Automatisierte Ausführungsemulation von Testprogrammen für Sensorsysteme

Franziska Mayer, Christian Schott, Erik Markert und Ulrich Heinkel
Professur für Schaltkreis- und Systementwurf
Technische Universität Chemnitz
Chemnitz, Deutschland
{franziska.mayer, christian.schott, erik.markert, ulrich.heinkel}@etit.tu-chemnitz.de

***Zusammenfassung*—Es wird eine Methodik zur Testprogrammverifikation und deren automatisierte Ausführung für ein namhaftes, industriell eingesetztes Testsystem vorgestellt. Der Ansatz basiert auf der Spezifikation von Bauteilvarianten. Die Bauteilantworten einer Variante werden während der Ausführung emuliert und ersetzen Messwerte physischer Bauteile durch das Testsystem. Die Generierung und Emulation der Bauteilvarianten erfolgen automatisiert. Es wird ausschließlich die Softwareumgebung des Testsystems benötigt. Weder physische Bauteile noch Online-Testzeit auf dem Testsystem sind notwendig.**

***Indexbegriffe*—Schaltkreistest, Mutationstest, Testprogrammverifikation, Ausführungsemulation**

I. Einleitung und verwandte Themen

Der Test von Integrierten Schaltkreisen ist essenzieller Bestandteil des Fabrikationsprozesses von der Spezifikationsphase bis zur Auslieferung. In dieser Veröffentlichung wird dabei das Problem der Verifikation von Testprogrammen adressiert, um Implementierungsfehler zu erkennen oder eine höhere Verifikationsabdeckung zu erreichen. Diese basiert auf dem Ansatz der Bauteilvariation, wie in [1] vorgestellt. An dieser Stelle wird erstmals die automatisierte Ausführungsemulation mehrerer Bauteilvarianten (*Device Variants*) beschrieben.
Jedes Testprogramm wird in der Regel vor dem Einsatz innerhalb des Produktionstests verifiziert. Durch die Anwendung geeigneter Verfahren soll gewährleistet werden, dass ein Testprogramm mit hoher Sicherheit zwischen korrekten und defekten Bauteilen unterscheiden kann. Neben der Anwendung statischer Verfahren wird das Testprogramm durch die Ausführung von Online-Testläufen mit Referenzbauteilen auf dem Testsystem verifiziert. Diese Testläufe erfolgen unter der Annahme, dass die Referenzbauteile, wie erwartet funktionieren. Demnach wird das Testprogramm ausschließlich darauf überprüft, korrekte Bauteile ordnungsgemäß zu erkennen. Eine Erweiterung um defekte Bauteile würde die Herstellung von Referenzbauteilen mit spezifischen Defekten erfordern. Aufgrund der Komplexität und der entstehenden Kosten stellt diese Art der Fehlerinjektion keine Möglichkeit dar.
Eine Methode der Fehlerinjektion, die für den Test von Software [2] eingesetzt wird, ist der Mutationstest [3]. Dabei werden durch Modifikationen am Quellcode verschiedene Versionen, sogenannte Mutanten, erzeugt und anschließend geprüft, ob die Testfälle diese erkennen. Neben dem Softwaretest ist dieses Testverfahren auch auf Hardwaredesigns [4] [5] [6] anwendbar. In [1] wird der Mutationstest erstmals für die Testprogrammverifikation eingesetzt.

Diese Forschungsarbeit wurde durch das Bundesministerium für Bildung und Forschung (BMBF) im Projekt VE-VIDES (FKZ: 16ME0252) gefördert.

II. Methodik

In [1] wird ein Prozess zur automatisierten Verifikation von Testprogrammen durch Bauteilvariation beschrieben. Das Testprogramm erhält während der Offline-Ausführung innerhalb der Softwareumgebung Eingabewerte von emulierten Bauteilen (*Ausführungsemulation*, siehe III). Nach der Ausführung werden die erhaltenen Testprogrammresultate mit der Erwartung aus der Spezifikation verglichen (siehe Abb. 1). Fehlende Übereinstimmungen weisen auf Fehler im Testprogramm hin und sind vor dessen Anwendung aufzulösen. Ergebnis der Verifikation soll ein Testprogramm sein, dessen Anwendung mit hoher Sicherheit korrekte und defekte Bauteile unterscheiden kann.

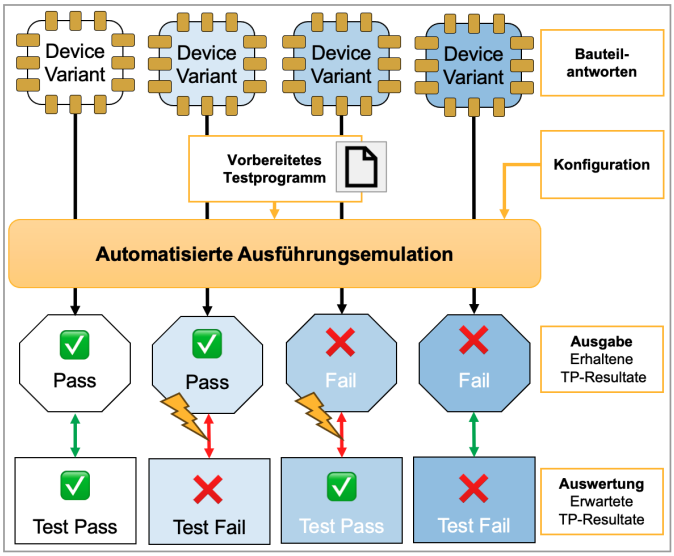

Abbildung 1. Automatisierte Ausführungsemulation und Auswertung

Zur formalen Spezifikation von Bauteilantworten (*Device Responses*) und erwarteten Testprogrammresultaten (*Expected Results*) ist ein abstraktes Format in einer erweiterten Testspezifikation entwickelt worden. Diese Testspezifikation

beschreibt die einzelnen Testschritte und verifikationsrelevanten Informationen durch verschiedene Attribute innerhalb eines Tabellenkalkulationsdokuments. Dabei stellt jede Zeile einen einzelnen Testschritt dar und jede Spalte enthält einen zugehörigen Attributwert. Zu den Standardattributen eines Testschrittes zählen u. a. Test Suite, Testname, Testnummer, Pins und Testgrenzen. Für die Testprogrammverifikation ist pro Testschritt die Angabe eines Variablennamens (*Result Identifier*) notwendig, der zur Speicherung der emulierten Bauteilantworten im Testprogramm dient.

Für die Spezifikation von Bauteilantworten und erwarteten Testprogrammresultaten ist ein eigener Befehlssatz entstanden. Dieser ermöglicht die Angabe durch Zahlenwerte oder Formeln. Neben dem Einsatz von Operatoren (+, -, *, /, (,)) ist die Verwendung von Parametern, Referenzen auf Testgrenzen (*LL - Untere Grenze*, *HL - Obere Grenze*) und eingebauten Funktionen (z. B. *random value [0;1]* zum Zurückliefern einer pseudo-zufälligen Zahl zwischen 0 und 1) möglich.

Eine gewisse Anzahl von emulierten Bauteilantworten für die individuellen Tests eines Testprogramms wird als Satz (*Device Response Set*) innerhalb der Testspezifikation definiert. Jeder Satz von Bauteilantworten repräsentiert dabei mindestens ein spezifisches Verhalten einer Bauteilvariante, das als korrekt (*pass*) oder defekt (*fail*) klassifiziert werden kann. Die Anwendung eines defekten Satzes von Bauteilantworten sollte dementsprechend zu einem nicht bestandenen Testlauf innerhalb der Ausführungsemulation führen. Defekte Bauteilsätze sind *Mutanten* und korrekte Bauteilsätze *funktional äquivalenten Mutanten* gleichzusetzen, deren Verhalten einer Variante eines als korrekt eingestuften Referenzbauteils entspricht.

Die Methodik sieht verschiedene Kategorien von Sätzen vor, die zur hinreichenden Verifikation eines Testprogramms dienen sollen. Vorgesehen sind ein nicht site-spezifischer Satz von Bauteilantworten innerhalb der Testgrenzen, zwei site-spezifische Sätze mit als defekt und korrekt klassifizierten Bauteilantworten und zwei permutierte Sätze. Bei site-spezifischen Sätzen werden durch die Multiplikation mit pseudo-zufälligen Faktoren verschiedene Werte für jede spezifizierte Site generiert. Permutierte Sätze führen zur Generierung mehrerer Bauteilvarianten, indem jeweils einem Testschritt eine Bauteilantwort zugewiesen wird, die zum Scheitern führt und den übrigen Testschritten das Gegenteil.

Die Verwendung eines Tabellenkalkulationsdokuments und die Formalisierung der Angaben erlaubt die automatische Generierung mehrerer Werte von Bauteilantworten und Resultaten durch automatisierte Analyse- und Generierungsmodule. Im ersten Schritt des Verifikationsprozesses dienen diese zur Berechnung der Werte von Bauteilantworten bzw. Testprogrammresultaten bei der Generierung der verschiedenen Bauteilvarianten aus den spezifizierten Sätzen.

III. Automatisierte Ausführungsemulation

Für den vollautomatisierten Produktionstest im Multisite-Modus müssen Test- und Handler-Systeme entsprechend konfiguriert und die Testdurchläufe einmalig gestartet werden. Testsysteme und deren Softwareumgebungen bieten dafür geeignete Programmierschnittstellen an. Es gilt, die Vorteile der automatisierten Produktionstestausführung, wie die gesammelte Ausführung mehrerer Bauteile ohne manuelles Eingreifen oder eine einzelne Resultatdatei aller Testdurchläufe, auf die Testprogrammverifikation anzuwenden. Bei der automatisierten Ausführungsemulation werden dabei physische Bauteile durch emulierte Bauteilvarianten und das Testsystem vollständig durch die Softwareumgebung ersetzt.

Unter Verwendung der verfügbaren Programmierschnittstellen ist eine Applikation für die Sitzungsverwaltung der Testersoftware, der Testprogramminitialisierung und der Ablaufsteuerung der Ausführungsemulation entwickelt worden. Die Applikation ist konfigurierbar und kann für verschiedene Testprogramme genutzt werden. Nach der Ausführung liegen die erhaltenen Resultate (*Obtained Results*) gesammelt in einer STDF-Datei [7] vor. Erwartete und erhaltene Testprogrammresultate werden verglichen und die Ergebnisse in Reports festgehalten. Dafür dient das neu hinzugefügte Auswerte- und Reportingmodul. In Abb. 2 sind die Abläufe und Module der Methodik einschließlich der automatisierten Ausführungsemulation zusammengefasst.

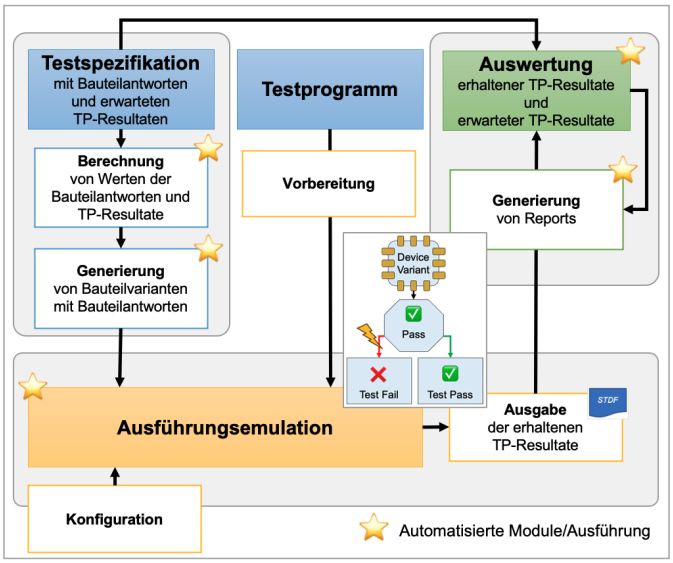

Abbildung 2. Testprogrammverifikation und Ausführung

IV. Demonstration und Auswertung

Demonstriert wird die Methodik und deren automatisierte Ausführungsemulation in der Softwareumgebung eines industriell eingesetzten Testsystems einschließlich der beschriebenen Plugins [8]. Ausgangspunkt bildet eine Testspezifikation mit 30 Testschritten und, den in II beschriebenen, fünf Kategorien von Sätzen mit Bauteilantworten für DC- und Funktionaltests einer Site. Da für die Testschritte keine Nachprozessierung erfolgt, entsprechen die erwarteten Resultate den emulierten Bauteilantworten. Daraus ergeben sich 193 Testläufe mit Bauteilvarianten. Die Ausführungszeit der automatisierten Ausführungsemulation beträgt dabei 100 Sekunden. In das Testprogramm ist vor der Ausführung ein Zuweisungsfehler eingebaut worden. Statt dem bereitgestellten

Messwert, wird ein mit null initialisierter Variablenwert zugewiesen. Da die Zuweisung innerhalb einer Schleife erfolgt, wirkt sich der Fehler auf mehrere Testschritte aus. Nach der Auswertung weisen die erzeugten Reports Abweichungen in den Testprogrammresultaten für die entsprechenden Testschritte auf. Da die Abweichungen eine Gruppe von Testschritten gleichermaßen betreffen, kann davon ausgegangen werden, dass der Fehler in gemeinsamen Initialisierungs-, Zuweisungs- oder Auswerteroutinen liegt. Da die betreffenden Testnummern bekannt sind, können auf diese Weise die Existenz von Implementierungsfehlern bestätigt und diese eingegrenzt werden.

V. Zusammenfassung und Ausblick

Vorgestellt wurde ein Prozess zur Testprogrammverifikation. Dieser basiert auf der Spezifikation emulierter Bauteilantworten, die während der Ausführung des Testprogramms im Offline-Modus bereitgestellt werden. Sowohl defekte als auch korrekte Bauteilvarianten können emuliert werden. Ausschließlich die Tester-Softwareumgebung ist notwendig. Demonstriert wurde der Prozess in der Softwareumgebung eines namhaften Testsystems.

Eine Erweiterung der Methodik um bis jetzt nicht unterstützte Testarten ist notwendig. Das Auswerte- und Reportingmodul sollte erweiterte Aussagen über das Maß der Verifikationsabdeckung oder Metriken zur Bewertung der Ergebnisse, z. B. dem Mutation Score, erfassen. Ebenfalls ist eine grafische Repräsentation der Ergebnisse denkbar. Eine geschlossene Integration in Anforderungsmanagement-Tools, durch Plugins in Entwicklungsumgebungen oder idealerweise in der Softwareumgebung von Testsystemen würde die Handhabung und Ausführung der Methodik zentralisieren und vereinfachen.

Literatur

[1] F. Mayer et al., "Automatic Verification of Mixed-Signal ATE Test Programs using Device Variation," 2021 IEEE International Test Conference (ITC), Anaheim, CA, USA, 2021, pp. 374-379, doi: 10.1109/ITC50571.2021.00053.

[2] Joan C. Miller and Clifford J. Maloney. 1963. Systematic mistake analysis of digital computer programs. Commun. ACM 6, 2 (Feb. 1963), 58–63. DOI:https://doi.org/10.1145/366246.366248

[3] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," in IEEE Transactions on Software Engineering, vol. 37, no. 5, pp. 649-678, Sept.-Oct. 2011, doi: 10.1109/TSE.2010.62.

[4] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson and J. Karlsson, "Fault injection into VHDL models: the MEFISTO tool," Proceedings of IEEE 24th International Symposium on Fault- Tolerant Computing, Austin, TX, USA, 1994, pp. 66-75, doi: 10.1109/FTCS.1994.315656.

[5] J.-H. Oetjens et al., "Safety evaluation of automotive electronics using Virtual Prototypes: State of the art and research challenges," 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 2014, pp. 1-6, doi: 10.1145/2593069.2602976.

[6] C. Constantinescu, "Using physical and simulated fault injection to evaluate error detection mechanisms," Proceedings 1999 Pacific Rim International Symposium on Dependable Computing, Hong Kong, China, 1999, pp. 186-192, doi: 10.1109/PRDC.1999.816228.

[7] Standard Test Data Format Specification, Version 4.0, Teradyne Inc.

[8] (2021, Sep.) SmarTest 7.6.2 Documentation, V93000 - Advantest Technical Documentation Center, Advantest Corp.