ESE Kongress 2011

Vortragsskript:

Die Rolle des Debuggers im Test-Prozess

Ingo Pohle, MicroConsult

Bestandteil jeder Entwicklung ist der Test. Je nach Anforderungen an das Ausgangsprodukt werden unterschiedliche Testansätze verwendet:

- bei hohen Anforderungen an Funktionalität und Qualität: aufwendigere Tests und Dokumentation sind erforderlich; Tests werden mit speziellen Test-Tools durchgeführt
- bei niedrige oder keine Anforderungen an die Sicherheit: Tests werden meist mit einem Debugger durchgeführt

Der Debugger wird dabei für das Auffinden und Beheben von Fehlern (Bugs) eingesetzt. Dabei können folgende Debugger-Basisfunktionalitäten verwendet werden:

- Programm in die Zielhardware laden (Typ. Programmieren in Flash-Speicher)
- Definition von Break-Points
- Programmablauf starten
- Beobachten von Systemzuständen (Beobachtung des laufenden Systems)
- Anzeigen und ändern von:
  CPU- Registern
  Dateninhalten aus Datenspeichern
- Zyklisches Auslesen von (Peripherie-)Registern bei minimaler Beeinflussung des laufenden Systems durch Watch Windows

Welche Aufgaben umfasst ein Testprozess für qualitativ hochwertige und funktionierende Embedded SW?

Für die Entwicklung qualitativ hochwertiger Embedded Software sind eine Reihe von Normen definiert. Diese betrachten alle Phasen der Entwicklung als einen gesamten Prozess, der unter anderem folgende Schritte der Softwareentwicklung definiert:

- Planung
- Entwicklung
- Test
- Inbetriebnahme

Durch Prozessmodelle wird die Reihenfolge dieser Arbeitsschritte genau beschrieben. Häufig wird bei der Entwicklung das V-Modell verwendet. Folgende Darstellung zeigt die eine Vorgehensweise, die dem V-Modell angelehnt ist:

Die einzelnen Aufgaben bei der Entwicklung werden in Normen detailliert beschrieben, Beispiele hierfür sind:

- **IEC 61508** allgemein gültige Norm für sichere Software
- **ISO 26262** beschreibt die funktionale Sicherheit im Automobil-Bereich
- **IEEE829** Test-Norm definiert die Dokumentation für die Testplanung und -ausführung.

Die Normen sollen dazu führen, dass Produkte so hergestellt werden, dass sie möglichst immer fehlerfrei funktionieren, oder auftretende Fehler diagnostiziert werden und geeignet darauf reagiert wird.

Die Normen beschreiben die Anforderungen an Maßnahmen zur:

- Fehlervermeidung
- Fehlerbeherrschung
- Dokumentation der erforderlichen Arbeitsschritte und Maßnahmen

Wird bei der Softwareentwicklung die Erreichung einer Sicherheitsanforderungsstufe SIL1 – SIL4 (Safety Integrity Level SIL) gefordert, müssen die einzelnen Arbeitsschritte strikt nach den entsprechenden Normen erfolgen. Die Stufen SIL1 – SIL4 beschreiben, welche Maßnahmen bei der Entwicklung getroffen wurden, die eine risikomindernde Wirksamkeit bei den sicherheitsrelevanten Funktionen erreichen.

Die Anforderungen für die Entwicklung von qualitativ hochwertiger Software sollte die Basis für alle Software-Entwicklungen sein.

Die Erhöhung der Qualität sollte Ziel jeder Software-Entwicklung sein.

Aufgaben, die der Debugger im Unit-Test übernehmen kann

Aufgaben, die durch einen Debugger im Unit-Testprozess für „sichere“ Software wahrgenommen bzw. abgedeckt werden können und wo der Einsatz von speziellen Testwerkzeugen notwendig ist:

Qualitätsmerkmal	**Qualitäts-Teilmerkmale**	**Testmethode**	**Testwerkzeug**
Funktionalität	Angemessenheit Richtigkeit **Sicherheit** Interoperabilität Ordnungsmäßigkeit/ Normgerechtigkeit	Black-Box Test	Typ.: **Unit Test Tool** bedingt **Debugger** mit User def. Test-Treiber
		White Box Test (Code Coverage)	**Debugger**
Zuverlässigkeit	Fehlertoleranz, Reife Wiederherstellbarkeit: Fehlertoleranz/Robustheit	Back Box Test: Verwendung ungültiger Parameter	Typ.: **Unit Test Tool** bedingt **Debugger** mit User def. Test-Treiber
Benutzbarkeit	Erlernbarkeit Bedienbarkeit Verständlichkeit	Benutzbarkeits-Test	Anwender, produktspez. Testwerkzeuge
Effizienz	Verbrauchsverhalten Zeitverhalten	Ressourcen- Verbrauchsbestimmung	Linker/Locator **Debugger:** **Cycle Measurement**
Änderbarkeit	Analysierbarkeit, Prüfbarkeit Modifizierbarkeit, Stabilität	Dokumenten-Review (Modularität der Architektur)	Manuelles Review
Übertragbarkeit	**Modularität**: Anpassbarkeit Installierbarkeit Konformität Austauschbarkeit	Dokumenten-Review (Architektur/ Schichtenmodell)	Manuelles Review

Hohe Qualität – Ziel jeder Entwicklung

Mit der Erhöhung der Qualität bei der Projektentwicklung können Verzögerungen bei der Entwicklung gesenkt, Ausfälle und damit nachträgliche Kosten reduziert werden. Ziel jeder Entwicklung sollte daher sein, Qualitätskriterien beim Design, der Umsetzung und beim Test zu berücksichtigen. Da sich einige der Qualitätskriterien widersprechen (z. B. Effizienz versus Zuverlässigkeit), sollten maximal bis zu 4 Qualitätsmerkmale angestrebt werden.

Neben der Hauptaufgabe des Debuggers – der Fehlersuche und Behebung – kann nun mit diesem Werkzeug zum Teil das Erreichen der gewünschten Qualitätsmerkmale geprüft werden:

Qualitätsmerkmal	Methode	Tool
Funktionalität	Black Box Test	typ. Unit Test Tool, zum Teil der Debugger
	White Box Test	Unit Test Tool, alternativ der Debugger
Zuverlässigkeit	Black Box Test	typ. Unit Test Tool, bzw. bedingt der Debugger
Effizienz	Ressourcen- Verbrauchsbestimmung	Linker/Locator und Debugger

Aufgaben des Tests

Ein Test unterscheidet sich vom Debuggen dadurch, dass die Vorgehensweise geplant ist. Folgende Aufgaben umfassen den „geplanten“ Test:

Testplanung	TEST- MANAGEMENT
Testdesign	
Testdurchführung	
Testdokumentation	
Testauswertung	

Testplanung
Erstellung der Planung für den Test

Testdesign: Definition der Testfälle
Es werden die Testfälle aus der Spezifikation abgeleitet und gewichtet:
muss-Kriterien: müssen zu 100% erreicht werden
soll-Kriterien: sollten zu 50% erreicht werden
kann-Kriterien: Erfüllung offen

Hierfür lassen sich einsetzen:
Unit-Test Tools; Excel/Word-Dokumente

Hinweis: Diese Methode bedeutet Aufwand, der Grad der Schlüssigkeit des Tests muss selber sicher gestellt werden.

Testdurchführung:
Hierfür wird ein Unit-Testtool benötigt, alternativ eignet sich auch zum Teil ein Debugger

Dokumentation und Testauswertung
Auswertung der Testergebnisse: Passed/Failed
Die Produktfreigabe hängt vom Ergebnis der Auswertung ab.
Hinweis: Alle Testschritte werden durch einen Testmanager begleitet und überwacht.

Mögliche Features des Debuggers für den Unit-Test

Features des Debuggers für die Prüfung der Funktionalität von Funktionen aus C/C++ Modulen (Units):

Automatisierte Unit Black Box Tests mit dem Debugger
Durch den direkten Zugriff des Debuggers können Tests direkt mit der Zielhardware durchgeführt werden.

Vorteile: Zeitersparnis beim Test, Wiederholbarkeit (Regressionstests), Gewissenhaftigkeit (alle vorgegebenen Parameter werden automatisch für den Test verwendet)

Makrotechnik im Debugger

Makrodefinition mit einer Script-Sprache oder proprietäre Makro-Sprache, Input-Daten werden aus einer Parameter-Datei gelesen, das Schreibe der Ausgabewerte erfolgt in eine EXCEL Datei, bzw. XML.

Nachteil: Es wird keine automatische Dokumentation des Tests unterstützt, sie muss gegebenenfalls selber erstellt werden.

Kopplung des Debugger an ein Testwerkzeug
erfolgt über eine offene COM-Schnittstelle (z. B. PLS-Debugger UAD2 und Test-Tool Tessy von Razorcat).

Voraussetzung: Der Debugger muss eine COM-Schnittstelle unterstützen.

Vorteile: Diese Kopplung verkürzt die Testzeiten durch den schnellen Target-Zugang des Debuggers. Der Test wird direkt auf der Zielhardware durchgeführt.

Die Programmierung des Debuggers und des Testwerkzeuges kann mit der Programmiersprache C++, C# oder mit einer Standard Script-Sprache erfolgen: Java Script, Phyton, Visual Basic Script, Windows Power Shell etc.

Test-Voraussetzung: Kenntnis der standardisierten Script-Sprache und des Testwerkzeuges.

Mögliche Features des Debuggers für den System-Test

Features des Debuggers für die Prüfung der Systemfunktionalität: Der Systemtest erfolgt unter Echtzeitbedingungen, d. h. das Programm läuft unter realen Bedingungen ab.

Folgende Methoden können für eine Software Analyse verwendet werden:

- **Test auf Betriebssystemebene (OS-aware Debugging)**

Der Test auf Betriebssystemebene wird heute durch die meisten Debugger unterstützt. Der Test kann auf der Task-Ebene unter Berücksichtigung der OS-Mechanismen durchgeführt werden.
Hinweis: Debugger unterstützen jeweils nur spezielle Auswahl von Betriebssystemen.

- **Performance Analyse**

Messung bei Funktionen (Profiling):
der Verarbeitungsgeschwindigkeit
der Speichernutzung
der generierten CPU-Last von Funktionen/Tasks

Ergebnis: Verifikation der Performance von Zuweisungen von Ressourcen

Beispiele für die Zuweisungen performanter Ressourcen: Cache Memory, Level 1 RAM, Interrupt Service Routinen bei hochfrequenten Abläufen, etc.

- **Speicher/Stack Analyse**

Untersuchung z. B. der verwendeten Stack-Tiefe und der Funktions-Verschachtelung (einige Debugger können Funktionsverschachtelungen visualisieren: Call Stack Graph). Speicherinhalte können gemäß ihres Datentyps visualisiert werden.

Ziel: Prüfung ob die Stack-Reservierung für die Applikation ausreicht

- **Programmfluss Analyse**

Ziel: Untersuchung des Programm-Kontrollflusses, mit dem Ergebnis der Identifikation nicht ausgeführter Anweisungen (toter Code). Methode: Code Coverage:

Statement Coverage

Untersuchung zeigt, ob alle Statements ausgeführt werden.
Ergebnis: Kennzeichnung/Markierung der ausgeführten Statements.

Function Coverage

Untersuchung, ob alle Funktionen/Tasks bearbeitet werden.

Ergebnis: Kennzeichnung/Markierung der ausgeführten Funktionen.

Hinweis: Ergebnisse der Code Coverage Analyse können für einen Nachweis archiviert werden (Beispiel: Export in XML-Datei).

Decition/Condition/Branch Coverage wird von Debuggern noch nicht unterstützt, diese Tests müssen mit einem Unit Test Tool durchgeführt werden.

Decition/Condition/Branch Coverage untersucht, ob alle Zweige von Bedingungen bearbeitet werden.

Ergebnis: Kennzeichnung/Markierung der ausgeführten Pfade.

- **Trace-Aufzeichnungen**

Bei Trace-Aufzeichnungen wird unterschieden in Program-Trace (Aufzeichnung von ausgeführten Befehlen) und Daten-Trace (Aufzeichnung von Variableninhalten).

Voraussetzungen: Trace I/O-Interfaces (für Programm-, bzw. Daten-Trace) am Mikrocontroller

Beispiel: Cortex-M3 Baustein mit Embedded Trace Makrozelle und Trace-Pins, On-Chip Trace Support (spezielle Trace-Hardware und SRAM).

Alternativ: Spezieller ASIC-Emulationsbaustein (wird nur von wenigen Debugger-Herstellern für spezielle Mikrocontroller angeboten).

- **Graphische Darstellung von Variablen und Ausdrücken (Logic Analyzer)**

Der Verlauf von Variableninhalten, bzw. Ausdrücken kann graphisch visualisiert oder in Dateien exportiert werden.

Hinweis: Die Debugger sampeln die Werte mit einstellbaren Perioden (z. B. Sample-Periode: 1 ms). Reale/vollumfängliche Werteverläufe können nur aufgezeichnet werden, wenn die Mikrocontroller über ein Daten-Trace-Interface verfügen.

- **Flash Programmierung**

Der Flash Programmiersupport wird verwendet für:
Flash Löschen: ausgewählter Flash-Segmente, bzw. gesamtes Flash; Flash Programmierung;
Flash Verifikation (Prüflesen, ob korrekt programmiert wurde)

- **MMU-Support**

Einige Mikrocontroller-Architekturen verwenden Memory Management Units (MMUs) für den Zugriff auf virtuelle Memory Bereiche. Für die Analyse der MMU-Einträge ist eine Erweiterung des Debugger erfordert.

- **Context-Analyse**

Einige Mikrocontroller-Architekturen verwenden nicht die klassische Methode der Abspeicherung von Rückkehr-Adresse bei Funktionsaufrufen, bzw. Rückkehr-Adresse plus CPU-Status bei der Annahme von Interrupt-/Exception-Routine im Stack, sondern eine verlinkte Context-Architektur. Diese Methode hat Vorzüge bei der Verwendung von Betriebssystemen. Daher werden Erweiterungen für die Analyse der Context-Einträge beim Debugging benötigt (Visualisierung der verlinkten Einträge).

- **Multicore-Debugging**

Das Multicore-Debugging wird nur bei speziellen Multicore-Mikrocontrollerarchitekturen benötigt und setzt spezielle Erweiterungen im Debugger voraus.

- **Core-Simulatoren**

Core-Simulatoren stehen nur für spezielle Mikrocontroller Architekturen zur Verfügung und werden nur von einigen Debugger Herstellern angeboten. Besondere Merkmale zeigen hier Debugger, die neben der Simulation des Programmes (Codes) auch eine Takt-genaue Simulation der On-Chip Peripherie anbieten (Beispiel: Keil-Debugger für ARM Architekturen).

- **Protokoll-Analyse**

Protokolle wie z. B. LIN, CAN, SPI, FlexRay, etc. können von einigen sehr hochwertigen Debuggern interpretiert werden.