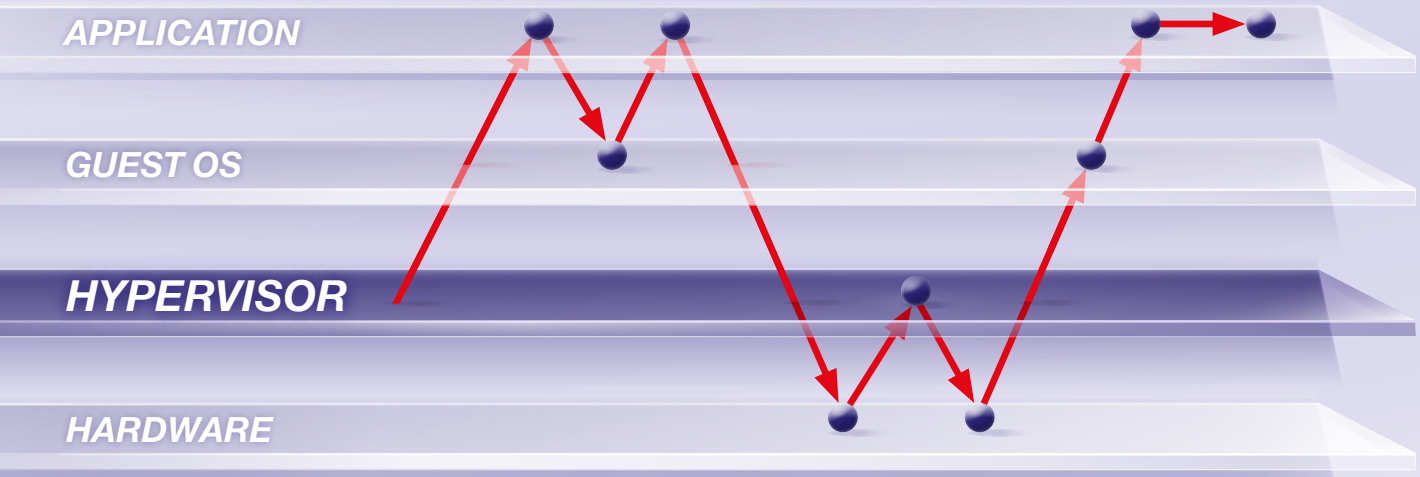
NEWS 2017

Deutsche Ausgabe

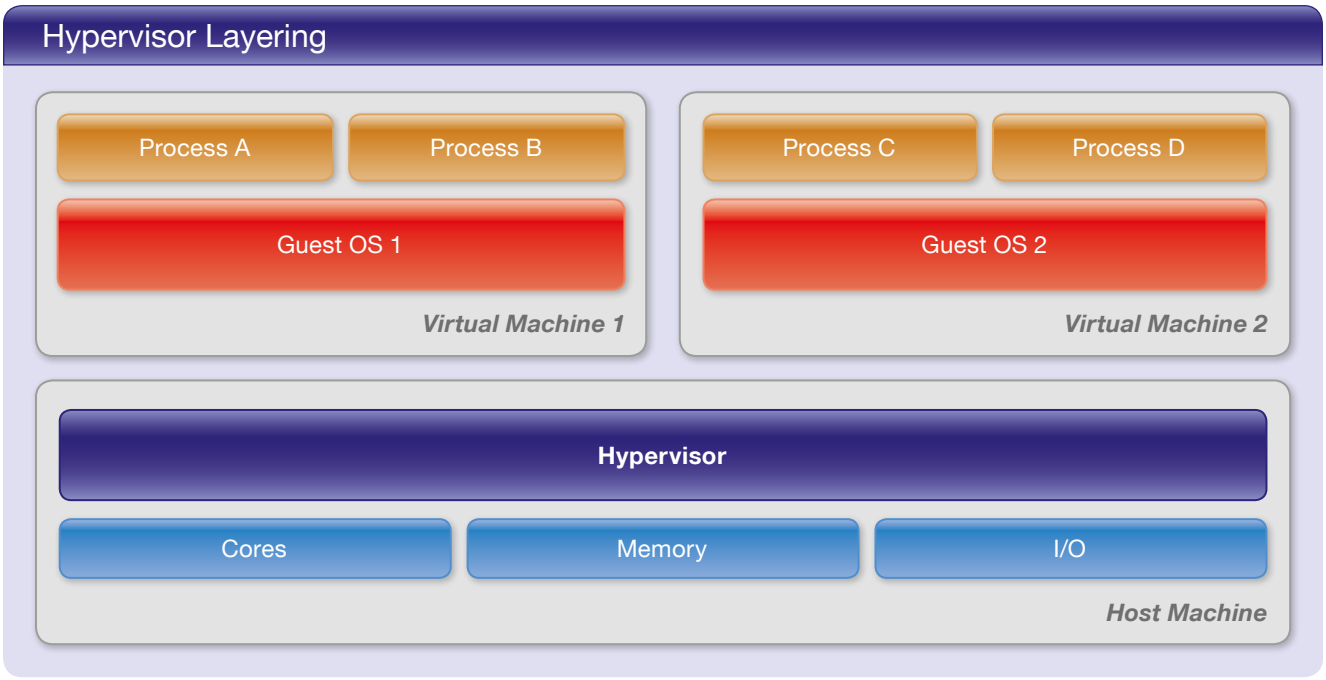


Seamless debugging through all software layers

INHALT

LAUTERBACH
DEVELOPMENT TOOLS

Hypervisor Debugging



Lauterbach präsentiert auf der embedded world 2017 erstmals die Leistungsfähigkeit seiner neuen Hypervisor-Unterstützung. Gezeigt wird eine Referenz-Implementierung, bei der auf dem HiKey-Board von LeMaker (Cortex-A53) ein Xen-Hypervisor mit zwei Linux-Guests läuft.

Virtualisierung in Embedded Systemen

Das Konzept der Virtualisierung ermöglicht es, mehrere Betriebssysteme gleichzeitig nebeneinander auf einer Hardware-Plattform zu betreiben. Die Virtualisierung kommt heutzutage zunehmend auch für Embedded-Systeme zum Einsatz, beispielsweise im Cockpit eines Autos. Echtzeitanwendungen, die klassisch von einem AUTOSAR-Betriebssystem überwacht werden, laufen hier parallel zu Android-basierten User-Interfaces auf derselben Hardware-Plattform. Ein Hypervisor, das Kernstück der Virtualisierung, sorgt dafür, dass alles sicher und effizient funktioniert.

Der Hypervisor, auch Virtual Machine Monitor genannt, ist eine Software-Schicht mit zwei Aufgaben:

1. Starten und Verwalten der virtuellen Maschinen (VMs).
2. Virtualisierung der physikalischen Hardware-Ressourcen für die VMs.

Ein auf einer VM laufendes Betriebssystem wird als Guest-OS bezeichnet. Alle Zugriffe der Guests auf die virtualisierten Hardware-Ressourcen werden vom Hypervisor auf die physikalischen Ressourcen abgebildet.

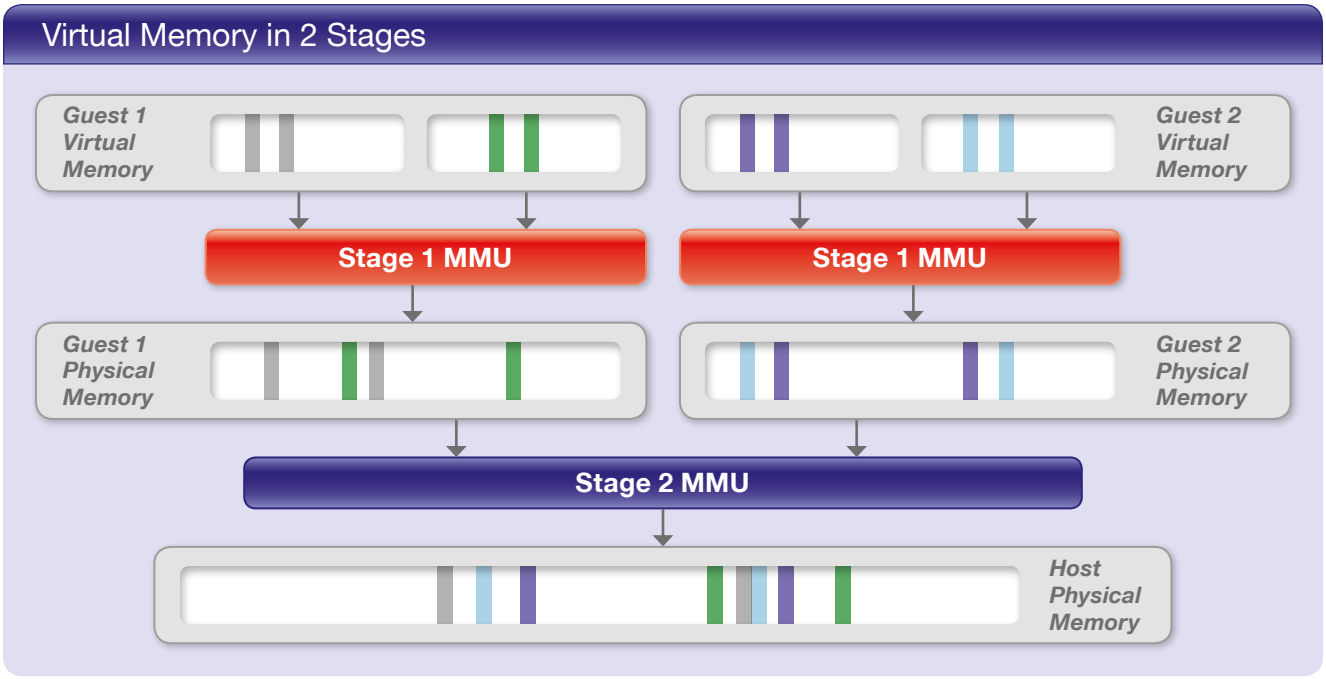
Für das Debugging von Bedeutung ist die CPU-Virtualisierung. Jeder virtuellen Maschine werden eine oder mehrere virtuelle CPUs (vCPUs) zugeordnet. Die Anzahl der vCPUs muss dabei nicht zwingend mit der Anzahl der CPU-Kerne auf der Hardware-Plattform übereinstimmen.

Ein zweites gleichermaßen wichtiges Thema ist die Speichervirtualisierung. Die VMs sehen nämlich nicht den physikalischen Speicher, sondern das sogenannte *Guest Physical Memory* als virtualisierten Speicher. Für den Zugriff auf den physikalischen Speicher verwaltet der Hypervisor für jede VM eine eigene Page Table. Da die Anwendungsprozesse, zumindest bei Betriebssystemen wie Linux, ohnehin mit virtuellen Adressen arbeiten, hat man es nun mit zwei verketteten Adressumsetzungen zu tun:

- Guest Virtual Memory auf Guest Physical Memory
- Guest Physical Memory auf Host Physical Memory

Siehe hierzu die Graphik „Virtual Memory in 2 Stages“ auf der gegenüberliegenden Seite:

- Die Stage 1 MMUs erhalten ihre Mapping-Informationen über die Page Table ihres Guest-OS.
- Die Stage 2 MMU greift auf die Page Tables des Hypervisors zurück.



Erweiterte Debug-Konzepte

TRACE32 wurde 2016 von Lauterbach systematisch erweitert, um seinen Kunden ein uneingeschränktes Debugging mit Hypervisor zu ermöglichen. Dazu wurden folgende Erweiterungen durchgeführt:

- Die TRACE32-Kommandosyntax wurde um eine Machine-ID erweitert. Diese erlaubt es, sowohl auf den Kontext der aktiven VM als auch auf den Kontext aller inaktiven VMs zuzugreifen. Eine virtuelle Maschine ist dann aktiv, wenn ihr ein Core zur Ausführung zugeteilt ist.
- Mittels der neuen Hypervisor-Awareness erkennt und visualisiert der Debugger die VMs des Hypervisors.
- Statt eines einzelnen Betriebssystems lassen sich nun mehrere Betriebssysteme gleichzeitig debuggen.
- Statt wie bisher nur auf die OS Page Tables der aktiven Guests zuzugreifen, kann der Debugger jetzt auch die Page Tables aller inaktiven Guests nutzen.

Die wichtigste Zielsetzung für alle Erweiterungen war ein nahtloses Debugging des Gesamtsystems. D.h., wenn das System an einem Breakpoint angehalten hat, kann man den aktuellen Zustand jedes einzelnen Prozesses, aller VMs, den aktuellen Zustand des Hypervisors und der realen Hardware-Plattform überprüfen und verändern. Zudem kann man an jede beliebige Stelle des Codes einen Programm-Breakpoint setzen.

Das uneingeschränkte Debugging, das Lauterbach seit nun fast 20 Jahren für Betriebssysteme wie Linux anbietet, bildete den Ausgangspunkt für alle durchgeführten Erweiterungen. Deshalb hier noch einmal eine kurze Zusammenfassung der wichtigsten Debug-Konzepte: Prozesse laufen in Betriebssystemen in einem privaten virtuellen Adressraum. Die OS-Awareness und der MMU-Support erlauben den TRACE32-Nutzern ein nahtloses Debuggen über Prozessgrenzen hinweg:

- Mit Hilfe der sogenannten Space-ID kann direkt auf die Adressräume der einzelnen Prozesse zugegriffen werden.
- Mit Hilfe der TASK-Option kann der aktuelle Registersatz und der Stackframe für jeden einzelnen Prozess dargestellt werden.

Machine-ID

Wie muss dieses Konzept erweitert werden, wenn die Betriebssysteme auf virtuellen Maschinen laufen?

1. Zunächst ist es erforderlich, virtuelle Maschinen eindeutig zu identifizieren. Dazu gibt TRACE32 jeder VM eine Nummer, die sogenannte Machine-ID. Der Hypervisor erhält 0 als Machine-ID. Ähnlich wie die Space-ID dazu dient, den virtuellen Adressraum eines Prozesses zu identifizieren, dient die Machine-ID dazu, den privaten Adressraum einer VM zu identifizieren.

TRACE32 Commands

Traditional OS-Aware Debugging	`Data.dump`	`<space _ id>:<virtual _ address>`
	`Data.LOAD.Elf <file>`	`<space _ id>:<virtual _ address>`
	`Register.view`	`/TASK <process _ name>`
	`Frame.view`	`/TASK <process _ name>`
< NEW > Hypervisor Debugging	`Data.dump`	**`<machine _ id>:::`**`<space _ id>::<virtual _ address>`
	`Data.LOAD.Elf <file>`	**`<machine _ id>:::`**`<space _ id>::<virtual _ address>`
	`Register.view`	**`/MACHINE <machine _ id>`** `/TASK <process _ name>`
	`Frame.view`	**`/MACHINE <machine _ id>`** `/TASK <process _ name>`

2. Um den Registersatz und den Stackframe eines beliebigen Prozesses darzustellen, muss der Debugger wissen, auf welcher VM bzw. unter welchem Guest-OS der Prozess läuft. Dazu wurde die MACHINE-Option eingeführt.

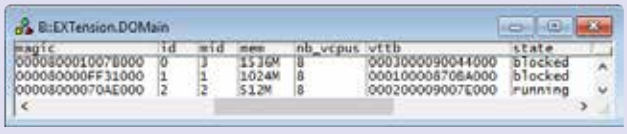
Diese beiden Erweiterungen reichen aus, damit der Debugger über Prozessgrenzen hinweg auf alle Informationen zugreifen kann. Die Übersicht „TRACE32 Commands“ oben stellt die erweiterte TRACE32-Kommandosyntax für das Hypervisor-Debugging der traditionellen, für das OS-aware Debugging genutzten Syntax gegenüber.

Hypervisor-Awareness

Analog zur OS-Awareness gibt es jetzt auch eine Hypervisor-Awareness. Durch sie erhält der Debugger alle Informationen über den auf der Hardware-Plattform laufenden Hypervisor. Die Hypervisor-Awareness setzt jedoch voraus, dass die Debug-Symbole für den Hypervisor geladen sind. Nun kann der Debugger eine Übersicht über alle Guests erstellen. Der Screenshot „Guest List“ für unsere Referenzimplementierung – Xen, Cortex-A53 – zeigt folgende Informationen:

- VM IDs und VM States, Anzahl der vCPUs pro VM
- Startadressen der Stage 2 Page Tables (vttb)

Guest List

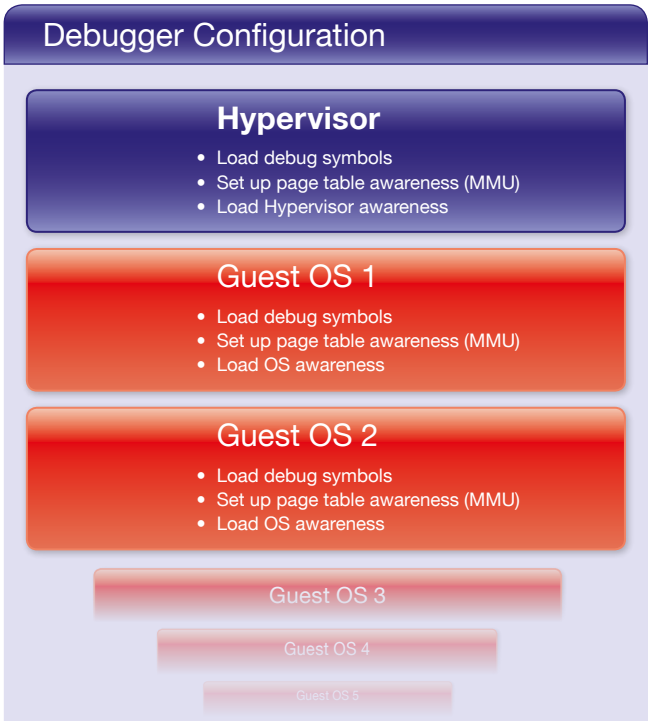


Die Awareness für die jeweiligen Hypervisoren wird von Lauterbach erstellt und seinen Kunden verfügbar gemacht. Eine Übersicht über alle aktuell unterstützten Hypervisoren zeigt die Tabelle „Currently Supported Hypervisors“ auf Seite 5.

Debugger-Konfiguration

Wie wirken sich die erweiterten Debug-Konzepte nun auf das Debugging mit TRACE32 aus? Schauen wir zunächst einmal auf die Konfiguration. Sowohl für den Hypervisor, als auch für jedes einzelne Guest-OS

Debugger Configuration

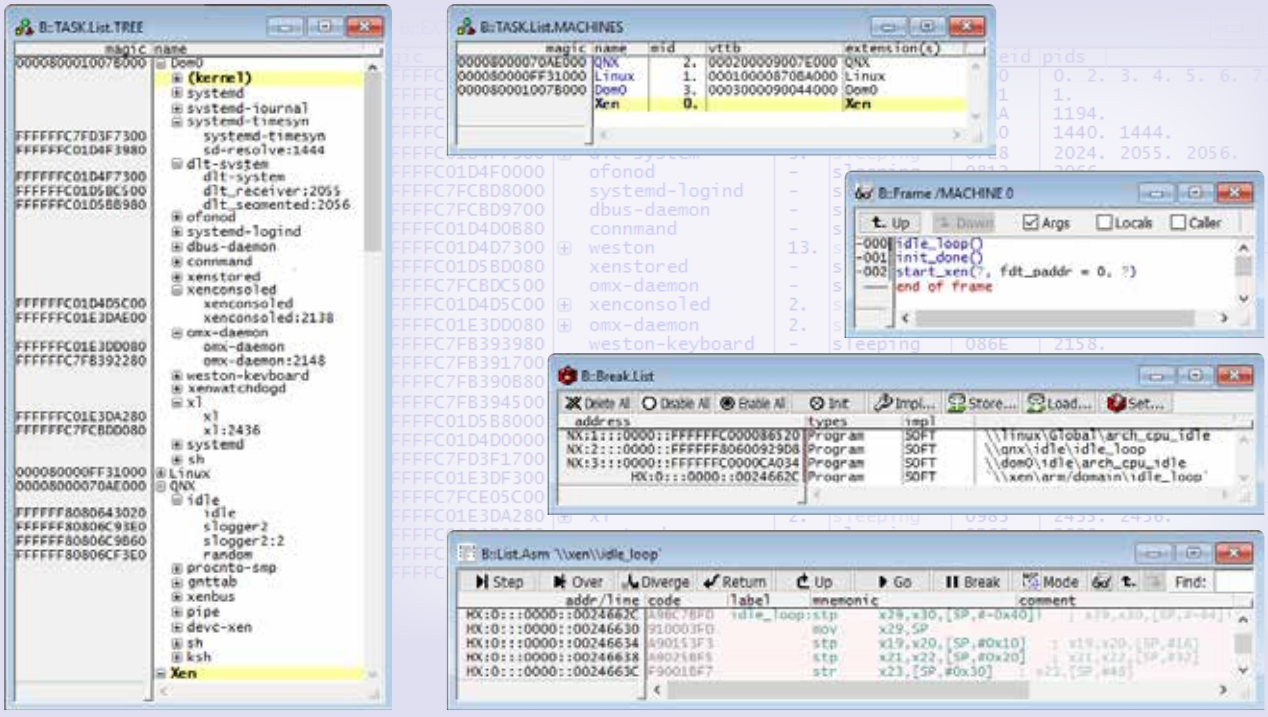


LAUTERBACH DEVELOPMENT TOOLS

Xen Hypervisor on Cortex-A53

Global Task List

Virtual Machine List



sind folgende Konfigurationsschritte notwendig: Debug Symbole laden, *Page Table Awareness* einrichten (MMU), TRACE32 Hypervisor-Awareness bzw. TRACE32 OS-Awareness laden. Die Graphik „Debugger Configuration“ zeigt die einzelnen Konfigurationsschritte im Überblick.

Debug-Prozess

Bei der Bedienung eines Debuggers stehen sich meist widersprüchliche Anforderungen gegenüber. Die eine Nutzergruppe wünscht sich eine einfache und intuitive Bedienung, während eine andere maximale Flexibilität und volle Skriptbarkeit einfordert. Werfen wir also einen Blick auf die intuitive Bedienung. Die Grundidee ist eigentlich ganz einfach: Hält der Debugger an einem Breakpoint an, visualisiert die GUI den Anwendungsprozess, durch den der Breakpoint ausgelöst wurde.

Interessiert man sich für einen anderen Anwendungsprozess, öffnet man einfach die globale Taskliste. Dort werden alle Tasks des Gesamtsystems gelistet. Mit einem Doppelklick kann man dann den Task auswählen, den man in der GUI angezeigt haben möchte. Die globale Taskliste bietet zudem eine einfache Möglichkeit, Programm-Breakpoints gezielt für einen Task zu setzen. Da die Debugsymbole beim Laden des Elf-File einer Machine-ID und einer Space-ID zugeordnet werden, können Funktionen und Variablen während des Debuggings wie gewohnt mit Namen angesprochen werden.

Maximale Flexibilität und volle Skriptbarkeit erhält man über die TRACE32 Kommandos, deren erweiterte Syntax bereits vorgestellt wurde.

Fazit

Da Lauterbach die bekannten Konzepte aus dem OS-Aware-Debugging systematisch für das Hypervisor-Debugging erweitert hat, wird den TRACE32-Nutzern der Umstieg einfach und mit geringer Einarbeitung gelingen.

Currently Supported Hypervisors

KVM	Wind River Hypervisor 2.x
VxWorks 653 3.x	Xen
	more to follow

Aktualisierung der Debug-Tools für Intel® x86/x64

Seit Januar 2017 liefert Lauterbach den neuen CombiProbe Whisker MIPI60-Cv2 aus. Die TRACE32 CombiProbe und die TRACE32 QuadProbe bieten nun für den Intel® Converged MIPI60 Connector durchgängig die gleichen Debug-Features an:

- Standard JTAG, Intel® Debug HOOKs mit Pmode und I2C Bus
- Merged Debug Ports (zwei JTAG Chains)
- Intel® Survivability Features (Threshold, Slew Rate ...)

Darüber hinaus unterscheiden sich die Debug-Tools durch ihr Einsatzgebiet. Die **TRACE32 QuadProbe**, die speziell für Server-Prozessoren ausgelegt ist, ist ein reines Debug-Tool und erlaubt das SMP-Debugging von hunderten von Threads für Targets mit bis zu vier Debug-Steckern.

Die neue **TRACE32 CombiProbe** mit **MIPI60-Cv2 Whisker,** die für Client- sowie für Mobile-Device-Prozessoren ausgelegt ist, erlaubt neben dem Debugging auch das Aufzeichnen und Auswerten von System-Tracedaten. Dafür unterstützt sie einen 4-Bit und einen 8-Bit Traceport mit mittlerer Bandbreite.

Die **TRACE32 CombiProbe** mit **DCI OOB Whisker** ist speziell für das Debuggen und Tracen von Endprodukten ohne Debug-Stecker ausgelegt. Unter der Voraussetzung, dass der Chip einen DCI-Manager enthält, können Target und Debugger direkt über die USB3-Schnittstelle Debug- und Tracemessages austauschen. Das dabei verwendete DCI-Protokoll unterstützt Standard JTAG und die Intel® Debug HOOKs sowie Tracemessages für die Aufzeichnung von System-Traceinformationen.

Debugger and System Trace for Intel® Converged MIPI60 Connector

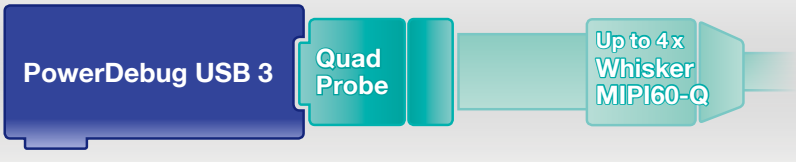
(devices and client applications)

Debugger and System Trace for USB3 Connector

(all applications)

Debugger for Intel® Converged MIPI60 Connector

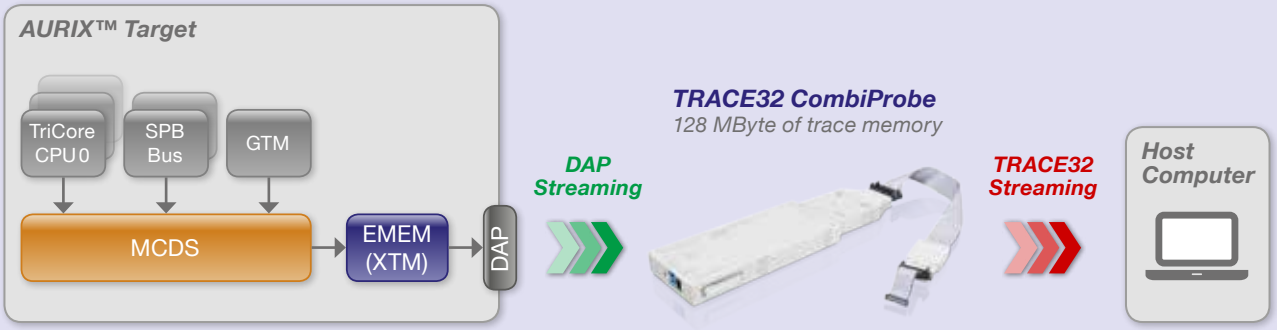
(server)



LAUTERBACH DEVELOPMENT TOOLS

TRACE32 CombiProbe TriCore DAP

DAP Streaming and TRACE32 Streaming



Seit Oktober 2016 liefert Lauterbach seine neue CombiProbe TriCore DAP für die AURIX™-Familie von Infineon aus. Damit bietet TRACE32 ab sofort umfassende Laufzeitanalysen für all die AURIX-User, deren Target-Hardware keine AGBT-Schnittstelle bereitstellt.

DAP Streaming

Die CombiProbe realisiert eine neue Technologie namens DAP-Streaming: Der Inhalt des Onchip-Tracespeichers wird zur Programmlaufzeit ausgelesen und lückenlos in den 128 MByte Tracespeicher der CombiProbe übertragen. Dafür muss der Chip eine Highspeed-Schnittstelle bereitstellen. Die AURIX™ DAP-Schnittstelle verfügt über passende Kenndaten: bis zu 160 MHz DAP-Taktfrequenz, Datenraten von bis zu 30 MByte/s. Zwar reicht die DAP-Bandbreite nicht, um den gesamten Programmfluss zu übertragen, trotzdem können umfangreiche Analysen durchgeführt werden:

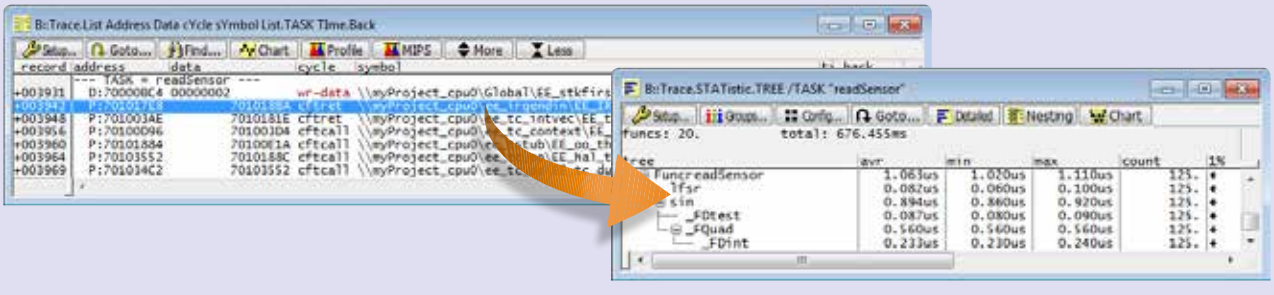
- Funktionslaufzeitmessung mittels des Compact Function Trace. Das ist ein spezieller Programmtrace-Mode, bei dem nur für Funktionseintritte (cftcall) und Funktionsaustritte (cftret) Tracedaten generiert werden. Die Graphik „Compact Function Trace“ unten zeigt beispielhaft den Aufrufbaum und die Laufzeitdetails, die TRACE32 auf Basis dieser Tracedaten berechnet.
- Analyse von ausgewählten Variableninhalten über die Zeit.
- Laufzeitmessungen für Tasks, ISR2s und OS-Services.

TRACE32 Streaming

Sollte der 128 MByte große Tracespeicher der CombiProbe nicht ausreichen, um alle relevanten Tracedaten aufzunehmen, lässt sich das DAP-Streaming mit dem TRACE32-Streaming kombinieren. TRACE32-Streaming überträgt die Tracedaten nach ihrem Eintreffen auf der CombiProbe sofort zum Hostrechner und legt sie dort in einer Datei ab. So ist es möglich, für Langzeitmessungen mehrere TByte an zusammenhängenden Tracedaten zu erfassen.

Mehr Informationen dazu finden Sie unter:
www.lauterbach.com/8467

Compact Function Trace



ARTI – AUTOSAR Run-Time Interface

Das OS-aware Debuggen und Tracen wird mit dem neuen ARTI-Standard an die aktuellen Anforderungen der Automobilindustrie angepasst. Lauterbach wirkt als offizieller AUTOSAR-Development-Partner aktiv an der Gestaltung dieses Standards mit, dessen Veröffentlichung für Anfang 2018 geplant ist.

Der ORTI-Standard, der seit 2003 durchgängig in der Automobilindustrie zum Einsatz kommt, unterstützt heute weltweit Tausende von Entwicklern beim Debugging und Profiling ihrer AUTOSAR-Systeme. Doch der Standard ist in die Jahre gekommen. Eine Aktualisierung steht dringend an.

Ziele

Neue Methoden zur Software-Entwicklung, Multicore- und Multi-ECU-Systeme, sowie gestiegene Anforderungen an die Validierung echtzeitkritischer Systeme – das alles muss der neue Standard abdecken.

Viele der Debug-, Trace- und Profiling-Features, die neu in den ARTI-Standard aufgenommen werden, sind bereits heute als proprietäre Lösungen im Einsatz. Das heißt, die Funktionalität hat sich bereits bewährt. Was fehlt, sind standardisierte Schnittstellen zwischen den unterschiedlichen im Entwicklungsprozess eingesetzten Tools. Dazu ein Beispiel.

Export von Tracedaten

Seit 2014 arbeitet Lauterbach eng mit verschiedenen Herstellern von Tools zur Validierung und Optimierung von Automotive-Software zusammen. TRACE32 exportiert die aufgezeichneten Echtzeit-Tracedaten, die dann in das externe Tool geladen und dort umfassend analysiert werden. Was fehlt derzeit?

1. Das vom Build-Tool bereitgestellte ORTI-File enthält lediglich Informationen zu den Tasks, den OS-Services und den ISR2s, jedoch keine Informationen zum Taskstart und Taskende, sowie zu den sogenannten Runnables. Vor dem Export muss diese fehlende Information in TRACE32 durch eine Nachbearbeitung der Traceaufzeichnung ergänzt werden.
2. Es gibt kein standardisiertes Format für den Export von Tracedaten. Das heißt, die externen Tools müssen das proprietäre TRACE32-Format einlesen.

Der neue ARTI-Standard wird beide Lücken schließen. Das vom Build-Tool bereitgestellte ARTI-File wird Informationen zu allen AUTOSAR-Objekten enthalten, gleichzeitig wird der Export von Tracedaten standardisiert.

Fazit

Da neben allen wichtigen Toolherstellern auch Toolnutzer an der Ausarbeitung des neuen ARTI-Standards mitwirken, wird dieser sicher ebenso erfolgreich und langlebig sein wie sein Vorgänger.