Die Zukunft der In-Circuit-Emulation – Code Coverage nach „embedded trace“

Prof. Dr. Christian Hochberger,
Dipl.-Ing. Alexander Weiss,
März 2007

Dokument Nr.: FA-GE-070312

Kontakt Autoren:

Prof. Dr.-Ing. Christian Hochberger
Technische Universität Dresden
Fakultät Informatik
Institut für Technische Informatik
Nöthnitzer Straße 46
01187 Dresden
Tel.: +49 351 463 39625
Fax : +49 351 463 38245
Email: christian.hochberger@inf.tu-dresden.de

Dipl.-Ing. Alexander Weiss
Accemic GmbH & Co. KG
Hochriesstr. 2
81326 Flintsbach
Tel.: +49 8034 90993-12
Fax : +49 8034 90993-27
Email: aweiss@accemic.com

Die Zukunft der In-Circuit-Emulation – Code Coverage nach „embedded trace“

Von Prof. Dr.-Ing. Christian Hochberger (TU Dresden) und Alexander Weiss (Accemic)

Immer schneller getaktete und immer komplexere Mikrocontroller erschließen neue, spannende Anwendungsbereiche, u.a. auch in sicherheitskritischen Applikationen. Sind die aktuellen Entwicklungstools in der Lage, mit dieser Entwicklung Schritt zu halten?

In den kommenden Jahren werden wir mehr und mehr technische Innovationen im Automotive-Bereich erleben, welche hauptsächlich durch die Verwendung von Mikrocontrollern, Mechatronik und der entsprechenden Betriebssoftware getragen werden. Dabei werden sowohl neue Komfort-Funktionen als auch sicherheitsrelevante Funktionen wie Fahrer-Assistenzsysteme immer mehr zu unserem Alltag gehören. Die aktive und passive Sicherheit kann gesteigert werden, indem Assistenten bei der Umsetzung von Fahrerwünschen (z.B. ABS, ESP), bei der normalen Fahrt (z.B. ACC), bei potentiellen Gefahrensituationen (z.B. LDW), bei akuter Gefahr (z.B. Emergency Break System) sowie bei und nach Unfällen helfen.

Fahrerassistenzsysteme erhöhen einerseits die Verkehrssicherheit, haben aber auf der anderen Seite auch das Potential, neue Gefährdungen in Folge von Fehlfunktionen zu erzeugen. Hier bemüht sich die Industrie, mit strengen Richtlinien für eine bestmögliche Funktionssicherheit sowie eine geringst mögliche Ausfallrate zu sorgen.
Grundlegende Sicherheitsnorm für diese Bereiche ist die IEC 61508 [1]. Diese sehr allgemein gehaltene Norm bedarf je nach Anwendungsbereich weiterer Detaillierungen. Vorreiter war hier die Luftfahrtindustrie, welche mit dem Standard RTCA DO-178B [2] eine geeignete Konkretisierung der Sicherheitsanforderungen definierte. Hier wird die Software in fünf Zertifizierungsstufen ("Software Level", Tabelle 1) unterteilt, die sich nach den Folgeschäden eines Ausfalls oder einer Fehlfunktion richten. Gleichzeitig werden je nach Zertifizierungsstufe Nachweise für Testabdeckungen ("Code Coverages") unterschiedlicher Komplexität verlangt.
Diese reichen von einer schlichten Überprüfung der Anforderungen über die sogenannte "Statement Coverage", bei der jede Anweisung des Quell-Textes einmal ausgeführt werden muss, bis hin zur "Modified Condition/Decision Coverage" bei der sogar jede mögliche Teilbedingung einer bedingten Anweisung für "wahr" und "falsch" durchlaufen werden muss.

Level	Gefahrenstufe	Geforderte Testabdeckung
A	Katastophal	Wie Level B, zusätzlich: Modified Condition / Decision Coverage (MC/DC)
B	Gefährlich/ schwerwiegend	Wie Level C, zusätzlich: Branch/Decision Coverage
C	Erheblich	Wie Level D, zusätzlich: Statement Coverage
D	Geringfügig	Abdeckung der Anforderungen
E	keine Auswirkungen	Keine speziellen Anforderungen

Tabelle 1: Geforderte Testabdeckungen für bestimmte Gefahrstufen (nach DO-178B [2])

Im Automobilbereich wird ebenfalls an geeigneten Konkretisierungen und Anpassungen der Sicherheitsanforderungen gearbeitet, u.a. an der Norm ISO WD 26262 [3] als „automotive“ Ableitung der IEC 61508. Weiter sollen hier AUTOSAR [4] oder das EU-Projekt „EASIS - Electronic Architecture and System Engineering for Integrated Safety Systems“ [5] genannt werden.

Sicherheit erfordert Trace

Neben ausgefeilten Werkzeugen zur fehlersicheren Generierung von Software ist der spätere Test der erzeugten Software von großer Bedeutung. Eine wichtige Grundlage dieser Tests sind die Trace-Daten, welche Informationen über das in einem Mikrocontroller ausgeführte Programm enthalten. Ein einfacher Trace enthält dabei die von der CPU ausgeführten Instruktionen. Für weitergehende Analysen wie z.B. die erwähnte "Modified Condition/Decision Coverage" werden zusätzlich die von der CPU gelesenen Daten benötigt. Darüber hinaus sind auch Informationen über von der CPU geschriebene Daten sowie die während eines DMA-Transfers veränderten Daten wichtig.

Im Sinne der oben geführten Diskussionen zur Überprüfung der Code-Abdeckung in sicherheitskritischen Anwendungen sollen hier nun kurz die Techniken genannt werden, die einen (zumindest lokal begrenzten) vollständigen Trace ermitteln können. Dabei soll auch im Einzelnen diskutiert werden, wie sehr eine Technik das System beeinflusst, wie realistisch die Timinginformation der gewonnen Trace-Daten ist und wie genau die Daten mit einem Programmlauf auf einem tatsächlichen Serienprozessor übereinstimmen. Außerdem spielt natürlich der Aufwand eine Rolle, den man treiben muss, um für eine neue Prozessorvariante ein entsprechendes Trace-Werkzeug zur Verfügung zu stellen. Für eine erweiterte Darstellung von Trace-Techniken sei hier auf [Bild 6] verwiesen.

Die momentan leistungsfähigste, aber auch teuerste Technik zur Gewinnung von Trace-Daten besteht im Einsatz von In-Circuit- Emulatoren (ICE). In diesem Fall werden spezielle Versionen der Chips hergestellt, die den Zugang zu allen relevanten internen Signalen über dedizierte Pins erlauben ("Bond-Out Chips"). Damit ist es dann z.B. auch möglich, spezielle Registerinhalte in Echtzeit aufzuzeichnen. Auf Grund der enormen Kosten (oft mehr als 10 000 Euro) und der Verzögerung zwischen der Verfügbarkeit von Mustern und der Verfügbarkeit von Emulatoren wird diese Technik in der Industrie immer seltener eingesetzt. Ebenso ist der Einsatz solcher Bond-Out Chips auf Arbeitsfrequenzen bis ca. 50 MHz beschränkt, da bei höheren Frequenzen die Beeinflussung der Signalqualität durch die Verkabelung so stark ist, dass es zu Funktionsstörungen kommt (z.B. DDR SDRAM Interface).

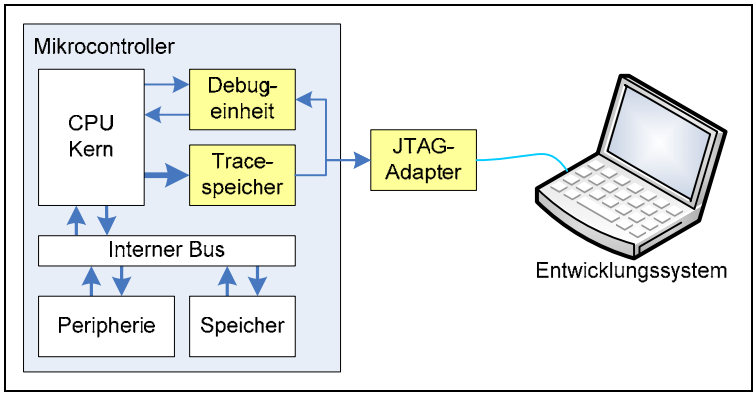

Abbildung 1: Gewinnung von Trace-Daten durch Nutzung der vorhandenen Debug-Fähigkeiten eines Mikrocontrollers in Kombination mit einem lokalen Speicher für Trace-Daten

Moderne Prozessoren enthalten zur Gewinnung von Trace-Daten mittlerweile öfter auch eine einfache Hardware-Unterstützung, die dann auch in den Serien-Chips enthalten ist. Die einfachste Variante hiervon ist in [Bild1] dargestellt. Durch Hinzunahme eines Trace-Speichers auf dem Chip können die relevanten internen Daten in diesen Speicher geschrieben werden. Der Trace- Speicher ist in der Regel als Ringspeicher organisiert, so dass die jeweils

jüngste Vergangenheit gespeichert ist. Es werden meist die Adressen der ausgeführten Instruktionen abgespeichert. Dieser Puffer kann nun durch die JTAG-Schnittstelle ausgelesen werden, wodurch allerdings die Bandbreite des Auslesens so stark limitiert ist, dass eine kontinuierliche Arbeitsweise in der Regel nicht möglich ist. Um dennoch einen möglichst großen Programmbereich abdecken zu können, besteht u.U. auch die Möglichkeit, die Daten, die in den Trace-Puffer geschrieben werden, zu filtern, so dass z.B. nur die Folgeadressen von Sprungbefehlen aufgezeichnet werden. Bei dieser Technik muss für den Trace-Puffer und die Filterlogik zusätzliche Chip-Fläche investiert werden. Da diese Technik meistens bei Serien-Chips angewandt wird, sind aus Kostengründen die Aufzeichnungstiefen stark limitiert. Dies führt dazu, dass der zu untersuchende Mikrocontroller ein paar hundert Instruktionen abarbeiten kann, und danach gestoppt werden muss, damit die Trace-Daten ausgelesen werden können.

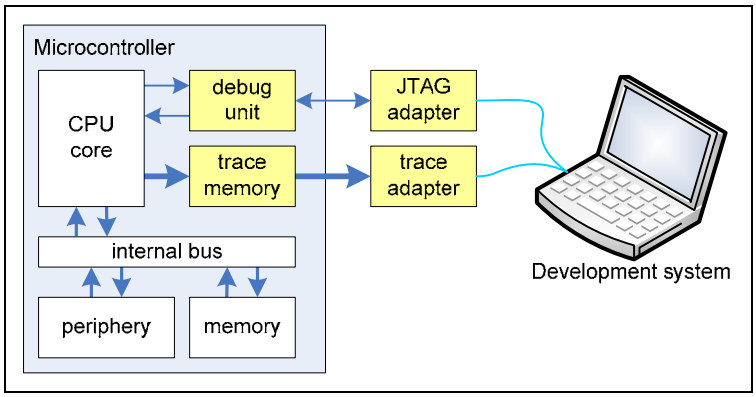

Abbildung 2: Gewinnung von Trace-Daten durch Nutzung der vorhandenen Debug-Fähigkeiten eines Mikrocontrollers in Kombination mit einem lokalen Tracespeicher und einer separaten Schnittstelle zur Ausgabe der Trace-Daten

Eine Verbesserung lässt sich erreichen, indem man nicht die JTAG-Schnittstelle zum Transport der Trace-Daten nach außen benutzt, sondern einen speziellen Trace- Bus auf dem Chip implementiert (häufig mit 4, 8 oder 16 bit Breite). Durch hinreichend hohe Taktraten auf diesem Bus (oft mehrere hundert MHz) ist man dann in der Lage, die erfassten Trace-Daten nach außen zu leiten [Bild 2]. Die externe Trace-Applikation mit einem speziellen Trace-Adapter kann dann nahezu beliebig tiefe Aufzeichnungen vornehmen, wobei aber immer noch zwischen der Komplexität der Hardware und dem Detaillierungsgrad der Aufzeichnung abgewogen werden muss. Auch hier muss oft ein Anhalten des Mikrocontrollers in Kauf genommen werden, um Lücken im aufgezeichneten Trace zu vermeiden.

Standardisierungsbemühungen

Da das Debug/Trace Problem natürlich alle Halbleiterhersteller trifft, gibt es seit einiger Zeit Standardisierungsbemühungen in diesem Bereich. Hieraus resultiert das so genannte NEXUS Forum [7]. NEXUS ist eigentliche eine Kurzbezeichnung für den Standard IEEE-ISTO 5001. Ziel dieses Standards ist es, mit den gleichen Werkzeugen Prozessoren verschiedener Hersteller debuggen zu können. Der Standard sieht hierbei vier verschiedene Klassen von Debugging vor. Diese reichen von einfacher Ablaufkontrolle (Breakpoints, Lesen/Manipulieren von Registern und Speicher, als Klasse 1 bezeichnet) bis zu umfangreichen Traces und fortgeschrittener Ablaufkontrolle (Echtzeit-Programm Traces, Write-Data Traces, Triggerung durch Watchpoints, Prozessor-Halt bei Trace Buffer Überlauf, als Klasse 4 bezeichnet).

In dem Bewusstsein des hohen Aufwandes verzichtet der NEXUS-Standard explizit auf die Forderung nach gleichzeitiger Erfassung von Daten- und Programm-Trace sowie auf einen Trace der DMA-Zugriffe.

Mehr Trace-Informationen als NEXUS erlaubt…
An dieser Stelle setzt die hidICE Technologie von Accemic an, welche auf verblüffend einfache Weise in der Lage ist, deutlich mehr Trace-Informationen zu liefern, als es im NEXUS-Standard definiert ist. Dieses Mehr an Trace-Daten ist kombiniert mit einem vergleichsweise geringen Aufwand, um die Trace-Daten von einem Mikrocontroller zu gewinnen.

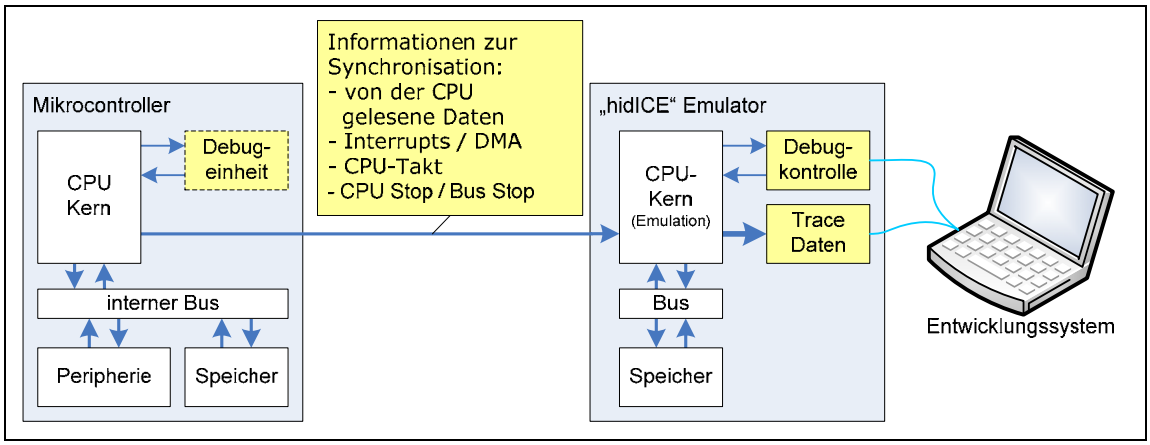

Abbildung 3: Prinzip der „hidICE“-Emulation

Das Grundprinzip der hidICE-Technologie besteht darin, dass die Trace-Daten nicht vom Target-Mikrocontroller direkt, sondern von einer parallel laufenden Emulation der CPU gewonnen werden. Die zweite CPU befindet sich direkt im Emulator und muss natürlich funktionsidentisch zur CPU des zu beobachtenden Mikrocontrollers sein. Wenn der Programmspeicher beider CPUs den identischen Code beinhaltet, führen die CPUs nach einem Reset initial auch die gleichen Instruktionen aus. Der individuelle Programmlauf entsteht erst dadurch, dass zur Laufzeit des Programms Daten gelesen und ausgewertet werden sowie durch den Einfluss auftretender Ereignisse (Interrupts). Die Idee hinter hidICE ist nun, dass nur die zur Synchronisation beider Programmabläufe erforderlichen Daten vom Target-Mikrocontroller zur emulierten CPU zu übertragen werden und damit sicher gestellt wird, dass beide CPUs die gleichen „äußeren“ Einflüsse erleben und deshalb auch den gleichen Programmablauf haben. Nun ist es ein leichtes, innerhalb des Emulators auf alle Trace-Informationen zuzugreifen, wobei hier neben den ausgeführten Instruktionen sowie gelesenen und geschriebenen Daten weitere wichtige Informationen wie Adressen und transferierte Daten während DMA-Transfers sowie ein Trace der CPU-Register nebenbei auch noch zur Verfügung stehen.

Bei den vorab diskutierten embedded-Trace Lösungen werden mit sehr großem Aufwand die Programmsprünge, sowie von der CPU gelesene und geschriebene Daten transportiert. Dabei wird viel Ingenieursarbeit darauf verwendet, diese Daten möglichst effizient zu übertragen und die Datenverluste bzw. die Beeinflussung des Systems durch notwendiges Anhalten der CPU gering zu halten.
Es ist leicht zu sehen, dass an dieser Stelle die zur Synchronisation beider CPUs in einem hidICE-System erforderlichen Daten nur einen Bruchteil der Datenmenge darstellt, die wir bei einer embedded Trace-Lösung zu übertragen haben.
Zur Synchronisation eines hidICE-Interfaces müssen im Wesentlichen nur Informationen über einen aufgetretenen Interrupt, über temporäre Stopps des Busses und der CPU sowie die aus dem IO-Bereich (UART, CAN, ADC usw.) gelesenen Daten übertragen werden. Wenn diese Informationen synchron mit dem CPU-Clock übertragen werden, ist es sehr einfach, den vollständigen Programmablauf in der emulierten CPU zu rekonstruieren.

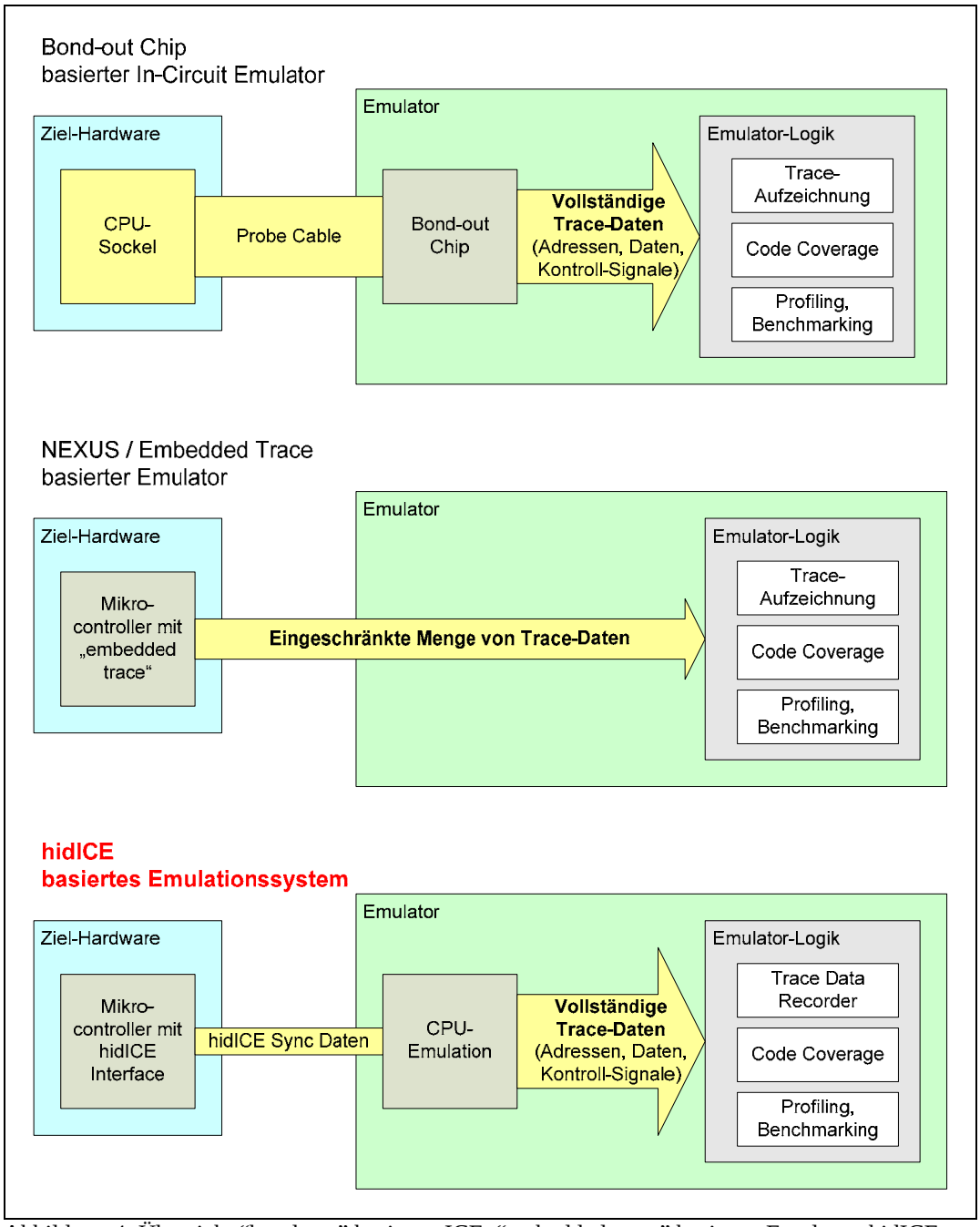

Abbildung 4: Übersicht “bond-out” basierter ICE, “embedded trace” basierter Emulator, hidICE

Der aufmerksame Leser wird sich nun die Frage stellen, was passiert, wenn ein Fehler in der Emulation auftritt. Dann laufen die beiden CPUs unweigerlich auseinander und der gewonnene Trace ist nutzlos. Um solch einen Fehler erkennen zu können, werden in der Target-CPU kontinuierlich Checksummen über gelesene und geschriebene Daten, ausgeführte Instruktionen sowie Adressen von Instruktionen und Daten gebildet. Wird diese Checksumme auch in der emulierten CPU berechnet, kann festgestellt werden, ob beide CPUs synchron laufen. Die Checksummen können dabei in der Zeit übertragen werden, wenn aktuell keine gelesenen Daten zu transportieren sind. Mit dieser Methode (hidICE System Integrity Control) kann aber nicht nur die korrekte Arbeitweise der Emulation überwacht werden, gleichzeitig werden nun auch sporadische Fehler erkannt, welche in der Target-CPU auftreten. Diese können beispielsweise durch instabile Speicherzellen, Probleme mit der Spannungsversorgung, zu enge Timings etc. verursacht werden und sind mit anderen Methoden nicht oder nur mit extrem großem Aufwand zu finden. Mit der hidICE-Technologie

kann der Zeitpunkt des Auftretens eines solchen Problems genau erkannt werden, obendrein steht noch ein Trace der Vorgeschichte des Fehlers zur Verfügung.

Breakpoints können in beliebiger Anzahl im Emulator implementiert werden. Damit entfällt die aufwändige Logik für Breakpoints in der Target CPU. Da die Emulation aufgrund der erforderlichen Übertragungszeit der gelesenen Daten um eine definierte Anzahl von CPU-Zyklen verzögert ist, hält die Target-CPU beim Auftreten einer gültigen Break-Bedingung im Emulator einige Taktzyklen verspätet an. Für die meisten Breakpoints ist dies akzeptabel, zusätzlich können aber auch in der Traget-CPU wie bisher gewohnt Breakpoints implementiert werden, die dann die Target-CPU ohne Verzögerung anhalten können. Wir empfehlen, emulatorseitige Breakpoints zur Abdeckung von Fehlerbedingungen zu verwenden, welche „eigentlich“ nicht vorkommen sollten und die Breakpoints der Target-CPU für die direkte Programmkontrolle (Step, goto etc.) zu verwenden.

Da sich auf der Target-CPU die Funktionalität des hidICE-Interfaces auf das „Einsammeln“ der für die Synchronisation relevanten Informationen (im wesentlichen CPU-Clock, gelesene Daten, Interrupts, DMA-Requests sowie Bus- und Pipeline-Stop) beschränkt, ist der Aufwand für die Implementierung eines hidICE Interfaces erstaunlich klein. Bei einer 32 Bit CPU kann man hier von weniger als 1000 Gates ausgehen.
Die Anzahl von IO-Pins, welche zur Ausgabe der Sync-Informationen erforderlich sind, hängt von der Busbreite eines Lesezugriffs auf den IO-Bereich sowie von der Anzahl der von der CPU für die Leseoperation benötigten Taktzyklen ab. Es werden aber auf jeden Fall deutlich weniger IO-Pins benötigt als bei einem NEXUS-Interface mit vergleichbarer Leistungsfähigkeit. Für einige gängige 16Bit MCUs werden 5-7 Pins benötigt, bei 32 Bit MCUs sind es 10-14 Pins. Diese Zahlen beziehen sich auf bisher getätigte beispielhafte Implementierungen und können von CPU zu CPU variieren.

Zusätzlich kann man diese Pins noch mit dem meist ohnehin vorhandenen JTAG-Interface gemeinsam nutzen. Wohlgemerkt, mit diesem geringen Aufwand ist der Zugriff auf einen simultanen Echtzeit-Trace von Instruktionen, gelesenen und geschriebenen Daten, DMA und CPU-Registern möglich – eine Funktionalität die den in NEXUS Level4 definierten Funktionsumfang (required und optional) deutlich übersteigt.

Mit dem auch bei NEXUS definierten Mechanismus zum Port Replacement kann die Anzahl der zur Ausgabe der Synch-Informationen benötigten I/O-Pins auf Null reduziert werden, sofern diese Pins mit einfachen Output-Funktionen belegt sind [Bild 5].

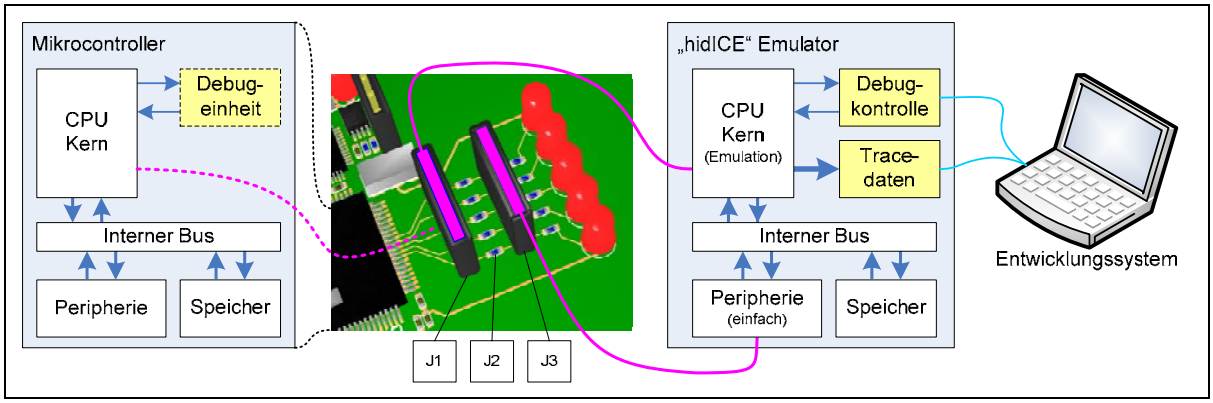

Abbildung 5: Port Replacement: Wenn der hidICE-Emulator angeschlossen ist, werden die zur Synchronisierung der CPU-Kerne erforderlichen Informationen über J1 an den Emulator übertragen. J2 ist offen. Die Ausgaben der emulierten Peripherie werden über J3 wieder auf die Ziel-Hardware eingespielt. Ist der hidICE-Emulator nicht angeschlossen, so werden die J2-Jumper angeschlossen und die Peripherie kann direkt vom Mikrocontroller angesprochen werden.

Da der Aufwand zur Gewinnung der Trace-Daten so gering ist, steht einer Verwendung eines hidICE-Interfaces in Serien-Mikrocontrollern nichts entgegen. Damit müssen keine Bond-Out- oder spezielle Evaluation-Chips mehr verwendet werden und die Software kann auf dem gleichen Mikrocontroller entwickelt und zertifiziert werden, welcher dann auch in der Massenfertigung verwendet wird. Ein unschätzbarer Vorteil, da es Halbleiterherstellern immer schwerer fällt, das absolut identische Verhalten von Evaluation-Chips und Mass Production Chips zu garantieren. Ist diese Übereinstimmung aber nicht gegeben, wird ein Großteil des Testaufwandes für ein neues Gerätes ad absurdum geführt, da letztendlich ein mit dem Mass Production Chip ausgeliefertes Gerät im Feld ein ganz anderes Verhalten zeigen kann als das zuvor mit einem Evaluation Chip getestete Gerät.

Die Funktionsfähigkeit des hidICE-Konzeptes kann mit einem hidICE Demonstrationssystem verifiziert werden. Dieses besteht aus 2 Xilinx Virtex4 Boards, welche über 5 LVDS Leitungen miteinander verbunden sind. Als CPU wurde die Xilinx PicoBlaze Implementierung gewählt, da diese bequemerweise vollständig in Verilog vorliegt. Das System ist mit 200 MHz getaktet und unterstützt asynchrone Ereignisse (Reset und Interrupt). Das hidICE Demonstrationssystem erlaubt die Aufzeichnung eines Traces, das Setzen von Breakpoints (sowohl in der Target-CPU als auch in der emulierten CPU). Mit einer eingeschleusten fehlerhaften Instruktion kann auch die zuverlässige Funktionsweise der hidICE System Integrity Control gezeigt werden.

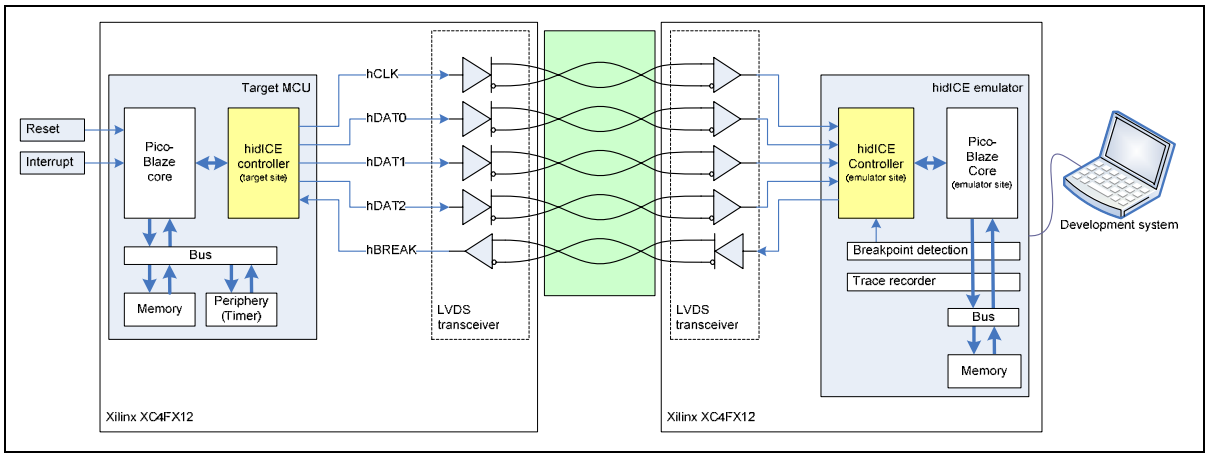

Abbildung 6: hidICE Demonstration System

Zusammenfassend lässt sich sagen, dass die hidICE-Technologie eine wesentlich über NEXUS Level 4 hinausgehende Funktionalität bietet. hidICE vereint die Vorteile der beiden etablierten Trace-Techniken, indem es die vollständigen Trace-Daten, die man von einem Bond-Out-Chip gewinnen kann, mit den hohen Taktfrequenzen einer embedded-Trace Lösung kombiniert. Die Implementierung einer hidICE-Schnittstelle in einem Mikrocontroller kann deutlich preiswerter als eine embedded Trace Lösung sein. Damit kann hidICE ohne größere Mehrkosten in Mass-Production MCUs eingesetzt werden und erlaubt die Entwicklung und Zertifizierung von Anwendungssoftware unter realen Bedingungen auf der gleichen Hardware, wie sie später auch im Feld eingesetzt wird.
Somit bietet hidICE eine überraschend einfache und kostengünstige Lösung, die die Zertifizierung von Anwendungssoftware auf einer Mass Production MCU selbst unter den härtesten Testbedingungen ("Modified Condition/Decision Coverage") gestattet.

Quellen

[1]	IEC 61508 Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems. IEC 1998.
[2]	Software Considerations in Airborne System and Equipment Certification- RTCA/DO-178B. RTCA Inc., Washington D.C., December 1992.
[3]	ISO/WD 26262
[4]	www.autosar.org
[5]	www.easis.org
[6]	Elmar Stahleder: Debugger mit Rückspiegel – Trace-techniken im Überblick, Elektronik 2005, H. 15, S. 34ff
[7]	www.nexus5001.org

Information zu den Autoren:

Prof. Dr. Christian Hochberger
absolvierte Studium und Promotion an der TU Darmstadt. Im Anschluss war er mehrere Jahre freiberuflich als Berater für Embedded-Systeme tätig. Nach einer vierjährigen Tätigkeit als Oberingenieur an der Universität Rostock wurde er 2003 an die TU Dresden berufen und lehrt und forscht dort im Embedded-Bereich.

Dipl.-Ing. Alexander Weiss
studierte an der TU München Elektrotechnik und war danach bei verschiedenen Projekten in der hochschulnahen Forschung im Bereich der Medizintechnik (Bildverarbeitung, vernetzte Mikrocontroller in sicherheitskritischen Applikationen) tätig.
Im Jahr 2001 wurde er Mitgründer und Geschäftsführer der Accemic GmbH & Co. KG, einem Spezialisten für innovative Mikrocontroller-Entwicklungstools.