

Automatisierter Unit-Test von Embedded Software

Schneller und reproduzierbarer Test einzelner Softwarekomponenten

Zur Erhöhung der Softwarequalität wird von verschiedenen Prozessstandards zur Softwareentwicklung (wie etwa SPiCE) zunehmend der separate Test von einzelnen Softwarekomponenten (Units, z.B. C-Funktionen) gefordert. In der Praxis sind solche Tests nicht sehr verbreitet und werden – wenn überhaupt – ausschließlich manuell durchgeführt, was die Reproduzierbarkeit der Tests stark erschwert.

Der folgende Beitrag schildert das momentane Vorgehen in der Praxis und die damit verbundenen Probleme. Dann wird gezeigt, wie Tests automatisiert und somit schneller und vor allem reproduzierbar durchgeführt werden können. Ferner wird eine Methode zur Testfallermittlung vorgestellt.

Vorgehen in der Praxis

Der dynamische Funktionstest von Embedded-Software wird meist manuell und interaktiv durchgeführt. Dazu lädt der Entwickler das zu testende Programm in einen Simulator oder in das tatsächliche Zielsystem, etwa mit Hilfe eines JTAG-Debuggers oder eines In-Circuit-Emulators.

Manueller Test

Beim manuellen Test wird das Programm schrittweise ausgeführt und der Entwickler beobachtet, ob es sich wie erwartet verhält. Will er den Test mit anderen Werten durchführen, kann er den Debugger benutzen, um vor Aufruf der Funktion die Werte im Speicher des Zielsystems wie gewünscht zu setzen. Oft werden zusätzliche, speziell für den Test geschriebene Programmteile implementiert, um bestimmte Eingaben für die Funktion zur Verfügung zu

stellen. Manuelle Tests sind mühselig und auch fehleranfällig, da der (menschliche) Tester Fehlverhalten übersehen kann. Vor allem stehen bei manuellen Tests die Eingabewerte und das Resultat für spätere Wiederholungen der Tests meist nicht mehr zur Verfügung.

Eigene Testdatenverwaltung

Komfortable Debugger erlauben, die Eingabewerte aus einer Datei zu entnehmen. Meist lässt sich das Ergebnis der Funktion durch den Debugger mit Werten aus dieser Datei vergleichen. Aber egal wie komfortabel der Debugger ist, aufwändig ist diese Vorgehensweise in jedem Fall. Im Prinzip muss eine Datenbank mit Eingabe- und Ergebniswerten erstellt werden, wobei die Anzahl, Reihenfolge und vor allem der Typ der Werte zu berücksichtigen ist. Festzulegen ist auch der Zugriff des Debuggers auf die Datenbank, die Ermittlung des Testergebnisses, das Erstellen einer Testauswertung, die Reproduzierung von Tests etc. Dies führt am Ende zu einer eigenen Testumgebung.

Proprietäre Testumgebung

Eine solche Testumgebung wird meist speziell für die zu testende Funktion erstellt und ist wenig flexibel. Das ist nachteilig, wenn sich Anzahl, Reihenfolge oder Typ der Parameter ändern. Oft ist die Anpassung und Erweiterung der Testumgebung für eine Funktion

ebenso aufwändig wie das Ändern der Funktion selbst. Auf eine Testumgebung, welche möglichst komfortabel die Wiederholung aller Tests erlaubt, kann allerdings kaum verzichtet werden: Wird an der Implementierung etwas geändert, muss sonst erneut getestet werden – und zwar strenggenommen alle Funktionen mit allen Testfällen. Diese Wiederholungen werden Regressionstests genannt und sind mit manuellem, interaktiven Testen aus Aufwandsgründen kaum möglich.

Der aktuelle Stand

In der Praxis gibt es firmeninterne, proprietäre Testumgebungen, die für alle angesprochenen Probleme Lösungen vorsehen. Die Güte und Flexibilität dieser Testumgebungen variiert mit dem Aufwand, der für sie aufgebracht wird. Dass bei der Entwicklung von Embedded-Software keine standardisierten Lösungen verwendet werden, hat einen einfachen Grund: Erstaunlicherweise gibt es nur sehr wenige kommerziell verfügbare Werkzeuge, deren Funktionalität auf die angesprochenen Probleme zielt.

Lösung

Tessy ist ein kommerziell verfügbares Werkzeug zum automatisierten dynamischen Unit-Test von in C geschriebener Software für ein-

Autor

Dipl.-Inf. FRANK BÜCHNER ist im Bereich eingebettete Systeme Software Product Manager bei Hitex Development Tools; Greschbachstr. 12, 76229 Karlsruhe
Fon: 0721/9628-0, Fax: 0721/9628-267
e-Mail: FBuechner@hitex.de
http://www.hitex.de

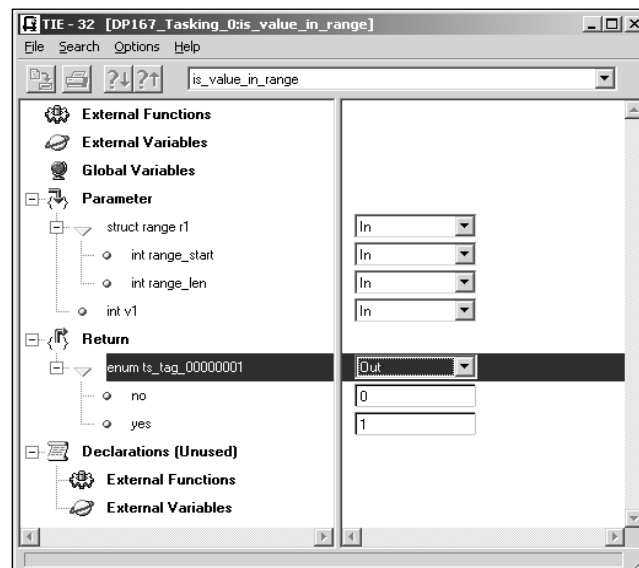


Abb. 1: Im Test Interface Editor (TIE) werden Anzahl, Typ und Passierichtung der Schnittstellenvariablen dargestellt

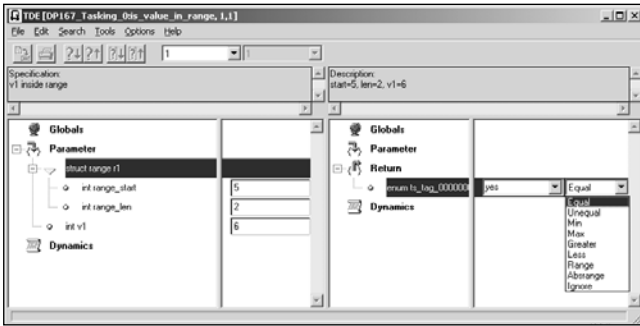


Abb. 2:
Mit dem ‚Test Data Editor‘ (TDE) werden die Ein- und Ausgabewerte der Testfälle spezifiziert

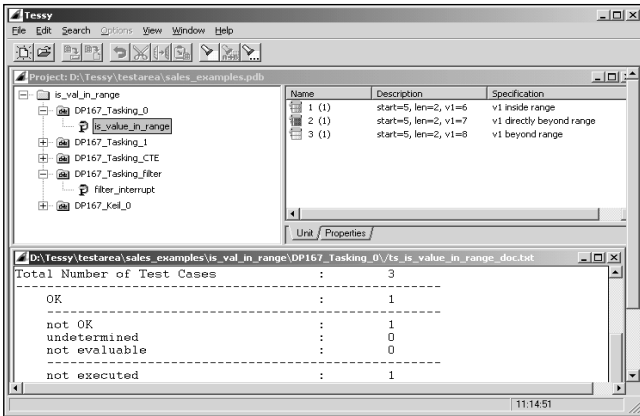


Abb. 3:
Testauswertung – Testfälle mit unerwartetem Ergebnis sind rot markiert

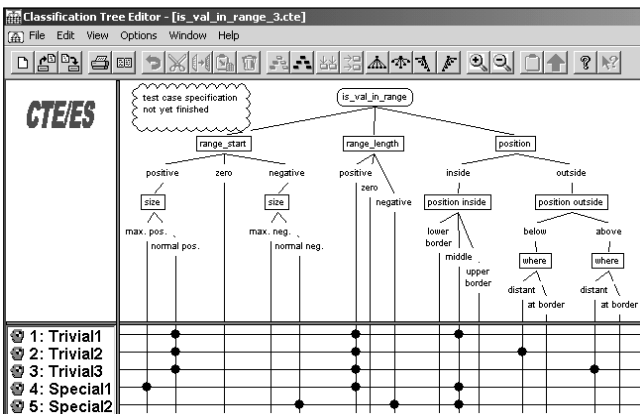


Abb. 4:
Der Klassifikationsbaueditor dient zum Zeichnen des Klassifikationsbaums und zur Testfallspezifikation

gebettete Systeme. Um den Test mit Tessy zu beginnen, wählt man einfach das Softwaremodul aus, das die zu testende Unit, in diesem Fall eine C-Funktion, enthält. Das Werkzeug analysiert dieses Modul und listet alle C-Funktionen auf, die sich in ihm befinden. Der Anwender wählt aus, welche Funktion getestet werden soll.

Schnittstellenanalyse

Tessy analysiert den Quellcode der zu testenden Funktion und bestimmt, welche Variablen der Funktionsschnittstelle Eingabe und welche Ausgabe (Ergebnis) bzw. beides sind. Anzahl, Typ und Passierrichtung (Input, Output, InOut) der Variablen der Funktionsschnittstelle wird durch den in Tessy enthaltenen Test Interface Editor (TIE) graphisch dargestellt (Abb. 1).

Das Werkzeug erstellt nun automatisch zusätzlichen Quellcode, den sogenannten Testtreiber, der die zu testende Funktion aufruft, sowie gegebenenfalls Platzhalter für Funktionen oder externe Variable, für die es noch keine Implementierung gibt. Dann erzeugt die Software aus dem Startup-Code für den betreffenden Mikrocontroller, dem Testtreiber, den optionalen Platzhaltern und der zu testenden Funktion die Testapplikation. Dabei wird üblicherweise ein Cross-Compiler für das betreffende Zielsystem eingesetzt.

Testdatenerfassung

Nun werden in Tessy die Eingabewerte und die erwarteten Resultate der Testfälle definiert, wobei der integrierte Testdateneditor (TDE) benutzt wird (Abb. 2). Der Editor stellt die Schnittstelle der zu testenden Funktion

graphisch dar und erlaubt, auch komplexe Schnittstellenelemente wie Strukturen bis auf die elementaren Datentypen aufzublättern. Die Testdateneingabe wird vom TDE besonders komfortabel unterstützt. Der Testdateneditor dient auch zur Angabe der Soll-Ergebnisse eines Testfalls. Transparent für den Benutzer werden die eingegebenen Werte in einer Datenbank abgelegt.

Testdurchführung

Zur Testdurchführung lädt die Software die Testapplikation über einen Debugger in ein Testsystem, etwa einen In-Circuit-Emulator. Der Test bezieht also den realen Mikrocontroller ein und prüft auch den verwendeten Cross-Compiler. Tessy führt alle gewünschten Testfälle auf dem Testsystem durch, wobei für jeden Testfall geprüft wird, ob das tatsächliche Ergebnis dem erwarteten Ergebnis entspricht. Die Testwerte werden jeweils aus der Datenbank entnommen und sind nicht in der Testapplikation enthalten, die Größe der Testapplikation ist also unabhängig von der Anzahl der Testfälle. Somit ist die Anzahl der Testfälle prinzipiell unbegrenzt und Tessy kann für 8-Bit-Mikrocontroller mit begrenztem Speicher eingesetzt werden.

Testauswertung

Das Werkzeug erstellt Reports unterschiedlichen Detaillierungsgrads über Durchführung und Ergebnisse der Testfälle (Abb. 3).

Fehlersuche im Testobjekt

Hat die Ausführung eines Testfalls mit einem bestimmten Satz von Eingabewerten nicht das erwartete Resultat erbracht, so muss natürlich die Ursache ermittelt werden. Die enge Integration mit Debuggern wie HiTOP von Hitex ermöglicht, den betreffenden Testfall automatisch wiederholen zu lassen, wobei Tessy einen Haltepunkt am Eintrittspunkt in die zu testende Funktion setzt. Somit führt die Testfallausführung bequem zum Übergang in den Debugger, und zwar an den Beginn der zu testenden Funktion, aufgerufen mit den Daten, die das unerwartete Resultat ergeben. Dies gestattet unmittelbar, mit der Fehlersuche zu beginnen. Ist das Problem gefunden, kann mit dem integrierten Editor die Quelle der zu testenden Funktion korrigiert werden und alle Tests lassen sich leicht mit dem geänderten Testobjekt wiederholen.

Regressionstest

Regressionstests sind Wiederholungen von bereits erfolgreich absolvierten Testfällen und

dienen dem Nachweis, dass Änderungen und Erweiterungen des Programms nicht zu unbeabsichtigten Auswirkungen führen. Automatische Regressionstests sind besonders nützlich nach Optimierung der Software oder bei Verwendung einer neuen Compiler-Version. Tessy enthält eine Batch-Funktion zur Durchführung von umfangreichen Regressionstests ohne Interaktion durch den Benutzer.

Testfälle anpassen

Durch Weiterentwicklung des Anwendungsprogramms kann es vorkommen, dass sich Testfälle nicht mehr ausführen lassen, weil sich die Schnittstelle der zu testenden Funktion geändert hat, etwa durch einen zusätzlichen Parameter. Bei eigenerstellten Testumgebungen ist dann meist hoher Anpassungsaufwand nötig, Tessy jedoch erledigt die Anpassung komfortabel und weitgehend automatisch.

Testdaten weiterverwenden

Die komfortable, möglichst umfangreiche Weiterverwendbarkeit von existierenden Testdaten auch bei geänderter Schnittstelle ist Voraussetzung, dass sich Testfälle aktualisieren lassen und somit jederzeit wiederholbar sind. Testwiederholungen bedeuten Regressionstests, und die wiederum sind ein wesentliches Mittel zur Sicherung von Softwarequalität. Erst mit Regressionstests wird es möglich, Software im Hinblick auf Code-Größe, Performance, Wartungsfreundlichkeit und Wiederverwendbarkeit zu optimieren, weil Regressionstests sicherstellen, dass die eigentliche Funktionalität unverändert bleibt.

Testabdeckungsanalyse

Die Testabdeckungsanalyse gibt an, in welchem Umfang die durchgeführten Tests die zu testende Software erfasst haben. Wird hier eine bestimmte Prozentzahl nicht erreicht, waren die Tests nicht ausreichend oder es liegt eine sonstige Abnormalität vor, wie etwa ‚toter Code‘.

Mit Tessy können die Pfade ermittelt werden, die während der Tests durchlaufen wurden. Tessy setzt die Zahl der durchlaufenen Zweige in Beziehung zur Zahl aller möglichen Zweige, dies ergibt dann die Zweigabdeckung (branch coverage), auch C1-Coverage genannt. Tessy gibt an, bei welchem Testfall welcher Pfad durchlaufen wurde: Ist die Zweigabdeckung nicht ausreichend, kann man leicht erkennen, welche Zweige nicht getestet wurden.

Testplanung mit CTE

Der ‚Classification Tree Editor‘ (CTE) ist Bestandteil der Testsoftware und wird zur Testfallspezifikation anhand der Klassifikationsbaummethode eingesetzt (Abb. 4). Die Klassifikationsbaummethode ist ein systematischer Weg zur Ermittlung einer Menge von redundanzarmen, fehlersensitiven Testfällen und führt von der Problemspezifikation zu Testfallspezifikationen. Dabei wird der sogenannte Klassifikationsbaum erstellt.

Die Anwendung der Klassifikationsbaummethode fördert das Nachdenken über Spezifikation und Testfälle. Dabei gewinnt man Klarheit über Fragen wie:

- ▶ Welche Aspekte gibt es für den Test und wie werden sie durch Testfälle abgedeckt?
- ▶ Welche Menge von Testfällen ist ausreichend?
- ▶ Gibt es redundante Testfälle?
- ▶ Wie viel Testaufwand ist zu erwarten?

Der CTE ist ein syntaxgesteuerter graphischer Editor für die Erstellung des Klassifikationsbaums und der darauf basierenden Ermittlung von Testfallspezifikationen. Der CTE verwaltet Testfallspezifikationen, Testfälle, Testfallanmerkungen etc. und kann diese Information auch anderen Werkzeugen zur Verfügung stellen.

Fazit

Tessy erleichtert also wesentlich das Testen von Programmen und ersetzt weitestgehend proprietäre Testumgebungen. Zu Testbeginn erzeugt Tessy noch fehlende Komponenten wie Testtreiber oder Platzhalterfunktionen. Durch die Testdatenverwaltung bleiben alle Testdaten für spätere Tests erhalten. Die Integration mit Debuggern erlaubt effiziente Fehlersuche bei unerwarteten Ausgabewerten. Bei der Weiterentwicklung des Programms werden auch die Tests weiterentwickelt und angepasst. Somit ist das Werkzeug besonders für Regressionstest geeignet.

Die Testfallplanung zu Beginn der Entwicklung korrespondiert mit der Testabdeckungsanalyse beim Abschluss der Tests. **TEST**