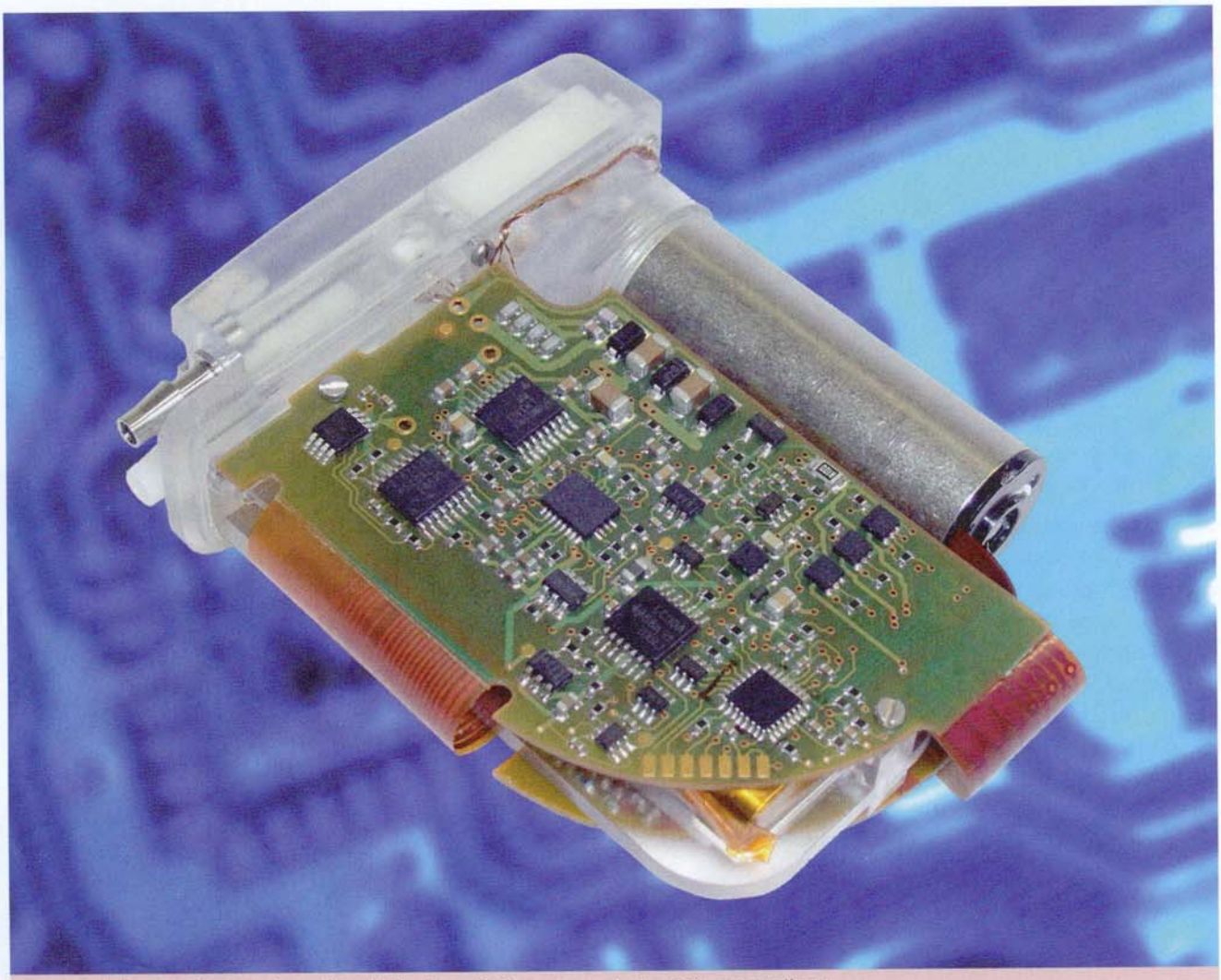
Programmiersprache C im Einsatz in medizintechnischen Systemen

Dr. Thomas Gillen, Rolf Schmid



Medizinisches Implantat mit zwei unabhängigen, in C programmierten Microcontrollern.

Kaum ein elektronisches Gerät arbeitet heute noch ohne Software. Programmierte Mikrocontroller finden sich heute selbst an Stellen, die noch vor wenigen Jahren die Domäne reiner Hardwarelösungen waren. Diese Entwicklung hat auch vor der Medizintechnik nicht Halt gemacht, und so finden sich die softwaregesteuerten Funktionen immer öfter auch in sicherheitskritischen medizinischen Systemen. Dementsprechend wachsen natürlich auch die Anforderungen an die Entwicklung von Software für Medizinprodukte.

Eine Reihe von Vorschriften und Hilfsmitteln sind in den letzten Jahren entstanden, um hier eine Qualität im SW-Bereich zu erreichen, die den Erfordernissen gerecht wird. Modellierungstools, Testtools, Bugreporting und Entwicklungsmodelle unterstützen den Entwickler wirkungsvoll bei fast allen Schritten des Entwicklungsprozesses.

Am Ende dieser Kette steht die Programmiersprache C und die entsprechenden C-Compiler. Genau hier verbergen sich diverse Fallgruben, die den Erfolg eines Projektes ernsthaft gefährden können. Zunächst ist da die Programmiersprache C selbst. Entwickelt in den späten 60er Jahren des letzten Jahrhunderts für die Hochsprachenimplementation des Betriebssystems Unix ist C heute die meist verwendete Sprache für kleinere Embedded-Systeme.

Durch den effizienten und kompakten Code ist sie für den Einsatz auf Embedded-Systemen geradezu prädestiniert. Allerdings ist nur wenigen Programmierern bewusst, dass die C-Compiler im Gegenzug praktisch keinen Code zum Erkennen und Abfan-

gen von Laufzeitfehlern generieren. Der bekannte Bufferoverflow aufgrund fehlender Unterstützung für die Bereichsüberprüfung von Arrays oder der gefürchtete Stackoverflow sind hier nur die bekanntesten Folgen.

Weiterhin sind aufgrund von Unklarheiten in der Sprachdefinition einige Funktionen von C stark abhängig von der jeweiligen Implementation, was zu unerwartetem Verhalten des generierten Codes führen kann. Hinzu kommt die enorme Flexibilität der Sprache C, die es einem Programmierer sehr leicht macht, einen potenziell gefährlichen Code zu erstellen.

Eine weitere Quelle möglicher Probleme sind die Compiler. Aufgrund der Vielzahl an Zielsystemen und der großen Anzahl an Compileranbietern wird eine spezifische Compiler/Zielsystem-Kombination möglicherweise nur von wenigen Programmierern genutzt.

Entsprechend groß ist daher die Wahrscheinlichkeit, dass ein Fehler im Compiler und somit im generierten Code gar nicht erst entdeckt wird. Aber auch bekannte Fehler werden je nach Relevanz erst nach einiger Zeit behoben. Zudem kann immer wieder ein gewisses chaotisches Verhalten beobachtet werden: Kleine Änderungen am Sourcecode können zu komplettem und komplexem Fehlverhalten an anderer Stelle führen. Diese Fälle sind nicht selten und treten in der Realität bei Embedded-Projekten immer wieder auf und müssen daher berücksichtigt werden.

Nur wenigen Programmierern sind diese Fallstricke bekannt, und noch weniger wissen, wie sie vermieden, behandelt oder umgangen werden können. Diesen inhärenten Problemen kann mit zwei einfachen, aber heutzutage wenig populären Maßnahmen begegnet werden: Reduktion der System-Komplexität und Beschränkung auf das unbedingt Notwendige.

Heutzutage soll jedes System alle Funktionen aufweisen oder zumindest so designt sein, dass später weitere Funktionen hinzugefügt werden können. Dies führt zu einem höchst komplexen und unübersichtlichen Code. Je umfangreicher aber ein Code ist, um so mehr Fehler können darin verborgen bleiben. Daher muss sorgfältig darauf geachtet werden, dass bei sicherheitsrelevanten Systemen nur die unbedingt nötige Funktionalität implementiert wird und somit die Codegröße klein bleibt.

Notfalls müssen die Komfortfunktionen von den kritischen Funktionen getrennt werden, im Extremfall durch Auslagern auf einen eigenen Prozessor. Dies erhöht zwar die Gesamtkomplexität, durch die saubere Trennung wird aber der kritische Teil überschaubarer, kann mit weniger Aufwand designt, implementiert, reviewt und getestet werden, sodass der resultierende Gesamtaufwand deutlich sinkt.

Weiter ist eine Beschränkung im Nutzungsumfang der verwendeten Funktionen der Programmiersprache C erforderlich, damit die unklar definierten und implementationsabhängigen Sprachkonstrukte weitgehend vermieden werden können. Es reduziert einerseits die damit verbundenen logischen Fehler und andererseits, durch einen intensiveren Gebrauch der verbleibenden Funktionen, die Wahrscheinlichkeit von unentdeckten Compilerfehlern.

Dies kann mit stringenten Coding-Guidelines erreicht werden. Leider zielen heute die meisten Coding-Guidelines mehr auf ein einheitliches optisches Erscheinungsbild und eine verbesserte Lesbarkeit des Quellcodes ab. Diese müssen so erweitert werden, dass auch der Umfang des zulässigen Sprachgebrauchs beschränkt wird.

Ein mittlerweile recht weit verbreiteter Coding-Standard ist Misra C, das speziell für sicherheitskritische Applikationen im Automobilbereich quasi verbindlich ist. Dieser Standard kann jedoch auch bei Software für die Medizintechnik erfolgreich eingesetzt werden.

All diese Maßnahmen ermöglichen, dass der Code mit überschaubarem Aufwand reviewt und gewartet werden kann und verschiedene Fehlerquellen vermieden werden können. Dies sind keine Garantien für eine fehlerfreie Funktion der Software, sie tragen aber durch Reduktion der Fehlerwahrscheinlichkeit dazu bei, dass insgesamt weniger Fehler unentdeckt bleiben.

Als Konsequenz ist daher festzuhalten, dass trotz all der leistungsfähigen Tools für Analyse, Design und Implementation von Embedded-Software die Erstellung von zuverlässiger Software für Embedded-Systeme nicht einfach ist. Jeder Programmierer von Embedded-Systemen muss sich bewusst sein, dass er einerseits mit einer für sicherheitsrelevante Systeme eher ungeeigneten Programmiersprache arbeitet, und dass zudem die eingesetzten Tools für die Codegenerierung mit Fehlern behaftet sein können. Ein sorgfältiger Systemtest anhand objektiver Testkriterien an einem konkreten Gerät ist daher unerlässlich, egal wie klein die vorgenommene Änderung ist. ■

Kontakt:

Art of Technology AG
CH-8005 Zürich
www.art-of-technology.ch