Code Coverage auf Embedded Devices

Testabdeckung bei kleinen Targets

Auch bei Embedded-Geräten muss die Software hinreichend getestet werden, bevor das Produkt marktreif ist. Die Überwachung des Testfortschritts, die von vielen Sicherheitsnormen gefordert wird, ist hier jedoch nicht ganz trivial. Denn oft sind die Ressourcen der Targets limitiert, Speicher und Prozessor können durch die Code Coverage an ihre Grenzen stoßen. Doch mit etwas Kreativität ist es möglich, auch im Embedded-Bereich die Testabdeckung zu erfassen und nachzuweisen.

Von Klaus Lambertz, Geschäftsführer der Verifysoft Technology GmbH

Die Anforderungen an Embedded-Devices steigen. Vor allem unter dem Schlagwort IoT (Internet of Things) erobern sich Embedded-Systeme eine wichtige, wenn nicht sogar kritische Position in den Geschäftsprozessen der Unternehmen. Das Marktforschungsunternehmen IDC erwartet laut einer Studie vom Juni 2018, dass der Markt für IoT-Technologien bis zum Jahr 2020 bei einer jährlichen Wachstumsrate von über 13 Prozent weltweit auf ein Volumen von 1,2 Billionen Dollar steigen wird. Embedded-Geräte werden also für alle Branchen kritisch für den Geschäftsbetrieb, die Frage nach der Sicherheit wird drängend. Und in anderen Bereichen ist die Sicherheit von Embedded-Devices heute schon ein zentraler Aspekt bei der Entwicklung.

In Medizintechnik, Luftfahrt, Straßen- und Schienenverkehr regeln strenge Normen, wie Embedded-Geräte getestet und zertifiziert werden müssen, um überhaupt zum Einsatz zu kommen. Außerhalb dieser traditionell auf Sicherheit fokussierten Branchen besteht offensichtlich Nachholbedarf. Laut einer Umfrage von Gartner vom Frühjahr 2018 wurden fast 20 Prozent der Unternehmen in den vergangenen drei Jahren bereits Opfer eines IoT-basierenden Angriffs. Einer der Gründe dafür ist laut Gartner, dass sich die Standards für IoT-Security erst langsam entwickeln. Es mangele noch an Security by Design.

Code Coverage ist oft zwingend

Für sichere Embedded-Geräte ist das Testing ein wichtiger Baustein. Nicht ohne Grund stellen Normen für die sicherheitskritische Softwareentwicklung genaue Anforderungen an die Testmethoden und die Testabdeckung. In der Regel setzen einschlägige Standards wie DO-178C in der Luftfahrt oder ISO 26262 im Automotive-Bereich Anweisungs- und Zweigüberdeckungstests voraus. Auch modifizierte Bedienungs- und Entscheidungsüberdeckungstests (MC/DC, Modified Condition/Decision Coverage) sind häufig üblich. Grundsätzlich gilt: Je höher die Sicherheitsanforderungen an eine Software, desto höher muss die geforderte Testabdeckung sein. Nicht alle Code-Coverage-Stufen sind in jedem Szenario sinnvoll, zudem bringen sie unterschiedliche Erkenntnisse:

- Die Function Coverage (Aufrufüberdeckung) ignoriert die inneren Abläufe der Software komplett, ihr Nutzen ist also relativ gering.
- Bei der Statement Coverage (Anweisungsüberdeckung) wird ermittelt, welche Anweisungen durch die Tests ausgeführt wurden. Hier lassen sich unter anderem toter Code erkennen und

Anweisungen, für die noch keine Tests vorliegen.

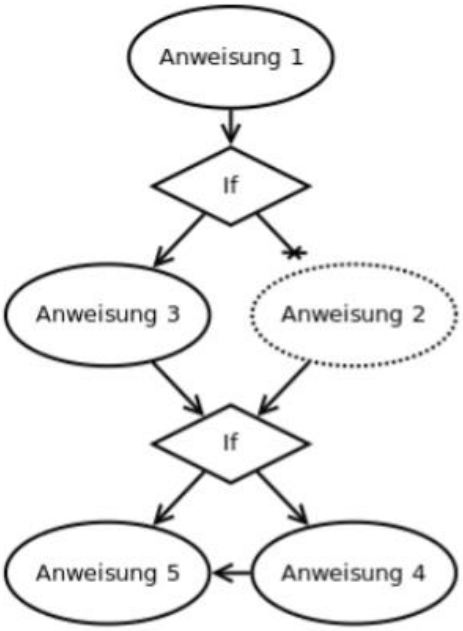

Abb. 1: Die Statement Coverage gibt an, wie viele Anweisungen ausgeführt wurden. In diesem Beispiel sind es sechs von sieben, die Test-Abdeckung beträgt also 85 Prozent.

- Die Branch Coverage (Zweigüberdeckung) ermittelt, ob alle Programmzweige durchlaufen wurden. Damit ist Branch Coverage ausführlicher als die Statement Coverage und eine mit vertretbarem Aufwand realisierbare Mindestanforderung für das Testing.
- MC/DC (Modified Condition/Decision Coverage) ist die höchste in den Normen geforderte Testabdeckungsstufe. Da die Überprüfung aller Bedingungskombinationen einen enormen Aufwand erfordern würde, versucht MC/DC diesen Testaufwand zu minimieren. Hierbei werden alle atomaren Bedingungen einer zusammengesetzten Bedingung herangezogen. Für jede der atomaren Bedingungen wird ein Testfallpaar getestet, welches zur Veränderung des Gesamtergebnisses der zusammengesetzten Bedingung führt, wobei sich jedoch nur der Wahrheitswert der betrachteten atomaren Bedingung ändert. Hierbei muss der Wahrheitswert der anderen atomaren Bedingungen konstant bleiben.

Um die Testabdeckung zu ermitteln, werden Code Coverage Analyser eingesetzt. Dazu ergänzen sie den Code vor der Übergabe an den Compiler mit Zählern für die gewünschten Testebenen. Diesen Vorgang bezeichnet man als das Instrumentieren des Codes. Ein sehr einfaches Werkzeug dafür ist zum Beispiel das GNU Coverage Testing Tool `gcov`. Mittels der gcc-Option `-ftest-coverage` kann der Code instrumentiert werden, um zu ermitteln, wie oft jede Code-Zeile ausgeführt wurde. Die Option `-fprofile-arcs` instrumentiert die Verzweigungen. Um erste Erfahrungen auf dem Gebiet der Code Coverage zu machen oder für kleine Projekte ist dieses Tool auf jeden Fall geeignet. Sein Nachteil ist – neben dem Fehlen von anspruchsvollen Testabdeckungsstufen, die für größere Testing-Anforderungen benötigt werden - die Aufbereitung und Interpretation der gewonnenen Informationen. Hier sind kommerzielle Tools deutlich überlegen.

CTC++ Coverage Report - Functions Summary #1/1

Directory Summary | Files Summary | Functions Summary | Untested Code | Execution Profile
To directories: First | Previous | Next | Last | Index | No Index

Directory: C:\Projects\hcontrol
TER: 71 % (44/62) structural, 78 % (49/63) statement

Source file: C:\Projects\hcontrol\regulators.c
Instrumentation mode: multicondition **Reduced to:** MC/DC coverage
TER: 71 % (20/28) structural, 71 % (17/24) statement
To files: Previous | Next

TER % - MC/DC		TER % - statement		Calls	Line	Function
75 % -	(6/8)	83 % -	(5/6)	3	4	lights()
100 %	(2/2)	100 %	(1/1)	1	20	close_windows()
100 %	(2/2)	100 %	(1/1)	2	25	open_windows()
100 %	(2/2)	100 %	(3/3)	1	30	open_windows_for()
100 %	(2/2)	100 %	(1/1)	1	37	heat()
0 % -	(0/2)	0 % -	(0/1)	0	42	air_condition()
60 % -	(6/10)	55 % -	(6/11)	2	47	temperature_control()
71 % -	(20/28)	**71 %** -	(17/24)			**regulators.c**

Abb. 2: Summary Coverage Report des Tools Testwell CTC++.
Für die Funktion lights wurden im Rahmen des Tests 5 von 6 Statements ausfeführt – entsprechend ist die Testabdeckung (TER = Test Effectivness Ratiio) hier 83%. Die strukturelle Testabdeckung, im Beispiel MC/DC, beträgt 75%.

Hits/True	False	Line	Source
		1	/* File io.c ----------
		2	#include <stdio.h>
		3	#include "io.h"
		4	/* Prompt for an unsigend int value and return it */
Top			
10		5	unsigned io_ask()
		6	{
		7	unsigned val;
		8	int amount;
		9	
		10	printf("Enter a number (0 for stop program): ");
0	10	11	if ((amount = scanf("%u", &val)) <= 0) {
		12	val = 0; /* on 'non sense' input force 0 */
		13	}
10		14	return val;
		15	}

Abb 3: Testwell CTC++ Report mit Coverage-Informationen im Quellcode.
Die Bedingung in Zeile 11 wurde 10 mal als „falsch“ ausgewertet und nie als „wahr“. Zur vollständigen Testabdeckung wäre dies noch nachzuholen.

Instrumentierung erweitert den Code

Die professionellen Lösungen arbeiten nach dem gleichen Grundprinzip wie gcov und instrumentieren den Code. Die Zähler werden in der Regel als globale Arrays abgelegt. Wann und wie diese Zähler dann verändert werden, hängt von der geforderten Code-Coverage-Stufe ab. Das folgende Beispiel einer in C geschriebenen While-Schleife macht deutlich, welche Konsequenzen die unumgängliche Instrumentierung hat:

```
while (! b == 0 )
{
   r = a % b;
   a = b;
   b = r;
}
result = a;
```

Durch die Instrumentierung – in diesem Fall mit dem Code-Coverage-Werkzeug Testwell CTC++ - ergibt sich folgende Struktur:

```
while ( (( ! b == 0 ) ? (ctc_t[23]++, 1) : (ctc_f[23]++, 0)) )
{
  r = a % b ;
  a = b ;
  b = r ;
}
result = a ;
```

Knappe Ressourcen

Durch die Instrumentierung wächst der Code. Die benötigten Arrays befinden sich im Datenspeicher, sowohl im RAM als auch im ROM sind zusätzliche Kapazitäten notwendig. Auch beeinflusst die Instrumentierung die Ausführungszeit. Bei Server- oder PC-Anwendungen kann dieser Effekt vernachlässigt werden. Bei Embedded-Geräten hingegen oft nicht, da die Hardware-Ressourcen aus Kostengründen oft sehr knapp kalkuliert sind. Hier ist darauf zu achten, einen Code Coverage Analyser mit einem vergleichsweise geringen Instrumentierungs-Overhead zu nutzen, da die Zähler sonst schnell die Grenzen des verfügbaren Speichers sprengen. Das gilt insbesondere, wenn sehr anspruchsvolle Testabdeckungsstufen wie MC/DC erforderlich sind.

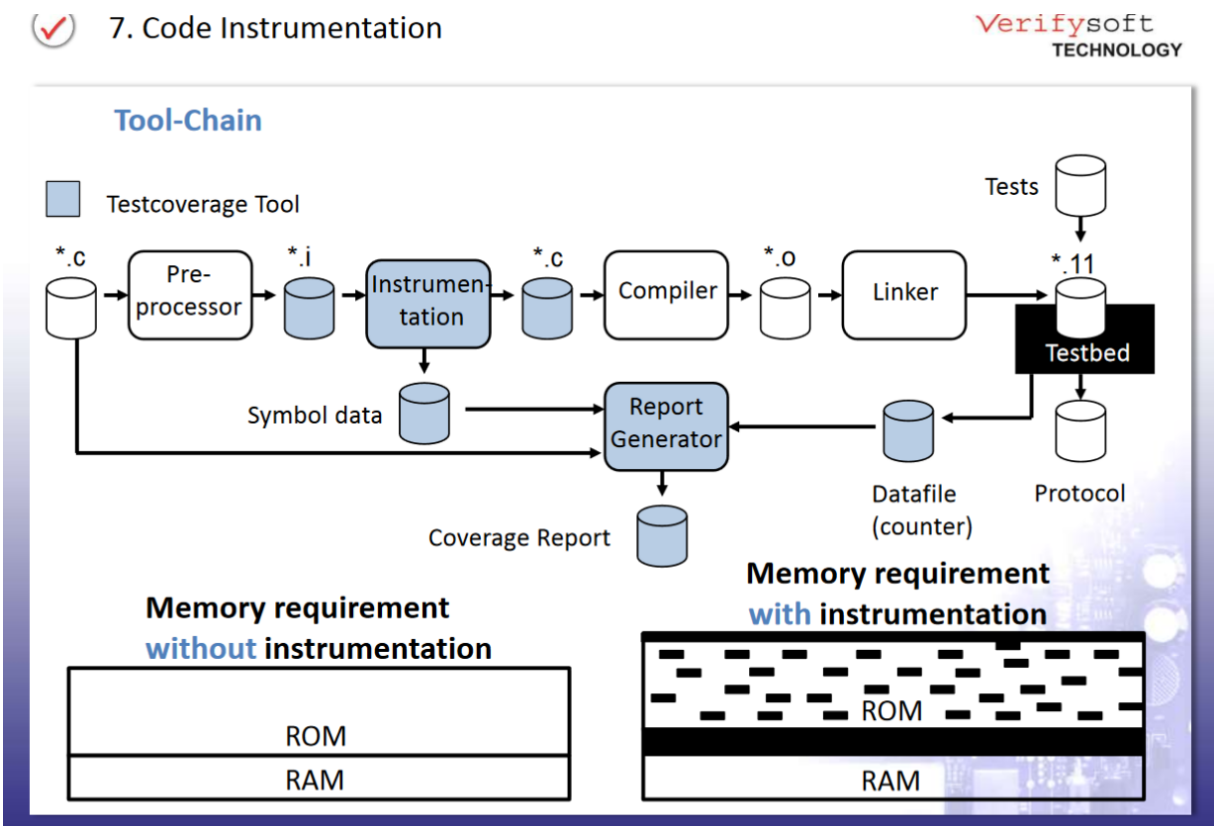

Abb. 4: Die Instrumentierung des Codes erhöht den Ressourcen-Bedarf spürbar.

Sollte das Code-Coverage-Tool einen zu hohen Instrumentation-Overhead haben, kann diese Hürde beim RAM mit der partiellen Instrumentierung umgangen werden. Dabei werden nur kleine Ausschnitte des zu testenden Programms instrumentiert und getestet. Der Test wird nacheinander mit allen Programmteilen wiederholt, die daraus gewonnenen Daten werden zu einem Gesamtbild zusammengefügt. Dadurch kann die Testabdeckung für das vollständige Programm ermittelt werden. Ein anderer Ansatz auf kleinen Targets ist, die Größe der Zähler zu beschränken. Normalerweise arbeiten Code-Coverage-Werkzeuge mit 32-Bit-Zählern. Diese können zumindest theoretisch auf 16 oder 8 Bit reduziert werden. Hierbei sollte man aber Vorsicht walten lassen, denn unter Umständen können die Zähler dann überlaufen. Die gewonnenen Daten müssen also mit großer Sorgfalt interpretiert werden. In extremen Fällen können die Zähler zudem auf einzelne Bits gesenkt werden. Diese Bit-Coverage kann zum Beispiel dann sinnvoll sein, wenn es nicht relevant ist, wie oft ein Programmabschnitt durchlaufen wurde.

ROM-Bedarf	
Ohne Instrumentierung	60 Bytes
Coveragestufe Funktionsüberdeckung	67 Bytes
Coveragestufe Zweigüberdeckung	118 Bytes
Coveragestufe Bedingungsüberdeckung	285 Bytes
Zusätzlicher RAM-Bedarf ohne Bit-Coverage (32-Bit-Zähler)	
Coveragestufe Funktionsüberdeckung	4 Bytes
Coveragestufe Zweigüberdeckung	16 Bytes
Coveragestufe Bedingungsüberdeckung	28 Bytes
Zusätzlicher RAM-Bedarf mit Bit-Coverage	
Coveragestufe Funktionsüberdeckung	1 Bit
Coveragestufe Zweigüberdeckung	4 Bit
Coveragestufe Bedingungsüberdeckung	7 Bit

Abb. 5: Durch die von Testwell CTC ++ angebotene Bit-Coverage lässt sich der RAM-Bedarf signifikant reduzieren.

Auch die gewählte Coverage-Stufe beeinflusst die Anforderungen an den verfügbaren RAM. Der zusätzlich benötigte Platz im ROM hingegen lässt sich kaum begrenzen. Zur Erfassung der Code Coverage ist eine kleine Bibliothek erforderlich, die unter anderem für die Übertragung der Zählerstände an einen Host zuständig ist. Neben dem Speicher belastet die Instrumentierung auch den Prozessor im Target. Hierdurch kann es vorkommen, dass ein definiertes Timing nicht mehr eingehalten wird. Besonders wenn die CPU bereits eng am Limit arbeitet, können fehlerhafte Abläufe auftreten. Die Buskommunikation ist dafür besonders anfällig. Hier sollte der Tester aufmerksam den Ablauf überwachen und die Ergebnisse sorgfältig prüfen. Leistungsfähige Code Coverage Tools sind jedoch in der Lage den Speicherbedarf für die Instrumentierung sowie die Änderungen des Laufzeitverhaltens relativ gering zu halten.

Fazit

Testing und die Ermittlung der Testabdeckung werden im Embedded-Umfeld deutlich an Bedeutung gewinnen. Denn die Software-Qualität wird immer wichtiger. Auch wenn sicher nicht alle Normen und Standards langfristig eine 100-prozentige MC/DC-Coverage für jede Art von Software zu verlangen: Es ist nur eine Frage der Zeit, bis die Standardisierungsgremien und Branchenverbände die Anforderungen auch abseits der sicherheitskritischen Anwendungen erhöhen. Bessere Tests sind aber auch im Interesse der Hersteller selbst. Denn fehlerhafte Produkte verursachen hohe Folgekosten, vom eigenen Ruf ganz abgesehen. Die vom PC bekannte Bananen-Software, die erst beim Nutzer reift, werden die Kunden im Embedded-Bereich kaum akzeptieren wollen.

09/2018