TDD in der Embeded-Praxis

Mehr Vertrauen in den Code mit automatisiertem Feedback

Daniel Penning - info@danielpenning.com

Eine testgetriebene Entwicklung kann die Softwarequalität maßgeblich steigern. Für ein Embedded-System sind automatisierte Tests oft nur mit Einsatz spezieller Testhardware möglich. Durch einen softwaregestützten Dual-Target-Ansatz können Tests ohne zusätzliche Hardware ausgeführt werden. Dieser Artikel argumentiert, dass dabei die ausschließliche Verwendung klassischer Unittests nicht ausreichend ist. Es wird aufgezeigt, wie eine umfassendere Testumgebung aufgebaut werden kann.

Einleitung

Agile Verfahren haben zu erstaunlichen Fortschritten in der Softwareentwicklung geführt. Trotz erwiesener Vorteile setzen sich SCRUM und ähnliche Verfahren zur Entwicklung eingebetteter Systeme allerdings nur langsam durch. Ein wesentlicher Grund sind spezielle "embedded"-Probleme wie Hardwareabhängigkeiten und die Ausführungsumgebung der Software.
Der Kern agiler Vorgehen liegt darin, Änderungen in den Anforderungen im Projektverlauf zu begrüßen (*embrace change*). Veränderungen an einer Codebasis können aber nur dann schnell erfolgen, wenn sichergestellt ist, dass die Anpassung keine bestehende Funktionalität beeinträchtigt. Daher wird ein besonderer Fokus auf den automatischen Test der Software gelegt. Ein schnelles und belastbares Feedback führt dazu, dass Entwickler die Auswirkungen von größeren Veränderungen schnell abschätzen können. Die Realisierung dieses Feedback-Elements stellt die zentrale Hürde in der erfolgreichen Umsetzung agiler Methoden für die Entwicklung eingebetteter Systeme dar. Fehlt dieses Element oder ist nur eine unzureichende Testabdeckung realisiert, kann das Potential derartiger Ansätze nicht voll ausgeschöpft werden.

Problemstellung und Analyse

Abbildung 1 zeigt das im Folgenden betrachtete System zur Regelung von Leistungselektronik. Es besteht aus einer Kombination von DSP und FPGA. Auf dem DSP läuft ein Echtzeitbetriebssystem (RTOS) und die in C++ entwickelte Anwendung. Das FPGA ermöglicht die zeitlich exakte Messung von Signalwerten und das Stellen von Schaltern über eine PWM mit genau definiertem Zeitverhalten. Für die Untersuchung wird als Leistungselektronik ein Tiefsetzsteller betrachtet.

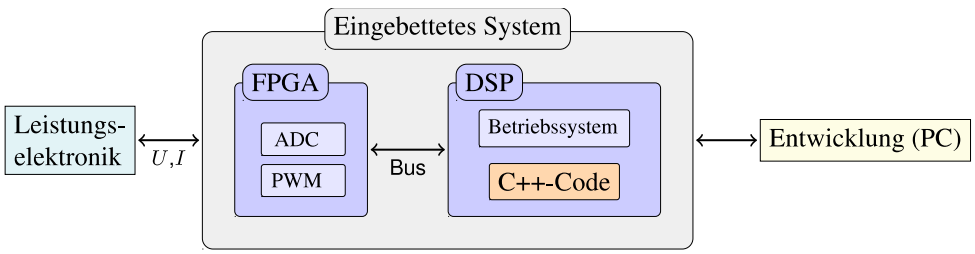

Abbildung 1: Betrachtetes Beispielsystem

Für dieses exemplarische System soll hier im Sinne der agilen Entwicklung ein testgetriebener Entwicklungsansatz vorgestellt werden. Wie eingangs beschrieben, erfordert dies die Bereitstellung eines schnellen und qualitativ hochwertigen Feedbacks. Dieses Feedback sollte zuverlässig Aufschluss über die Stabilität des aktuellen Softwarestandes liefern und damit beispielsweise folgende Fragen beantworten:

A. Verhalten sich einzelne Methoden wie gewünscht?
B. Wie verhalten sich verschiedene Klassen im Zusammenspiel?
C. Werden die nötigen Tasks in der richtigen Reihenfolge abgearbeitet?
D. Erfüllt eine im Code realisierte Regelung die Anforderungen?
E. Löst ein softwaregesteuerter Überlastschutz korrekt aus?

(A) und (B) sind das klassische Einsatzgebiet von Unittests. Methoden werden unabhängig voneinander auf bestimmte Erwartungswerte geprüft. Für das Isolieren von Abhängigkeiten werden Stubs und Mocks eingesetzt. Diese Tests werden bereits zunehmend in der Embedded-Praxis eingesetzt.
Da alle Hardware- und Systemabhängigkeiten isoliert sind, kann der zu überprüfende Code inklusive der Tests bereits für die Architektur des Entwicklungsrechners (PC) kompiliert und auch dort ausgeführt und somit überprüft werden.
Ausgezeichnete Literatur [1, 2] beschäftigt sich mit der praktischen Umsetzung, insbesondere der für Unittests nötigen Softwarearchitektur. Testframeworks wie googletest [3] unterstützen den Entwickler bei der Formulierung aussagekräftiger Unittests in C++.

(C) zielt auf das Zusammenspiel der Software mit dem RTOS ab. Die Taskpriorität, das eingesetzte Scheduling-Verfahren und die zeitliche Verfügbarkeit von Betriebsmitteln (Mutex, Queue, etc.) bestimmen maßgeblich die Ausführreihenfolge der Tasks. Fehler bei der Auslegung der Software auf eine Multitasking-Umgebung führen leicht zu Deadlocks, bei denen die komplette Ausführung unterbrochen wird. Die Abhängigkeit zum Betriebssystem und die zeitliche Dimension grenzen diese Integrationstests klar von klassischen Unittests ab. Nötige Testkonfigurationen müssen für das Zielsystem übersetzt und dort ausgeführt werden. Das Ergebnis muss zudem auslesbar sein. Daher lassen sich solche Tests üblicherweise nur schwer realisieren.

(D) und (E) beziehen die externe Umgebung in die Tests ein. Für bestimmte Ausgangssignale der externen Leistungselektronik wird erwartet, dass die Software mit definierten Stellsignalen reagiert. In der Praxis wird dieser Beweis üblicherweise mit Systemtests erbracht. Dazu wird entweder ein Hardwaresimulator oder - wenn möglich - die reale externe Hardware an das System angeschlossen und der Ablauf überwacht.

Das Konzept der klassischen Unittests lässt sich direkt auf eingebettete Systeme übertragen. Per definitionem können und sollen diese allerdings nicht die komplexe zeitliche Interaktion mit RTOS und externer Hardware prüfen. In der klassischen Softwareentwicklung spielen diese Abhängigkeiten häufig nur eine untergeordnete Rolle. Das Ziel von eingebetteten Systemen ist allerdings gerade die Interaktion mit dieser *einbettenden* Umgebung. Ein optimales Feedback lässt sich hier nur realisieren, wenn auch die Integrationstests (C, D, E) einbezogen werden.

Praktisch ist eine manuelle Testausführung auf dem Zielsystem mit hohem Zeit- und Kostenaufwand verbunden und kann daher nicht in der gewünschten Frequenz und kurzen Zeitspanne erfolgen. Daher muss eine Möglichkeit zur Automatisierung gefunden werden.

Lösung
Unittests lassen sich leicht realisieren, da sie auf dem PC ausgeführt werden können. Dazu wird ein sogenannter Dual-Target-Ansatz verwendet. Der eigentlich für das Zielsystem (hier der DSP) bestimmte Quellcode wird dabei zusätzlich für die Architektur des PC (bspw. x86) kompiliert. Für Integrationstests lässt sich dieser Ansatz nicht direkt übernehmen, da der Code Abhängigkeiten aufweist, die auf dem PC nicht vorhanden sind. Abbildung 2 zeigt diese Abhängigkeiten für das Beispielsystem.

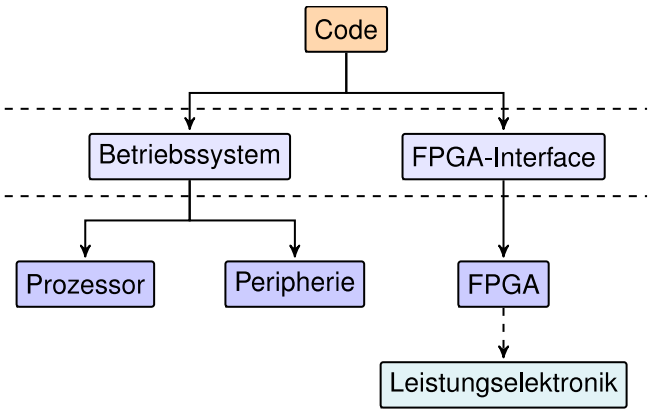

Abbildung 2: Hardware Abstraction Layer

Nahezu jedes größere Embedded-System verwendet bereits die hier mit HAL (Hardware Abstraction Layer) bezeichnete Abstraktion von der eigentlichen Hardware. Wird dieser HAL in einer ausreichenden Form auch für den PC zur Verfügung gestellt, so kann der gesamte Code bereits für den Entwicklungsrechner kompiliert und dort ausgeführt werden. Die nicht länger vorhandene Hardwareabhängigkeit beschleunigt den Entwicklungsprozess und ermöglicht ein optimales Feedback durch jetzt leicht automatisierbare Integrationstests.

Das verwendete RTOS muss zunächst für eine Ausführung auf dem PC (bspw. unter Microsoft Windows®) simuliert werden. Dazu können die Tasks mit den nativen Threads, die auf jedem Desktop-Betriebssystem bereitstehen, nachgebildet werden. Ein besonderes Augenmerk muss dabei auf die Realisierung des Schedulers gelegt werden. Das RTOS verwendet ein präemptives Multitasking, kann also laufende Tasks jederzeit unterbrechen. Dieses Verhalten kann unter Windows simuliert werden, wenn der Scheduler in einem eigenen hochpriorisierten Task arbeitet. Alle eigentlichen Programm-Tasks werden in niedrigpriorisierten Threads gekapselt. Ein weiterer hochpriorisierter Task kann die Generierung von Interrupts simulieren. Diese Nachbildung kann sich je nach Betriebssystem als komplex und zeitaufwendig erweisen. Für das populäre FreeRTOS™ existiert bereits eine offizielle Portierung für Windows [4].

Zusätzlich müssen nötige Treiber für die Peripherie mit sinnvollen Implementierungen bereitgestellt werden.

Das Beispielsystem interagiert mit der zu beeinflussenden Leistungselektronik ausschließlich über das FPGA. Integrationstests sollen das Verhalten einer Regelung überprüfen können. Daher muss neben der reinen Nachbildung der FPGA-Blöcke auch eine Simulierung der angeschlossenen Leistungselektronik erfolgen.

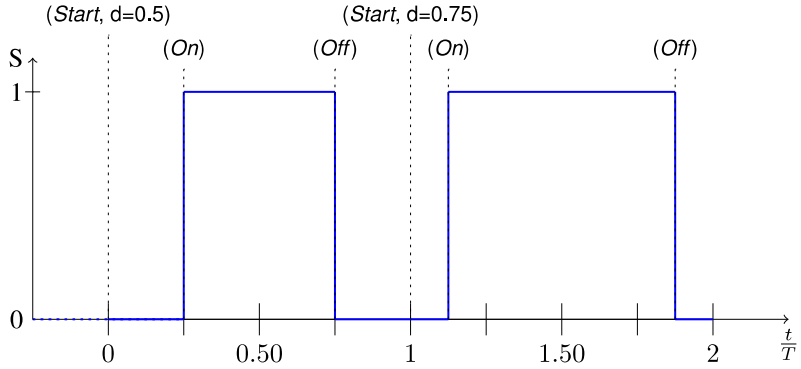

Abbildung 3: Zeitverhalten eines PWM-Blocks

Abbildung 3 zeigt das Verhalten des PWM-Blocks. Die Nachbildung muss den zeitlich deterministischen Charakter erhalten. Dazu wird die in Abbildung 4 gezeigte Struktur verwendet. Mit einer zeitdiskreten simulierten Systemzeit wird ein Environment Controller (EC) implementiert. Der EC steuert das Auslösen von zeitabhängigen Events, bspw. des On-Events (Setzen des Ausgangs auf einen High-Pegel). Der PWM-Block selbst wird gleichzeitig mit einem Modell des angeschlossenen Systems verbunden.

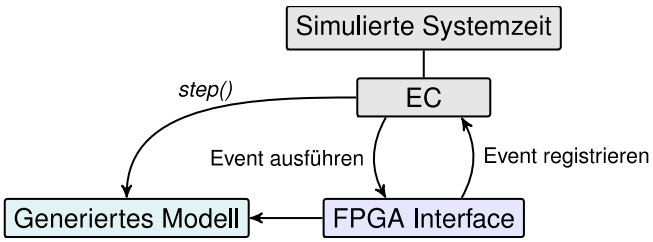

Abbildung 4: Verarbeitung zeitabhängiger Vorgänge

Aus der bereits durchgeführten Auslegung des Reglers stehen PLECS®-Modelle [5] für die Leistungselektronik – hier der Tiefsetzsteller - bereit. PLECS ermöglicht den Export dieser Modelle in einem ANSI-C kompatiblen Format. Die generierten zeitdiskreten Modelle können so leicht an den nachgebildeten PWM-Block angebunden werden. Das von der Anwendung benutzte FPGA-Interface kann nun auf ein realistisches Modell zugreifen. Stelleingriffe über die Beeinflussung des PWM-Tastgrades führen zu einer dynamischen Reaktion im Modell. Systemgrößen können über einen weiteren Analog-Digital-Block aus dem Code heraus ausgelesen werden.

Damit sind die Grundvoraussetzungen für Integrationstests geschaffen. Für die automatisierte Ausführung müssen die Signalverläufe von interessanten Kenngrößen aufgezeichnet und mit vorgegebenen Referenzsignalen abgeglichen werden. Dazu wurde ein hier nicht näher besprochenes Signalframework entwickelt, das die nötigen Informationen aus XML-Dateien ausliest, Signale während der Testlaufzeit aufnimmt und eine automatische Beurteilung vornimmt.

Beispiel

Abbildung 5 zeigt einen Test zur Prüfung der Abschaltlogik. Dabei wird zum Zeitpunkt t=4ms per Software das anliegende Modell so beeinflusst, dass es einen Lastkurzschluss simuliert. Die Abschaltlogik ist als Teil der Anwendung innerhalb des DSP in einem Regelungs-Task realisiert. An der Ausführung dieses Tests sind somit alle nachgebildeten Komponenten beteiligt. Weiterhin wird ein grafischer Export generiert, der in Abbildung 6 dargestellt ist. Aus diesem lässt sich der Erfolg/Nichterfolg des Tests verständlich nachvollziehen.

```
TEST(ControlSpecificationTests, Overload_EmergencyShutdown) {
  const double emergencyTime = 4E-3; // auch im XML definiert
  Callback emergencyAction =  std::bind(&BuckOLExtended::SetShortCircuit,
      m_BuckOL);
  m_EC.GetScheduler().ScheduleOnce(emergencyTime, emergencyAction);
  SimulateAndAssert("EmergencyShutdown.xml", MODEL_OK);
}
```

Abbildung 5: Integrationstest für die Notabschaltung

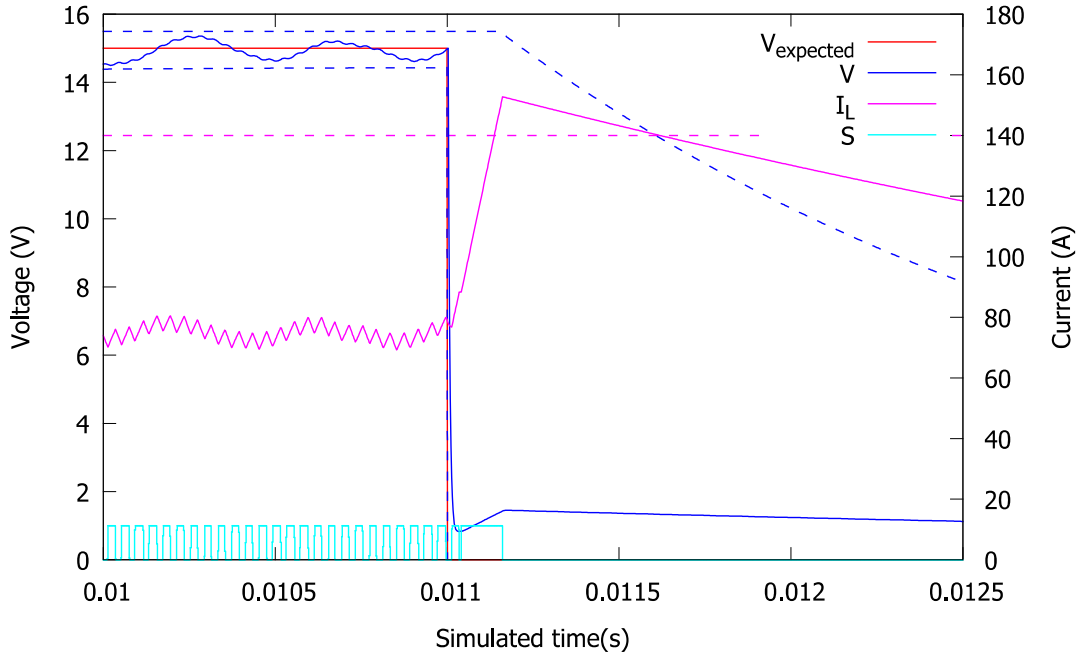

Abbildung 6: Grafisches Ergebnis des Integrationstests

Zusammenfassung

Der entwickelte Ansatz zeigt, dass eine agile und testgetriebene Methodik für komplexe Systeme mit Hardwareabhängigkeiten möglich ist. Dabei muss sichergestellt sein, dass der Entwickler zu jedem Zeitpunkt ein optimales Feedback für den Programmstand erhält. Ein komplett softwarebasierter Testansatz ist automatisierbar, minimiert die Feedbackzeit und senkt Kosten. Durch die Unabhängigkeit von realer Hardware lässt sich bereits früh entwickeln und die Time-to-Market senken.

Hardwarespezifische Details können so allerdings nur aufwendig, zeitabhängige Vorgänge innerhalb des DSP überhaupt nicht abgebildet werden. Systeme mit einem hohen Logikanteil werden dagegen von einer erhöhten Testabdeckung profitieren. Die Komplexität und der Umfang von Programmen für eingebettete Systeme steigen stetig. Gleichzeitig erlaubt die zunehmende Rechenleistung eine höhere Abstraktion auf Softwareebene. Daher ist anzunehmen, dass in Zukunft vermehrt Einsatzgebiete entstehen, die von der Anwendung einer solchen testgetriebenen Methodik profitieren.

Verweise

[1] Martin, Robert C.: Agile software development: principles, patterns, and practises. Prentice Hall PTR, 2003.

[2] Grenning, James W.: Test-driven development for embedded C. Pragmatic Bookshelf, 2011

[3] https://github.com/google/googletest

[4] http://www.freertos.org/FreeRTOS-Windows-Simulator-Emulator-for-Visual-Studio-and-Eclipse-MingW.html

[5] http://www.plexim.com/de/plecs

Autor

Daniel Penning ist freiberuflicher Softwareentwickler im Bereich eingebetteter Systeme. Mit mehr als 10 Jahren Berufserfahrung unterstützt er Unternehmen, agile Entwicklungsprinzipien (Scrum, TDD, CI) gewinnbringend auf komplexe Systeme anzuwenden. Als zertifizierter Scrum Master ist er aktuell im Bereich der Sicherheitstechnik tätig.