

Modellbasierte Softwareentwicklung für Embedded Systems

Werkzeugkette für sämtliche Entwicklungsphasen

Am Beispiel des technischen Stands von Kraftfahrzeugen (70 Steuergeräte und mehr) wird deutlich, dass Embedded Systems für viele Produkte eine herausragende Bedeutung haben. Die zügige und sichere Umsetzung der Produktanforderungen in leistungsfähige, betriebssichere und erweiterbare Soft- und Firmware ist ein sehr wichtiger Faktor für den Erfolg des Gesamtprodukts. Hier bietet ein modellbasierter Ansatz entscheidende Vorteile. Besonders die automatische Codegenerierung führt zu teilweise dramatischen Kosteneinsparungen bei der Entwicklung und Pflege des Endprodukts.

DIPL.-ING. CORD ELIAS

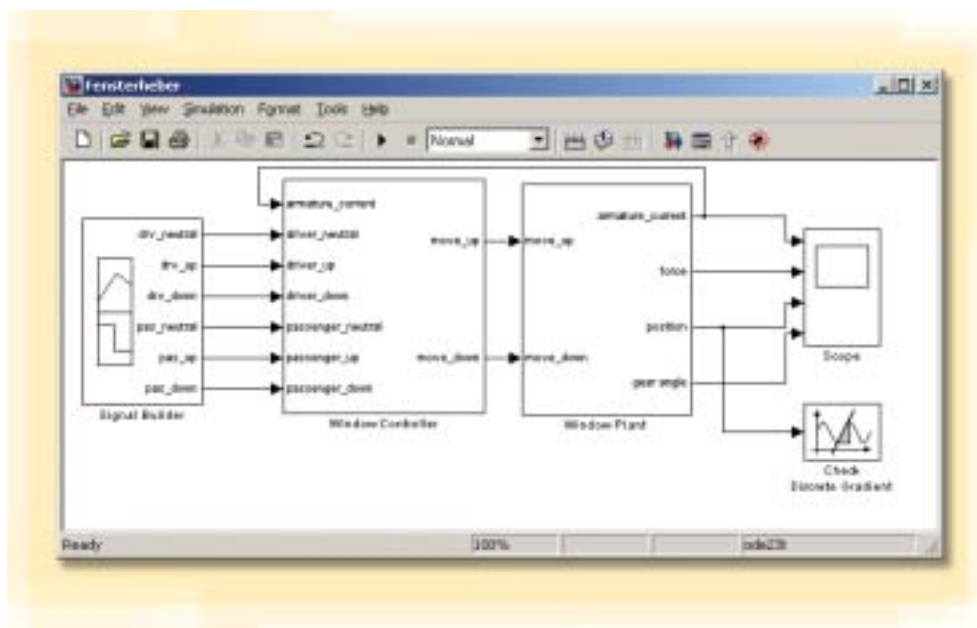


Abb. 1: Fensterheber, oberste Hierarchieebene

Die ‚klassische‘ Form der Softwareentwicklung für Embedded Systems ist im Wesentlichen dadurch gekennzeichnet, dass die verschiedenen Prozessschritte von der Analyse bis zur Systemintegration über Spezifikationen in Form von Text, Pseudo-Code und verschiedene Diagrammartentypen miteinander verknüpft sind. Bei jedem Übergang in eine andere Prozessphase ergibt sich ein Bruch, da die Ergebnisse der vorherigen Phase durch die Anwender interpretiert und in eine andere Darstellung umgesetzt werden müssen. Dass sich an diesen Schnittstellen Fehler einschleichen, ist nur allzu

menschlich. Der größte Teil des Aufwands wird in die manuelle Erstellung des Programmcodes für die Zielhardware gesteckt. Dabei geht es nicht nur darum, möglichst guten Code zu schreiben und diesen auf der Zielplattform ‚zum Laufen‘ zu bekommen. Es müssen in der Regel auch noch Fehler aus der Konzeptionsphase behoben werden, die aufgrund der oben beschriebenen Prozessstruktur vorher schlichtweg nicht entdeckt werden konnten.

Eine Alternative zu dem klassischen Ansatz stellt die modellbasierte Entwicklung dar, welche die o.g. Nachteile vermeidet und viele weitere Vorteile bietet. Sie basiert auf einer integrierten



Dipl.-Ing. CORD ELIAS arbeitet bei The MathWorks GmbH im Bereich Applikation Engineering. Sein Arbeitsschwerpunkt liegt auf dem Thema ‚Embedded Applications‘.

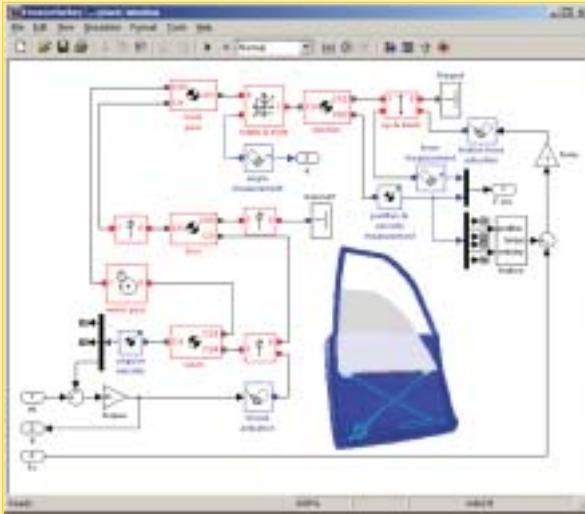


Abb. 2: Modellierung der Fensterhebermechanik

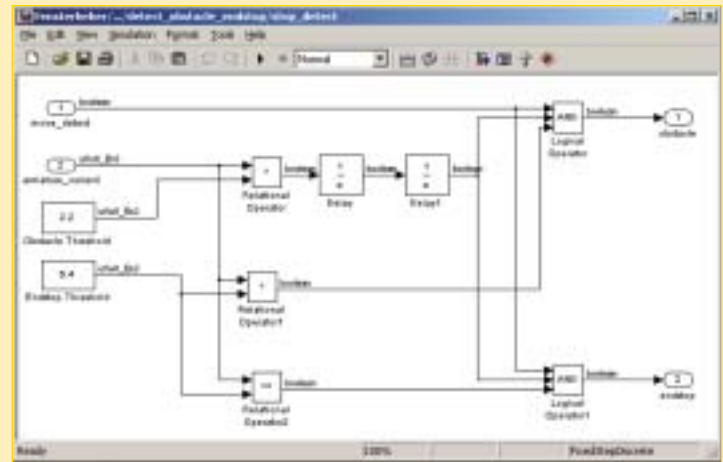


Abb. 4: Teilfunktion des ‚Window Controller‘

Werkzeugkette, die sämtliche Prozessphasen von der Spezifikation bis zur automatischen Codegenerierung abdeckt. Bewährt hat sich hier die Matlab/Simulink-Produktfamilie von The MathWorks, bestehend aus den Hauptkomponenten Simulink (Modellierung), Stateflow (Zustandsautomaten) und Real-Time Workshop Embedded Coder (automatische Codegenerierung für Embedded Systems). Es existieren Schnittstellen zu ergänzenden Werkzeugen von Drittanbietern (u.a. Requirements Engineering, automatische Generierung von Test-Vektoren, Configuration Management).

Modellierung

In der modellbasierten Entwicklung werden Blockdiagramme als einheitliche und durchgängige Darstellungsform für sämtliche Prozessphasen benutzt.

Abbildung 1 zeigt ein typisches Blockdiagramm. Es handelt sich um eine Automobilanwendung. Aufgabe ist es, die Komponente ‚Fenstersteuerung mit Einklemmschutz‘ für ein Türsteuergerät (auf Basis eines 8-Bit-Mikrocontrollers) zu entwickeln. Dazu sind in dem Blockdiagramm folgende Komponenten vorhanden:

- ▶ Subsystem ‚Window Plant‘: Hier wird das Verhalten der Strecke modelliert. In diesem Fall liegt eine analytische Beschreibung vor (weitere Möglichkeiten sind Systemidentifikation oder Anbindung an reale Hardware).
- ▶ Subsystem ‚Window Controller‘: Hier wird das Verhalten der zu implementierenden Steuerung komplett beschrieben.
- ▶ Signalquelle für Testsignale (‚Signal Builder‘): Hier werden verschiedene Fahrer-/ Beifahrerverhaltensweisen in Form von Testszenarios zur Verfügung gestellt.
- ▶ ‚Scope‘-Block: Ermöglicht die bequeme Visualisierung des Motorstroms etc.
- ▶ Verifikationselement (‚Check Static Range‘): Es wird automatisch überprüft, ob die Fenstergeschwindigkeit einen Grenzwert nicht überschreitet.

Das Blockdiagramm ist hierarchisch aufgebaut, ‚Plant‘ und ‚Controller‘ bestehen ebenfalls wieder aus Subsystemen.

Abbildung 2 zeigt einen Teil eines ‚Plant‘-Systems, hier wird die Fenstermechanik sehr komfortabel mit Elementen aus einer Mechanikbibliothek modelliert, so dass von dem Entwickler dieses Teilsystems keine Bewegungsgleichungen aufgestellt werden mussten. Die Entwicklung der

Steuerung (System ‚Window Controller‘) erfolgt nun in folgenden Schritten: Zunächst wird das Controllersubsystem im Detail erstellt. Dazu werden neben Basisblöcken wie Addieren, Multiplizieren auch State Charts (Zustandsdiagramme) eingesetzt. Im ersten Schritt wird für die Arithmetik die Gleitkommadarstellung verwendet, das vereinfacht den Designprozess in dieser Phase. Dabei kann zu jedem Zeitpunkt im Rahmen der Gesamtsimulation (vgl. Abb. 1) ein Test im Zusammenspiel mit dem Plant-Teil erfolgen.

Implementierung

Nachdem sichergestellt ist, dass die Anforderungen durch das gewählte Controllerdesign eingehalten werden, folgt die Implementierungsphase. Dazu sind u.a. folgende Schritte erforderlich:

- ▶ Automatisierte Umstellung des Controller-Modells von Gleitkomma auf Festkomma-/ Integer-Darstellung. Als Basis-Datentyp wird dabei der ‚natürliche‘ Datentyp des Zielprozessors gewählt, hier 8Bit-Integer.
- ▶ Iteratives Verfeinern des Modells, so dass sich unter Berücksichtigung der zur Verfügung stehenden Ressourcen der Zielhardware eine optimale Lösung ergibt. Dabei müssen Parameter wie Signal-Dynamik, Rechengenauigkeit, Datentypen und Rechenaufwand gegeneinander abgewogen und in ein stimmiges Verhältnis zueinander gebracht werden. Unterstützt wird dieser Prozess durch ein Werkzeug zur automatischen Bestimmung der optimalen Skalierung.
- ▶ Festlegung von Implementierungsdetails im Modell, u.a. die Zuweisung von Speicherklassen (Abb. 3) und die Aufteilung in Quelldateien
- ▶ Integration von Treibercode
- ▶ Optimale Parametrierung des Controllermodells für die automatische Codegenerierung. Dieser Vorgang wird durch ein interaktives

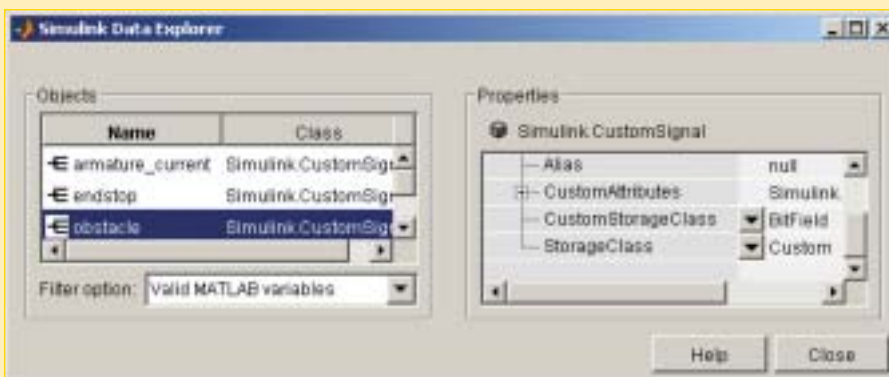


Abb. 3: Festlegung der Speicherklassen

Werkzeug unterstützt. Der Anwender legt beispielsweise fest, ob bzgl. RAM- oder ROM-Verbrauch optimiert werden soll.

Das so entstandene Teilmodell des Controllers enthält nun viele Implementierungsdetails. Trotzdem kann es noch für die reine Simulation verwendet werden, sogar ein Wechsel zwischen Gleitkomma- und Festkommaformat ist per ‚Schalter‘ jederzeit möglich. Dies ist besonders für die Produktpflege und -Weiterentwicklung sehr wichtig. Es ist auch einfach möglich, die Parametrierung mit Datentypen in generischer Art und Weise vorzunehmen. Für jede zu unterstützende Zielplattform wird dann ein eigener Datensatz mit Skalierungsinformationen angelegt, der zur individuellen Parametrierung des Modells aus einem Data-Dictionary geladen wird. Nach der vollständigen Parametrierung des Modells mit Implementierungsdetails erfolgt die automatische Codegenerierung – es ist lediglich ein Mausklick erforderlich.

Als Beispiel wird eine kleine Teilfunktion des Systems ‚Window Controller‘ betrachtet (Abb. 4). Einen Code-Auszug (ohne automatisch generierte Kommentare) zeigt Listing 1 (Kasten).

- ▶ Das Signal ‚armature_current‘ liegt als ‚unsigned 8 Bit‘-Größe vor. Die beiden Konstanten ‚Obstacle Threshold‘ und ‚Endstop Threshold‘ wurden als physikalische Größen

```
<Listing 1>
pwbits.obstacle = move_detect &&
(rtDWork.Delay1_DSTATE) &&
((uint8_T)(armature_current < 51U));

pwbits.endstop = move_detect &&
(rtDWork.Delay1_DSTATE) &&
((uint8_T)(armature_current >= 51U));

rtb_Delay = rtDWork.Delay_DSTATE;

rtb_Relational_Operator = (armature_current)
> (18U);

rtDWork.Delay1_DSTATE = rtb_Delay;

rtDWork.Delay_DSTATE =
rtb_Relational_Operator;
```

(Strom in Ampere) eingegeben, was für den Anwender sehr bequem ist. Für die Codegenerierung werden diese Konstanten vom System automatisch in die Zahlendarstellung des Vergleichs-Signals ‚armature_current‘ umgerechnet.

- ▶ Die beiden Ausgangs-Signale ‚obstacle‘ und ‚endstop‘ werden in einem Bitfeld abgelegt. Dies ist möglich durch die Verwendung von speziellen Speicherklassen (Custom Storage Classes), die durch den Anwender auf eigene Bedürfnisse anpassbar sind.

Nach der Codegenerierung kann der automatisch generierte Code in ein bereits vorhandenes

Code-Framework eingebunden werden, um die komplette Applikation zu erstellen. Es ist aber auch möglich, modellbasiert eine komplette Applikation zu erstellen – inklusive Gerätetreibern (ADC, PWM, etc.) und einem Scheduler, der präemptives Multitasking unterstützt.

In diesem Fall muss keine Zeile Code per Hand geschrieben werden, die Applikation wird komplett automatisch erstellt. Ermöglicht wird das durch ‚Embedded Targets‘, die für einen bestimmten Mikrocontroller bzw. DSP die entsprechenden Ressourcen zur Verfügung stellen.

Der modellbasierte Ansatz bietet für die Entwicklung von Software für Embedded Systems entscheidende Vorteile. Die automatische Codegenerierung ist dabei das letzte Glied in einer Kette von Werkzeugen, die sorgfältig aufeinander abgestimmt sein müssen. Es sollte besonders darauf geachtet werden, dass diese Werkzeugkette sämtliche Entwicklungsphasen gut unterstützt, die Basis-Werkzeuge (Simulations-Umgebung, automatische Codegenerierung) optimal aufeinander abgestimmt sind und die gesamte Entwicklungsumgebung offene Schnittstellen bietet.

Beitrag als PDF per ‚more@click‘:



Anzeige