Was macht Ihr Prozessor jetzt gerade?

Techniken zur Messung von Software-Aktivitäten in Echtzeit

Ulrich Dreher, iss innovative software services GmbH

Obwohl Debugger, Emulatoren und andere Entwicklungshilfsmittel in den vergangenen Jahrzehnten gewaltige Fortschritte gemacht haben, fehlt ihnen eine Funktion, die die Entwicklung, Fehlersuche und Validierung von Echtzeitsystemen erheblich erleichtern würde: die Möglichkeit, Software-Events und Ereignisse der ‚realen Welt' in ‚harter Echtzeit' miteinander zu verknüpfen.

Dieser Beitrag beschäftigt sich damit, diese Schwachstellen zu überwinden: Er zeigt Techniken, die die Messung ausgewählter Software-Eigenschaften in Echtzeit erlauben. Der Schwerpunkt liegt dabei auf der Geschwindigkeit und dem Umfang, in dem Messdaten zur Verfügung gestellt werden können, die Aussagen über Abläufe in der Software erlauben. Mit Hilfe dieser Daten können Software-Abläufe dann mit Ereignissen der ‚realen Welt' in Beziehung gesetzt werden.

Ein kleines Glossar

Im Folgenden werden Begriffe verwendet, die möglicherweise nicht allgemein gebräuchlich sind. Daher an dieser Stelle ein kleines Glossar:

Begriff(e)	**Bedeutung**
PCB	Printed Circuit Board - Platine
Target Steuergerät	Bezeichnet eine Einheit, deren Funktion durch einen Prozessor oder Controller und dessen Software dominiert ist. Wobei auf einem „Steuer"gerät durchaus auch eine Regelung implementiert sein kann. Damit ist ggf. **Ihr** Gerät gemeint.
physikalisches Ereignis physikalische Größe reale Welt	bezeichnen in unterschiedlichem Kontext „Dinge", die sich „anfassen" bzw. messen lassen. Im Gegensatz dazu ist „Software" zwar nicht surreal, aber messtechnisch in der Regel nur schwer erfassbar.
(endlicher) Zustandsautomat	bekannt auch unter der englischen Bezeichnung „(finite) state machine"
Echtzeit	wird im folgenden Kapitel noch näher betrachtet

Was ist „Echtzeit"?

„Echtzeit"-Anforderungen sind bei Steuerungs- und Regelungssystemen allgegenwärtig. Die Definition von „Echtzeit" ist allerdings in erheblichem Maße von der jeweiligen Anwendung abhängig: „Echtzeit" kann Reaktionszeiten (Zykluszeiten) bezeichnen, die sich um Größenordnungen unterscheiden (s. Tabelle 1).

Zykluszeit	Beispiele für Anwendungen
100 ms	Benutzerschnittstellen, SPS (speicherprogrammierbare Steuerung)
1 - 10 ms	„klassische“ Steuerungssysteme
10 µs - 200 µs	Frequenzumrichter („Inverter“) für Synchronmaschinen
< 10 µs	„Digital Power Supply“ (digital geregelte Stromversorgungen) z.B. ein 1 kW AD/DC Netzteil mit einer Regelschleife mit 200 kHz

Tabelle 1 Beispiele für unterschiedliche Interpretationen von „Echtzeit“

Tabelle 1 führt nur einige wenige Beispiele auf, zwischen den angegebenen Zeitbereichen bestehen zudem Lücken. In der Realität tendiert jede Anwendung dazu, das Attribut „echt“ in Bezug auf zeitliche Anforderungen für sich neu zu definieren, wobei gerne zwischen „harter“ Echtzeit (Reaktionszeiten bis max. 5 ms) und „weicher“ Echtzeit (Reaktionszeiten > 5 ms) unterschieden wird. Auch die Grenze von 5 ms ist willkürlich gewählt - je nach Organisation und Aufgabenstellung mag ein anderer Wert zur Abgrenzung herangezogen werden.

Wie schon erwähnt, haben Emulatoren und Debugger in den letzten Jahrzehnten gewaltige Fortschritte hinsichtlich der gebotenen Funktionen und ganz allgemein hinsichtlich der Verarbeitungsleistung gemacht. Aber eine Funktion zur Verknüpfung von Ereignissen der „realen Welt“ und „Software-Ereignissen“ fehlt. Es mag heute den einen oder anderen Debugger geben, der über eine (kleine) Zahl digitaler Eingänge verfügt. Wenn es sich allerdings darum handelt, Ereignisse der realen Welt mit Vorgängen in der Software zu korrelieren, ist dies selten ausreichend.
Und Software-Werkzeuge, die mit hoher Geschwindigkeit Messwerte aufzeichnen können, sind auch nicht besonders hilfreich, wenn die aufgezeichneten Daten erst visualisiert werden, nachdem das auslösende Ereignis längst Geschichte ist.

Es gibt jedoch einige Techniken - teilweise sehr einfach und trotzdem weitgehend unbekannt - die geeignet sind, die vorstehend beschriebenen Lücken zu füllen.
Nachfolgend sind einige dieser Verfahren beschrieben, die es erlauben, Beziehungen zwischen physikalischen Ereignissen und Software-Abläufen aufzudecken. Sie ermöglichen es, Spezifika der Software (wie z.B. Interrupt-Latenzen, die Task-Reihenfolge oder Variablen der Steuerungssoftware) in Echtzeit mess- und damit auswertbar zu machen. Dies ist eine Fähigkeit, die den bislang vorhandenen Werkzeugen abgeht.

Betrachten wir also einige regelmäßig wiederkehrende Fragestellungen und mögliche Verfahren, das „Arbeiten“ von Software messbar zu machen, und beurteilen wir, wie gut sie sich zur Beantwortung der gewählten Fragen eignen.

Die Fragen
Nachfolgend sind Abstraktionen einiger Fragen aufgelistet, die in meiner Arbeit immer wieder gestellt wurden. Anwendungsbeispiele aus der Vergangenheit versuchen, die Fragestellung zu verdeutlichen.

- **„Wie viel Rechenleistung ist eigentlich noch verfügbar?“**

  Eine Frage, die mit schöner Regelmäßigkeit auftaucht.

  Eine Episode aus der Entwicklung des Inverters einer PMSM mit einer 20 kHz Stromregelung: Die ursprüngliche Antwort des zuständigen Entwicklers war „noch eine ganze Menge“.
  Die Anwendung einer der nachfolgend beschriebenen Methoden brachte zu Tage, dass die aktuelle Auslastung der CPU bei 95 % lag. Von diesem Zeitpunkt an war die Prozessorlast unter ständiger Kontrolle.
  Das Wissen um die Kritikalität der Prozessorlast war der Schlüssel dazu, dass „überraschende“ Probleme im weiteren Verlauf des Projekts deutlich reduziert auftraten.

- **„Verhält sich meine Regelung wie notwendig und erwartet?“**

  Ein Beispiel aus der Weiterentwicklung eines Steuergeräts der Einspritztechnik: zur Steuerung der Abläufe wurde ein Zustandsautomat implementiert. Üblicherweise werden Zustandsautomaten in einer einzigen Funktion „verortet“, die zyklisch oder ereignisgesteuert aufgerufen wird. Sie sind ausgesprochen beliebt, weil sie einfach zu implementieren und ausgesprochen robust sind.

  Im vorliegenden Fall schied die Standardlösung aus: die Zustände waren teils zeitgesteuert, teils ereignisgesteuert. Und aufgrund technischer Randbedingungen mussten die einzelnen Teile des Zustandsautomaten „in der ganzen Software verstreut“ implementiert werden.

  Die Entwicklung erfolgte mit Hilfe eines Tools, das die Messung und Visualisierung der Zustandsvariablen erlaubte: die Abfolge war sehr klar $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow ... \rightarrow 7 \rightarrow 0$. Abgesehen von einem vorzeitigen Abbruch ($\ldots \rightarrow 0$) musste die Sequenz in immer gleich Reihenfolge durchlaufen werden. Der zeitliche Ablauf allerdings war abhängig vom Eintreten physikalischer Ereignisse. Das Einhalten der korrekten Reihenfolge war am Oszilloskop visuell sehr einfach zu überprüfen, genau wie die Korrelation mit den physikalischen Ereignissen.

- **„Warum funktioniert die Software nicht wie sie soll?“**

  Eine immer wieder gestellte Frage.

  Bei der Entwicklung eines schnellen Messsystems, das die einzelnen Zellen eines Brennstoffzellensystems überwachte, wollte der Kunde noch ein paar besondere „Features“ (Unterabtastung, unterschiedliches Sequencing der Zellen).
  Die Implementierung des „Standardablaufs“ der Messung war einfach. Die Realisierung der zusätzlichen Features war dagegen aufgrund komplexerer Abfolgen und Algorithmen sehr aufwändig. Und schlecht zu debuggen, da die reinen Zahlenwerte nicht sonderlich aussagekräftig waren.

  Das Vorhandensein eines speziellen Entwicklungshilfsmittels (der „historischen“ OLDA) half enorm: die entscheidenden Variablen (2 Werte) konnten in Echtzeit visualisiert werden. Wie das „Bild“ dieser beiden Variablen auf dem Oszilloskop aussehen musste, war klar. Ein einziger Durchlauf der Software reichte jeweils zur Beurteilung gut/schlecht aus.

- **„Verhält sich mein System wie geplant?“**
  Diese Frage wird ebenfalls mit schöner Regelmäßigkeit gestellt (s. erster Punkt).
  Bei der Systemkonzeption wird in der Regel geplant, welche Teilaufgaben in welchen Tasks und Prozessen zu implementieren sind. Da nicht alles läuft, wie ursprünglich geplant, ist eine regelmäßige Kontrolle des Status quo von Nöten. Dafür gibt es mittlerweile eine ganze Anzahl von Softwarelösungen. Sobald allerdings „Events“ ins Spiel kommen, stoßen Softwarelösungen schnell an ihre Grenzen. Ein unabhängiges Tool, das nur ein absolutes Minimum an Instrumentierung benötigt, ist hier von Vorteil.

- Die „Königsdisziplin“: **Die Evaluierung von sin/cos Winkelsensoren.**
  Aktuell werden zur Winkelerfassung von Synchronmaschinen bevorzugt sin/cos Winkelsensoren eingesetzt, da diese eine hohe Auflösung bei vergleichsweise geringem Aufwand bieten. Um von den Werten zum Winkel zu gelangen, benötigt man die Arkustangens-Funktion. Wie das Leben so spielt, hat die Arkustangens-Funktion alle 90° eine kritische Phase, in der geringe Fehler in den Messwerten von Sinus und Kosinus zu gewaltigen Winkelfehlern führen. Wäre es nicht schön, die gemessenen Werte kontinuierlich überwachen und evtl. mit den physikalischen Messwerten - oder auch mit dem berechneten Winkelwert - vergleichen zu können?

- Zum Abschluss eine zunehmend wichtiger werdende Frage:
  „Wie wirkt sich eine Änderung der Hardware-Konfiguration aus?“
  Zu beurteilen sind z.B. Einflüsse der Cache-Größe, der Cache-Konfiguration, der Verwendung von Inline-Code usw.
  Im Grunde genommen kann diese Frage durch Verwendung der bereits erwähnten Softwarelösungen zur Laufzeitmessung beantwortet werden. Diese erfordern allerdings eine umfangreiche Code-Instrumentierung, die wahrscheinlich die Messwerte verfälscht.
  Ein weniger invasives Messverfahren wäre wünschenswert.

Einige Vorbemerkungen

Wenn es um Echtzeitmessungen an „realen“ Objekten geht, taucht umgehend die Frage auf, welche Ressourcen eigentlich zur Verfügung stehen, um diese Messungen durchführen zu können. Um die meisten (wenn vielleicht auch nicht alle) Targets mit einem einzigen Werkzeug unterstützen zu können, müssen die Erwartungen an die Verfügbarkeit von Ressourcen auf ein Minimum reduziert werden. Diese Ressourcen sollten gleichzeitig ein Maximum an Geschwindigkeit bieten, um die Messung nicht von dieser Seite her einzuschränken.
Unter Berücksichtigung dieser Einschränkungen bieten sich zwei Alternativen an:

- Die Nutzung eines (oder mehrerer) unbenutzter Pins
- Die Nutzung einer nicht anderweitig belegten Schnittstelle (die selbstverständlich ebenfalls mindestens einen verfügbaren Pin benötigt)

Die nachfolgend aufgeführten Messverfahren betrachten Beispiele beider Fälle. Darüber hinaus benötigen alle betrachteten Messverfahren zur Durchführung der Messungen noch weitere Hilfsmittel: ein Oszilloskop, ein Logic Analyzer oder ein Datenlogger wird als Messmittel, zur Datenaufzeichnung, -visualisierung und ggf. -speicherung benötigt.

Die Messverfahren

Aus den vorstehend erwähnten Einschränkungen hinsichtlich der Verfügbarkeit von Ressourcen resultieren eine Reihe von Möglichkeiten Varianten, Messdaten verfügbar zu machen:

A	„Bit Banging“ (abgekürzt: „BB“) Wenn ein unbenutzter Pin zur Verfügung steht, der kontaktiert werden kann, ist damit die einfachste aller Schnittstellen verfügbar. Sie kann 1 Bit Information bereitstellen. (Sogar etwas mehr als 1 Bit, wenn wir an die Ausgabe von Pulssequenzen zur Unterscheidung verschiedener Ereignisse denken.) Wer sich an die allerersten PIC Mikrocontroller erinnern kann: diese hatten keinen Hardware-UART, stattdessen wurden serielle Schnittstellen mit beeindruckenden Baudraten mittels Bit-Banging realisiert.
B	„Multi-Bit Banging“ (abgekürzt: „MB“) Wenn Sie mehr als einen einzigen unbenutzten Pin zur Verfügung haben, sind komplexere Ausgaben möglich: Sie können mehr als ein einziges Bit gleichzeitig ausgeben. Oder etwas messbar machen, das mehr als 1 Bit an Information enthält: z.B. den Zustand eines Zustandsautomaten. Da Pins in der Hardware üblicherweise als Byte-breite (oder noch breitere) „Ports“ organisiert sind, kann MB ziemlich kompliziert werden, wenn die ungenutzten Bits unterschiedlichen Ports angehören.
C	Nutzung eines Digital-Analog-Wanders[1] (kurz: „DAC“) In seltenen Fällen verfügt Ihr Prozessor über einen nicht benötigten DAC. Die Schwierigkeit bei der Nutzung ist einerseits die Verfügbarkeit an sich, andererseits die Anbindung an das Messmittel: Analogsignale benötigen evtl. noch einen Verstärker auf Ihrem Board, was die Nutzung auf Entwicklungssteuergeräte beschränkt.
D	Nutzung eines ungenutzten UART[2] Aktuelle Prozessoren sind mit reichlich digitalen Schnittstellen ausgestattet. Die Wahrscheinlichkeit, dass Sie über einen nicht anderweitig belegten UART verfügen, ist gar nicht so klein. In dieser Form ist ein Echtzeitbezug zu physikalischen Ereignissen allerdings schwer herzustellen, da der serielle Bitstrom nur mühsam auszuwerten ist.

E Die OLDA[3] („OnLine Data Analyzer“)

Die historische OLDA ist ein Werkzeug, das nur einem kleinen Benutzerkreis zur Verfügung stand und dementsprechend weitgehend unbekannt blieb.

Die aktuelle µOLDA kann als die Verbindung eines UART mit einem DAC betrachtet werden - ein seriell angeschlossener DAC.

Im Folgenden betrachten wir die Fälle A/B (eine Unterscheidung ist hier nur hinsichtlich der „Breite“ der Information sinnvoll) und C/E näher. (Soweit die notwendige Hardware verfügbar ist, ist C ein (schnellerer) Sonderfall von E.)

Bewertung der Leistungsfähigkeit der einzelnen Verfahren

An dieser Stelle wollen wir uns mit der Zusammenfassung der Ergebnisse der Bewertung der Verfahren begnügen - eine detailliertere Betrachtung und einige Beispielaufnahmen findet sich in den Folien Präsentation.

Betrachten wir also die Fragestellungen und die in Frage kommenden Messverfahren, so ergibt sich hinsichtlich der Eignung und Leistungsfähigkeit die folgende Tabelle 2:

Aufgabenstellung	**BB / MB**	**DAC / OLDA**
Laufzeitmessung einer einzelnen Task	+++ / +++	+++ / ++
Entwicklung und Validierung eines Zustandsautomaten	**o** / ++	+++ / ++
Allgemeine Algorithmenentwicklung	- / +	+++ / ++
Überwachung der Abarbeitung von Tasks und Prozessen	- / ++	+++ / ++
sin/cos Auswertung	- / **o**	+++ / +++
Änderung der Hardware-Konfiguration	**o** / +	+++ / ++

- nicht anwendbar o problematisch +, ++, +++ gut, besser, am Besten

Tabelle 2 Eignung der Messverfahren für die verschiedenen Aufgabenstellungen

Ohne den Ausführungen des Vortrags allzu sehr vorzugreifen: ein „lokal“ verfügbarer DAC ist einem seriell angebundenen DAC hinsichtlich der Geschwindigkeit natürlich immer überlegen - identische Qualität der Ausführung voraussetzend. Und BB/MB sind hinsichtlich der Geschwindigkeit nicht zu übertreffen, soweit die Geschwindigkeit der Signalausgabe nicht zur Reduzierung von Abstrahlungen „künstlich“ reduziert ist.

Was bitte ist eine OLDA?
Wie vorstehend erwähnt, steht OLDA für „OnLine Data Analyzer“. Der Begriff „Analyzer“ ist insofern irreführend, als die OLDA eigentlich keine Analysefunktionen inkorporiert. Ihre Funktion ist die eines DAC - also die Wandlung digitaler Werte in eine Analogspannung. Die eigentliche Analyse erfolgt dann mit Hilfe des angeschlossenen Messmittels (Oszilloskop, Datenlogger).
Die Target-Anbindung der „historischen“ OLDA erfolgte über den Adress- und Datenbus - eine Schnittstelle, die sich infolge der gestiegenen Taktfrequenzen als zunehmend unzuverlässig erwies. Darüber hinaus erwies sich auch die breite Anschlussleitung immer wieder als hinderlich. Letzten Endes war die historische OLDA - bei allem Nutzen - nicht mehr verwendbar.

Der Nachfolger: die µOLDA
Einige Jahre mussten wir auf eine OLDA verzichten und uns mit BB/MB und ähnlich beschränkten Verfahren behelfen, da kein adäquates Hilfsmittel zur Verfügung stand und auch hinsichtlich der Schnittstellen zum Target keine praktikablen Lösungen verfügbar waren.
Mit zunehmender Integrationsdichte und Leistungsfähigkeit der Mikrocontroller hat sich jetzt ein neuer Weg eröffnet, eine derartige Funktion zur Verfügung zu stellen: die serielle Anbindung des DACs - die µOLDA.
Serielle Schnittstellen sind mittlerweile schnell genug, um Baudraten > 1 MBit, in vielen Fällen auch > 10 MBit zur erreichen. Damit sind Datenraten von 100 kS/s bis in den Bereich von MS/s realisierbar. Hinzu kommt, dass in der Regel mehr serielle Schnittstellen (UART oder SPI) auf einem Target zur Verfügung stehen, als in der Anwendung tatsächlich genutzt werden.
Diese Erkenntnis führte zur Entwicklung der µOLDA, deren Komponenten und „Ökosystem“ in Bild 1 dargestellt sind.

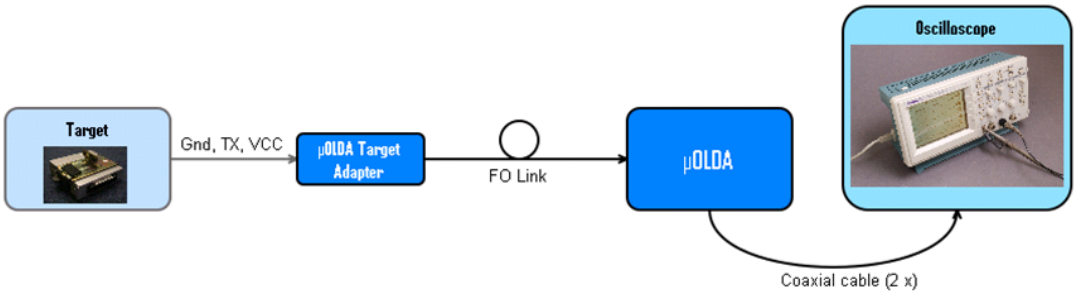

Bild 1 Die Komponenten eines µOLDA-Systems

Das µOLDA-System besteht aus 3 Komponenten:

1 dem sog. „Target-Adapter“ - im Grunde genommen ein einfacher elektro-optischer Wandler.

2 einer faseroptischen Verbindung. In Hinblick auf Einfachheit und Robustheit wurde das Versatile Link System gewählt, das mittels einer Plastikfaser (POF - Polymer Optical Fiber) Leitungslängen größer 30 m erlaubt und darüber hinaus hinsichtlich der Flexibilität der Leitung Vorteile gegenüber Kupferleitungen hat. Außerdem ist die Störfestigkeit optischer Verbindungen unübertroffen.

3 der eigentlichen µOLDA (in der der optoelektrische Wandler, die DACs und die notwendige Logik verbaut sind).

Um das System zu komplettieren, kommen noch Stromversorgungen, das bereits erwähnte Oszilloskop und eine Leitung zur Anbindung des Targetadapters an Ihr Target hinzu.

Hardware- und Software-Voraussetzungen
Die µOLDA ist also ein (zweikanaliger) DAC. Und da sie mit einer seriellen Verbindung arbeitet, benötigen Sie auf dem Target eine serielle Schnittstelle. Für das Target ist die Liste der notwendigen Hardware-Voraussetzungen erfreulich kurz. Sie benötigen:

- einen ungenutzten UART. Hilfreich, aber nicht Bedingung sind UARTs mit einem FIFO. Diese erlauben eine etwas einfachere Code-Instrumentierung, die allerdings zu Lasten der Latenzzeit der Daten gehen kann.
- einen freien Pin am Target, den Sie als TX Pin des UARTs nutzen können
- ein bisschen Strom (< 5 mA, 3,3 - 5 V) und
- eine kurze 3-adrige Verbindungsleitung (GNC, VCC, Signal) zur Verbindung von Target und Targetadapter.

Das war's.

Wie bitte? Sie haben keinen ungenutzten UART? In diesem Fall können Sie auch ein ungenutztes SPI-Interface verwenden. Es muss nur Datenformate > 10 Bit, „LSBit first" unterstützen, damit Sie den Datenstrom eines UART emulieren können.
Sie haben auch kein unbenutztes SPI-Interface? Dann müssen wir uns einmal in aller Ruhe unterhalten…

Und die Voraussetzungen auf Seiten der Software?

- Ein wenig Code, um den UART zu initialisieren.
- Die Code-Instrumentierung, um Werte in das Senderegister zu schreiben.
- Nur in Sonderfällen: noch ein bisschen mehr Code, um zu prüfen, ob das Senderegister den nächsten Wert aufnehmen kann.
- Für einige spezielle Anwendungen (z.B. Messung des Task Sequencing) benötigen Sie auch noch ein paar Bytes RAM.
  Bytes - nicht kBytes!

Was Sie nicht benötigen: Interrupts, ISRs und dergleichen.

Der Targetadapter - die Schnittstelle zu Ihrem Target
Einige Dinge werden sich voraussichtlich nie ändern:

- Die Baudraten von UARTs entsprechen maximal der halben Taktfrequenz des UART. (Üblicher sind Teiler /4, /8 und /16.)
  Wenn es um höchste Datenübertragungsraten geht, müssen wir uns um eine weitere Teilung des Takts durch einen Baudratengeneratoren nicht kümmern.
- Für SPI-Interfaces gilt im Grunde genommen dasselbe. Wobei der SPI-Port Ihres Prozessors möglicherweise schneller ist als sein UART. Dies ist von Fall zu Fall unterschiedlich.

Der Targetadapter benötigt für seine Funktion eigentlich mehr als die vorstehend spezifizierten 5 mA. Deshalb ist eine Potentialtrennung vorhanden, die die Stromauf-

nahme aus dem Target auf die erwähnten 5 mA reduzieren hilft. Um Ihr Target nicht mehr als notwendig zu belasten wird die isolierte Seite des Targetadapters von einer eigenen Stromversorgung gespeist,
Der Targetadapter ist ziemlich klein. Damit passt er möglicherweise noch in Ihr Gehäuse, was im Einzelfall der Robustheit zugute kommen kann. Sollten Sie ihn allerdings in Ihr Target integrieren wollen: Wir stellen Schaltung und BOM gerne zur Verfügung.

Die µOLDA - einige Eckdaten

Baudrate (UART oder SPI)		DC - 16 MBd
Serielles Übertragungsformat (Nutzdaten)		8 Bit (9 Bit)
Analogausgänge		2 (4 bei 9 Bit Format und Kaskadierung)
	Auflösung	7 Bit, glitchfrei
	Maximale Updaterate je Kanal	1,6 MS/s
	Adressierung der Ausgänge	„hart“ (d.h. jeder Kanal hat eine feste, nicht konfigurierbare Adresse) 8 Bit Frames : 1 Adressbit 9 Bit Frames : 2 Adressbits
	Unterstützte Zahlenformate	unsigned, signed
	Ausgangsspannung	unsigned 0 - 5 V signed -2,5 - 2,5 V

7 Bit Auflösung erscheinen auf den ersten Blick recht wenig. Wenn Sie sich allerdings ein wenig mit dem Datenblatt Ihres Oszilloskops beschäftigen, finden Sie dort mit einiger Wahrscheinlichkeit einen ENOB (Effective Number Of Bits) Wert von 7.5 - kaum besser als die 7 Bit der µOLDA.
Angesichts der Entscheidung für eine normale serielle Schnittstelle ermöglichen es 7 Datenbits auch, 1 oder 2 Adressbits (je nach Framelänge) zu übertragen und damit 2 bzw. 4 Variablen auszugeben. Und immer noch höchste Datenraten zu realisieren.
In Summe betrachten wir diese Designentscheidung als guten Kompromiss hinsichtlich Geschwindigkeit, einfacher Anwendung und Auflösung.

Zusammenfassung
Wie gezeigt, bietet die Ausgabe Digitaler Werte als Analogspannungen die Möglichkeit, Ereignisse der realen Welt und Software-Reaktionen zueinander in Bezug zu setzen. Soweit dafür ein DAC des Targets nicht zur Verfügung steht, besteht neuerdings die Möglichkeit, einen DAC seriell anzuschließen und damit derartige Messungen auch für Targets ohne (freien) DAC zu ermöglichen.
Und wie können Sie davon profitieren?
Wir kennen Ihre aktuellen Probleme nicht, insofern ist eine konkrete Antwort auf diese Frage nicht möglich. Aber wenn es Ihnen helfen würde, ein paar Daten der Software in Echtzeit messen zu können:

Bleiben Sie dran!

Referenzen

[1] FreeRTOS: Trace Hook Macros [More Advanced]
http://www.freertos.org/rtos-trace-macros.html

[2] “Debugging Embedded Systems with Minimal Resources”. Circuit Cellar 12.07.2016
http://circuitcellar.com/cc-blog/debugging-embedded-systems-with-minimal-resources/

[3] “On Real-Time Measurement Techniques“.
Ulrich Dreher, 24.02.2016, embedded world Conference 2016.

Autor

Ulrich Dreher verfügt über knapp 30 Jahre Erfahrung in der Softwareentwicklung, überwiegend für deeply embedded Systeme. Davon mittlerweile gut 20 Jahre im Bereich automotiver Entwicklungen.
Neben der Softwareentwicklung beschäftigt er sich auch mit Hardwareentwicklung, EMV-Optimierung und verwandten Themen, die im weitesten Sinne dem Bereich „Elektrotechnik“ zuzurechnen sind.

Kontakt

Email dreher@iss-stuttgart.de
Informationen zur µOLDA über OLDA-Info@web.de