der Interpretation des Fahrerwunsches steht eine hohe Gesamteffizienz des Antriebssystems sowie die Unabhängigkeit von externen Ladegeräten für die Traktionsbatterie im Vordergrund. Der hier eingesetzte Verbrennungsmotor zeigt bei Momenten unterhalb von 40 Nm einen sehr schlechten Wirkungsgrad. Aus diesem Grund stellt in diesen Lastbereichen der Elektromotor seine Antriebsleistung zur Verfügung. Erst bei höheren Bedarfsmomenten wird der Verbrennungsmotor zugeschaltet.

In diesen Bereichen höherer Bedarfsmomente kommt die Methode der Momentenaufteilung zum Tragen, so daß der Verbrennungsmotor nur in Bereichen mit einem guten Wirkungsgrad betrieben wird [2]. Über das Fahrerwunschemoment hinaus erzeugte Leistung wird über den als Generator betriebenen Elektromotor in die Traktionsbatterie zurückgespeist. Auf diese Weise kann die Traktionsbatterie im Fahrbetrieb auf einem hohen Ladezustand gehalten werden, wodurch immer eine ausreichende reine elektrische Reichweite garantiert wird.

Eine Auflistung zeigt die Vielfalt der Aufgaben der VMU:

- Interpretation des Fahrerwunsches (Antriebsleistung).
- Kennfeldgesteuerte, verbrauchsoptimierte Betriebsstrategie bestimmt Antriebsleistung von Verbrennungs- und Elektromotor.
- Batteriemangement (Nivellierung des Ladezustands)
- Startvorgang des Verbrennungsmotors
- Schaltvorgang (zeitkritisch)
- Kommunikation in Echtzeit mit den Steuergeräten
- Fahrtrichtungsumkehr (Vor- und Rückwärtsfahrt)
- Fahrmoduswahl (hybrid oder rein elektrisch)
- Thermisches Management

- Displayansteuerung

4.2 Vorgehensweise zur Erstellung einer OSEK-Anwendung

In diesem Abschnitt wird auf die Vorgehensweise beim Erstellen einer komplexen Anwendung mittels eines OSEK-Betriebssystems eingegangen.

4.2.1 Taskstruktur

Die Konfiguration des Betriebssystems wird weitgehend durch die Programmstruktur festgelegt. Der erste Schritt beim Aufbau des Programms besteht in der Festlegung der Taskstruktur. Wie bereits erwähnt, werden in einer Task zusammengehörige Aufgaben oder auch Aufgaben, die zu gleichen Zeiten bearbeitet werden müssen, zusammengefaßt. Abbildung 6 stellt die hier vorgestellte Taskstruktur dar. Sie enthält eine Übersicht über die Aufgaben der jeweiligen Task sowie eine Darstellung der Synchronisation der Tasks untereinander.

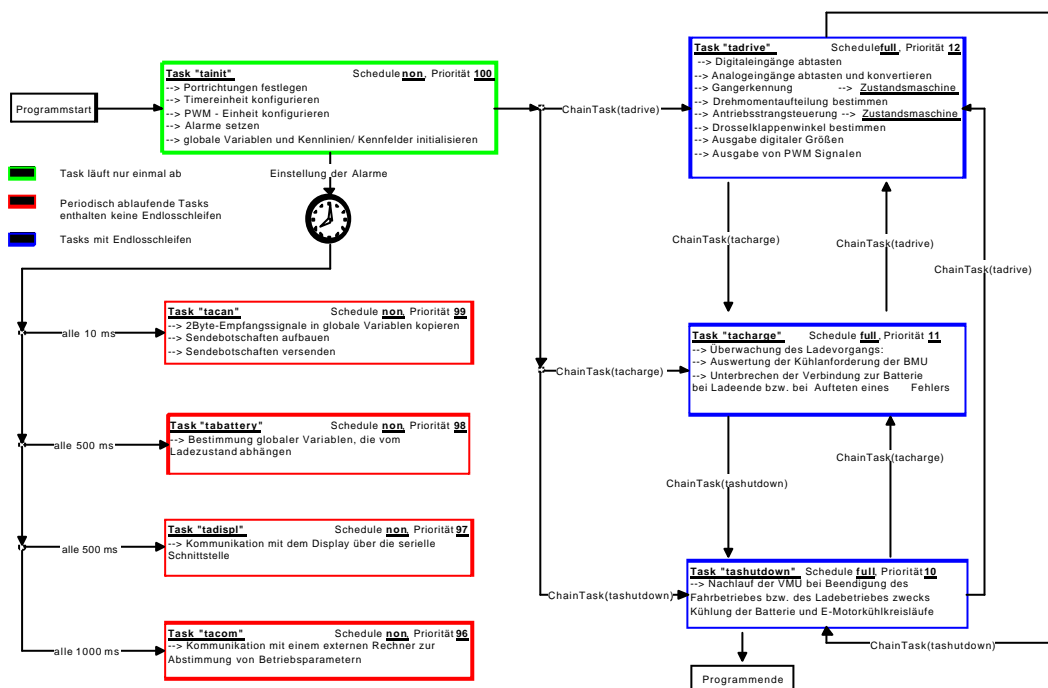


Abb. 6: Taskstruktur des Hybridfahrzeugs INMOVE

Im zweiten Schritt muß das Zeitverhalten festgelegt werden, indem jeder Task bestimmte Eigenschaften zugewiesen werden. In der vorliegenden Anwendung kommen ausschließlich Basic Tasks zum Einsatz. Der Eventmechanismus zur Synchronisation der Tasks wird nicht angewendet. Bei Programmstart wird zunächst die Task „taint“ vollständig durchlaufen. Diese Task erhält die höchste Priorität und wird bei Programmstart automatisch vom Betriebssystem aktiviert. Die verbleibenden Tasks können in zwei Kategorien unterteilt werden.

Die erste Kategorie enthält Tasks, die keine Endlosschleifen besitzen und nicht preemptiv sind, d.h. sie können auch von Tasks höherer Priorität nicht unterbrochen werden. Nach erfolgtem Durchlauf terminieren sich diese Tasks von selber und geben den Prozessor wieder frei. Die Aktivierung dieser Tasks erfolgt in regelmäßigen Zeitabständen durch das Betriebssystem über Alarme. Diese regelmäßige Aktivierung spielt vor allem für die Task „tacan“ eine wichtige Rolle, da einige über CAN verbundene Komponenten in regelmäßigen Zeitabständen eine CAN Botschaft erwarten. Die beabsichtigt unterbundene Unterbrechbarkeit dieser Tasks stört den Ablauf der vorher gerade laufenden Task nur unwesentlich, da es sich ausschließlich um eine Task mit sehr kurzer Rechenzeit handelt. Der Vorteil dieser nicht möglichen Unterbrechbarkeit ist, daß keine Datenkonsistenzprobleme auftreten.

Die Tasks der zweiten Kategorie enthalten Endlosschleifen und sind preemptiv. Je nachdem, ob sich das Fahrzeug gerade im Ladezustand, im Fahrzustand oder in der Nachlaufphase befindet, ist entweder die Task „tadrive“, die Task „tacharge“ oder die Task „tashutdown“ aktiv. Bei Programmstart wird eine dieser drei Tasks, in der Regel die Task „tadrive“, von der Task „tainit“ aus aktiviert. Aufgrund der Endlosschleife läuft diese Task permanent ab und wird zu bestimmten Zeiten kurzzeitig durch eine periodisch ablaufende Task unterbrochen. Die gegenseitige Aktivierung dieser drei Tasks geschieht selbständig mit Hilfe der Betriebssystemfunktion „ChainTask()“. Wenn z.B. während des Fahrbetriebs der Ladestecker eingesteckt wird, so erkennt dies die momentan aktive Task „tadrive“, und die Task „tadrive“ terminiert sich selbst, während sie gleichzeitig die Task „tacharge“ als ihren Nachfolger aktiviert. Natürlich müssen an den Stellen, an denen mehrere Byte große globale Variablen manipuliert werden, besondere Maßnahmen ergriffen werden, um Datenkonsistenz zu gewährleisten.

Das Programm kann nur von der Task „tashutdown“ aus verlassen werden.

4.2.2 OIL-Konfigurator

OSEK ist ein statisches Betriebssystem, d.h. die Festlegung aller verwendeten Betriebssystemmittel erfolgt zur Laufzeit. Auf eine dynamische Erzeugung von Systemobjekten wurde aus Laufzeitgründen verzichtet. Um die Portierbarkeit und eine einfache Integration von Anwendungen zu erreichen, muß zusätzlich eine plattformunabhängige und herstellerübergreifende Beschreibung der Parameter für die Systembeschreibung möglich sein.

Diese Anforderungen führten zur Spezifikation der Beschreibungssprache OSEK-Implementation-Language (OIL). Mittels des OIL-Konfigurators wird eine Systemheaderdatei mit implementierungsspezifischen Datentypen und Code erzeugt, die zur Verbindung der OSEK Bibliotheken mit der Anwendung dienen.

Durch die standardisierte OIL Syntax wird die Beschreibung der Anwendung weitgehend unabhängig von der jeweiligen OSEK Implementierung.

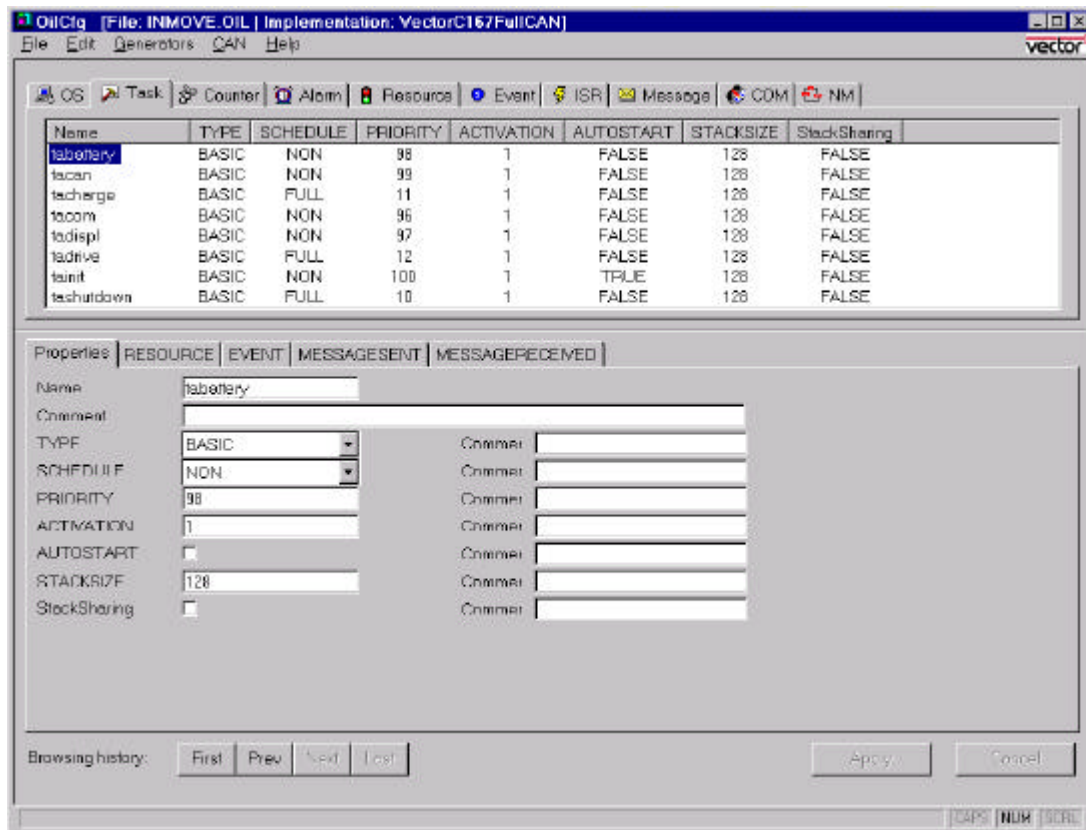


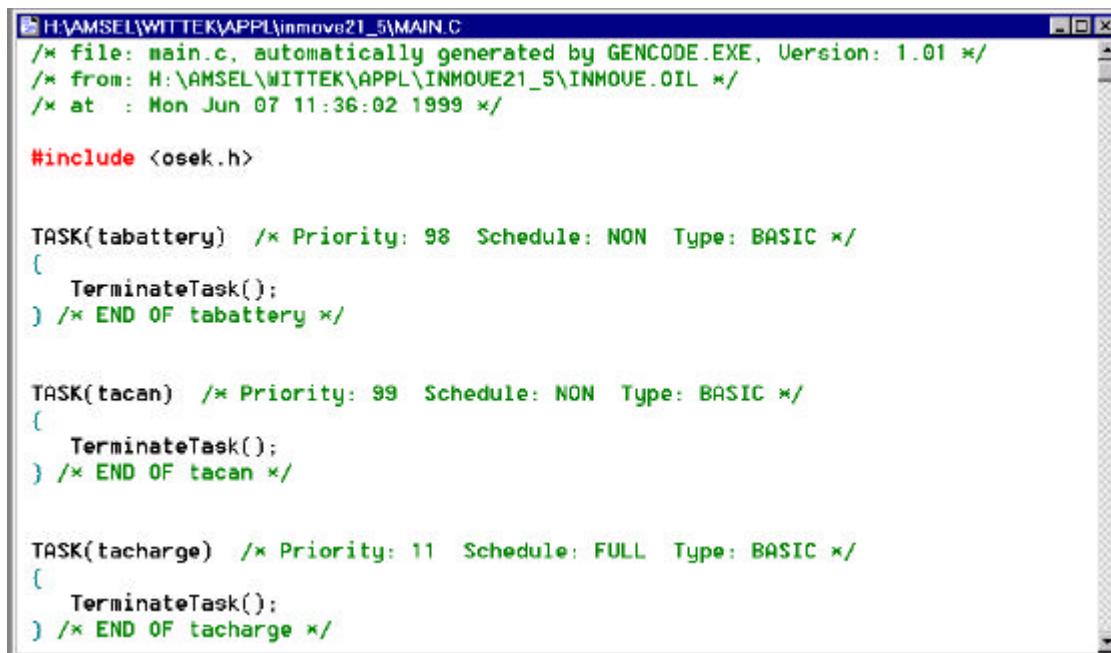
Abb. 7: OIL-Konfigurator

Im in Abb. 8 dargestellten Ausschnitt des OIL-Konfigurators finden sich alle zuvor in der Taskstruktur vorgestellten Tasks wieder. Gut erkennbar ist, daß die Eigenschaften, z.B. Scheduling und Prioritäten, einer Task sehr leicht über entsprechende Auswahlfenster ausgewählt und zugewiesen werden können.

Weiterhin besteht auch die Möglichkeit bei Kommunikation mit anderen Steuergeräten eine sogenannte Datenbasis einzubinden. In dieser Datenbasis befinden sich alle zu versendenden und zu empfangenden Botschaften. Einige Toolhersteller bieten zur Erstellung dieser Datenbasis, z.B. für CAN, Editoren an.

4.2.3 Templategenerierung

Der letzte Schritt vor dem Erstellen des Anwendercodes stellt die Templategenerierung dar. Aus dem OIL-Konfigurator wird dieses Template automatisch generiert. Es stellt einen Rahmen zur Verfügung, in den der Anwender lediglich noch seinen Code eintragen muß.



```
H:\AMSEL\WITTEK\APPL\inmove21_5\MAIN.C
/* file: main.c, automatically generated by GENCODE.EXE, Version: 1.01 */
/* from: H:\AMSEL\WITTEK\APPL\INMOUE21_5\INMOUE.OIL */
/* at : Mon Jun 07 11:36:02 1999 */

#include <osek.h>

TASK(tabattery) /* Priority: 98 Schedule: NON Type: BASIC */
{
    TerminateTask();
} /* END OF tabattery */

TASK(tacan) /* Priority: 99 Schedule: NON Type: BASIC */
{
    TerminateTask();
} /* END OF tacan */

TASK(tacharge) /* Priority: 11 Schedule: FULL Type: BASIC */
{
    TerminateTask();
} /* END OF tacharge */
```

Abb. 8: Templategenerierung

Der große Vorteil liegt darin, daß die generierte Rahmenstruktur des Programms in Form des Templates fehlerfrei erzeugt wird. Alle zuvor eingestellten Parameter sind im OIL-File enthalten.

6 Zusammenfassung und Ausblick

In diesem Vortrag wurden die Beweggründe für die Wahl eines OSEK Betriebssystems vorgestellt. Eine wichtige Rolle spielten hierbei die Aspekte der Erweiterbarkeit, Portierbarkeit und Wiederverwendbarkeit von Software. Beispielhaft konnte aufgezeigt werden, daß bei Einsatz eines OSEK Betriebssystems die Anzahl der Codezeilen sowie die Komplexität der Software deutlich reduziert werden konnte. Auch die Anzahl der globalen Variablen wurde stark reduziert, da für die Ablaufsteuerung vollständig auf globale Variablen verzichtet werden konnte.

Anwendungen, die mit Hilfe eines OSEK-Betriebssystems erstellt werden, werden durch die vorhandenen Tools, wie z.B. den OIL-Konfigurator und Datenbasiseditoren, gut unterstützt. Auf diese Weise kann eine Anwendung schnell und übersichtlich skaliert und erstellt werden. Dies wurde am Beispiel des Projekt INMOVE demonstriert.

Problematisch stellt sich die Aufteilung des Gesamtprogramms auf einzelne Tasks dar. Auch am Institut für Kraftfahrwesen Aachen werden komplexe Betriebsstrategien beispielsweise für Hybridfahrzeuge vor der Softwarerealisierung mittels Simulation (z.B. Matlab/Simulink) entwickelt und optimiert. Hierbei häufige Verwendung finden sogenannte Stateflows oder auch Zustandsdiagramme bzw. -maschinen. Eine Aufteilung dieser Zustandsmaschinen auf eine Vielzahl von unabhängigen Tasks stellt sich zum Teil als recht schwierig dar.

Zur Erhaltung der Datenkonsistenz werden dann doch wieder eine Vielzahl von Zustandsvariablen benötigt. Hilfreich ist hier sicherlich der Einsatz von Codegeneratoren, die aus dem Simulationstool lauffähigen Code erzeugen.

7 Literatur

- [1] Kuder, H.; Vetter, J.
OSEK-Erfahrungen mit dem Betriebssystem und Systemgenerierung
VDI Berichte Nr. 1415
Baden-Baden, 1998

- [2] Rütthlein, A.; Bady, R.; Renner, C.
INMOVE - A Single Shaft Parallel Hybrid Concept
1999 Global Powertrain Congress
Stuttgart, 1999

- [3] Amsel, C.; Brock, H.; Crampen, A.; Renner, C.
Realisierung einer Echtzeitanwendung auf Basis des OSEK Betriebssystems
XIX. Tagung Elektronik im Kraftfahrzeug
Essen, 1999

- [4] Kincke, K.; e.a.
OSEK - Ein Gemeinschaftsprojekt in der deutschen Automobilindustrie
VDI Berichte Nr. 1152
Baden-Baden, 1994

- [5] OSEK/VDX Operating System
Version 2.0 revision 1

- [6] OSEK/VDX Communication
Version 2.1 revision 1

- [7] OSEK/VDX Network Management
Version 2.5 revision 1