Ristretto, per favore

Programmierung von MCs

Tam Hanna

Massimo Banzis Arduino öffnete die Welt der Mikrocontroller (MC) für Quereinsteiger: Teure Kommandogeräte, selbstentworfene Hauptplatinen und andere Spielereien waren nicht mehr notwendig. Auch wenn die Programmierbarkeit in C zunächst ein Vorteil war, erweist sich diese Sprache inzwischen als größtes Hemmnis – Java ist für Ein- und Umsteiger leichter zu handhaben. Ein Grund sich STM32Java näher anzusehen. Dabei handelt es sich um eine Kombination aus mehreren Produkten. IS2T liefern die Java-Runtime, die mit Keils C-Toolchain kompiliert wird. ST Microelectronics nutzt das Paket dann, um die Aufmerksamkeit von Entwicklern an sich zu binden. Kann STM32Java die Entwickler wie ein starker Kaffee aufputschen?

Aus Java mach C

Mikrocontroller üben auf Entwickler von Java-VMs einen immensen Reiz aus. Die extrem ressourcenschwachen Prozessoren unterscheiden sich von klassischen Workstations: 32-Bitter sind nur im High-End-Bereich verbreitet. Auch wenn es im Moment noch keine definitive Implementierungsweise gibt, ist im Laufe der letzten Jahre eine Vielzahl interessanter Konzepte aufgetaucht. Dieser Artikel möchte einige davon kurz vorstellen.

Googles *Boris Farber* empfiehlt Entwicklern auf Kongressen regelmäßig die Beschäftigung mit den Interna der Java-VM. Meyers und Downings Klassiker [MeDo97] beschreibt die VM als einen – im Großen und Ganzen – normalen Prozessor, der statt benannter Register numerische Registervariablen verwendet.

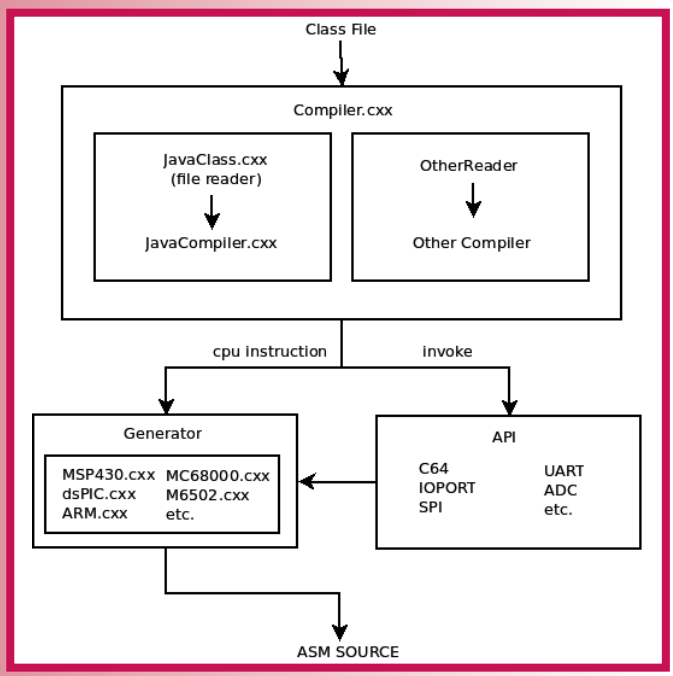

Abb. 1: Der Java-Compiler wandelt Java-Bytecode in Maschinencode um (Bildquelle: [Kohn15])

Da die JIT-Compiler Java-Bytecode seit Jahr und Tag zur Laufzeit in nativen Code der jeweiligen Host-CPU umwandeln, kam *Mike Kohn* im Jahr 2014 auf den Gedanken, diese Kompilation auf der Workstation durchzuführen und so nativen Code für Mikrocontroller zu erzeugen. Daraus entstand Java Grinder. Die von Mike Kohn bereitgestellte Abbildung 1 zeigt den grundlegenden Kompilationsprozess.

Portierungen auf neue Architekturen erfolgen durch das Umsetzen der in der Datei Generator.h definierten Befehle: Wer die Datei in einem Texteditor öffnet, findet sich mit einer Liste verschiedener Methoden konfrontiert:

```
class Generator : ... {
  public:
  ...
  virtual int insert_field_init_byte(char *name,
                                     int index, int value) = 0;
  virtual int insert_field_init_short(char *name,
                                     int index, int value) = 0;
 ...
```

Eine weitere interessante Komponente ist *naken_asm*. Es handelt sich dabei um ein weiteres Projekt aus dem Hause Kohn, das verschiedene Arten von Assemblercode in Bytecode umwandelt. Am Ende steht eine mehr oder weniger fertige .hex-Datei, die auf den jeweiligen Prozessrechner gesendet werden kann.

Laut Kohn ist das Projekt im Großen und Ganzen einsatzbereit: Neben einigen fehlenden Bytecodes gibt es im Moment auch keine Unterstützung für switch/case und keinen Garbage Collector. Letzteres ist aus Sicht eines Embedded-Programmierers nicht unbedingt schlecht: Statische Speicherallokation ist der Echtzeitfähigkeit eines Systems zuträglich.

VM in Klein ...

Java Grinder [Kohn15] ist insofern komplex, als es den Entwicklern der VM-Software immenses technisches Wissen abverlangt. Eine sparsam geschriebene virtuelle Maschine (VM) ist insofern günstiger, als sie – bei Befriedigung der Ressourcenansprüche – mit geringem Aufwand auf andere Controllerarchitekturen portiert werden kann.

Für Prozessoren der 8-Bit-Mikrocontroller-Familie Atmel AVR, die auch im Arduino steckt, gibt es mehrere Kandidaten, die ihrerseits unterschiedliche Vorgehensweisen zum Erreichen eines gemeinsamen Ziels befolgen. Dmitry Grinberg führt mit *UJC* (uJ Class, [uJ]) ein eigenes Klassenformat ein, das diverse Optimierungen für schnelleres Parsing mitbringt. Zum Deployment einer Klasse auf den Mikrocontroller muss diese durch ein als classCvt bezeichnetes Programm in dieses Format umgewandelt werden – das Laden normaler Klassen nimmt laut Grinberg bis zu dreifach mehr Zeit in Anspruch.

HaikuVM erledigt diese Konversion ebenfalls am Desktop: Die Klassen werden in C-Strukturen umgewandelt, die im Rahmen der Kompilation des Programms auf die MC-Unit wandern. Zur Laufzeit werden sie von einem reduzierten Interpreter abgearbeitet.

Aus Platzgründen können wir die VMs an dieser Stelle nicht weiter besprechen: Ein im JavaSPEKTRUM [Hanna15] erschienener Artikel beleuchtet die Nutzung von HaikuVM und Konsorten im Detail.

... und in Groß

ST Microelectronics versucht, seine ARM-MCUs (Advanced RISC Machine), die auch in einigen Arduinos stecken, seit län-

gerer Zeit aggressiv in den von Texas Instruments und Co. dominierten Markt zu schieben. Dabei entsinnen die Amerikaner immer wieder „leckeres“ Fast Food – Java-Support für die durchaus leistungsfähige 32-Bit-ARM-Architektur stand schon mehrmals auf dem Speisezettel.

Unter [STM32JE] lässt sich nach dem Anlegen eines kostenlosen Nutzeraccounts eine Demoversion von STM32Java herunterladen: Neben der .zip-Datei ist auch das PDF erforderlich, das einen Code für die Aktivierung der Keil-Compiler enthält. Führen Sie den in der .zip-Datei befindlichen Installationsassistenten aus, und erlauben Sie auch das Deployment von ST-Link und der dazugehörenden Treiber. Starten Sie STM32Java daraufhin – die Erstinitialisierung nimmt mitunter etwas Zeit in Anspruch.

Das französische Beratungsunternehmen IS2T besteht darauf, dass jede Testinstallation des Produkts von einem Mitarbeiter aktiviert wird. Es ist aus diesem Grund ratsam, die Aktivierung von STM32Java so schnell wie möglich durchzuführen. Nach der Freischaltung steht unter `https://stm32.microej.com/` eine .zip-Datei bereit, die nach Extrahierung zwei Dateien freigibt.

Laborspielzeug

Der Artikel basiert auf dem STM32F429DISCO-Evaluationsboard. Diese für rund 35 Euro erhältliche Platine ist aufgrund ihrer reichhaltigen Ausstattung auch für allgemeine Experimente mit ARM-MCUs empfehlenswert.

Laden Sie währenddessen die unter [ARMKEIL] bereitstehende MDK*.exe-Datei herunter. Sie enthält eine vollwertige C-Toolkette, die für das Kompilieren und Linken des von STM32Java erzeugten Codes zuständig ist. Nach der erfolgreichen Installation zeigt Keil einen Paketmanager an, in dem Sie Erweiterungsmodule für diverse Prozessrechner herunterladen können – für den in den folgenden Schritten verwendeten Controller ist die in Abbildung 2 gezeigte Konfiguration notwendig.

STM32Java unterstützt von Haus aus nur wenige Prozessoren. Klicken Sie auf „Help -> Welcome -> Manage Platforms“ und prüfen Sie, ob Ihr Zielboard aufgelistet ist. Ein Klick auf den auf der Willkommensseite befindlichen Button „Try Out Java“ öffnet ein Konfigurationsfenster, in dem Sie nach Auswahl der gewünschten Plattform mehr oder weniger Beispiele vorfinden.

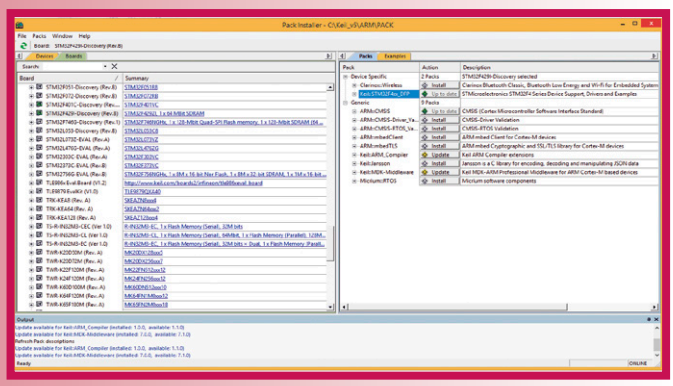

Abb. 2: Der Device-Support ist notwendig – Fehler beim Herunterladen der Paketdefinition sind irrelevant

Die von IS2T bereitgestellte Runtime ist auf die Realisierung von grafischen Benutzerschnittstellen optimiert: Die Arbeit mit nativer Hardware liegt dem Produkt nicht sonderlich und sollte – so sogar die offiziellen Empfehlungen (siehe [STM32Java-Webinar]) – eher durch C-Routinen erfolgen, die aus dem Java-Code heraus aufgerufen werden.

STM32Java-Beispiel: Rechteckwelle

Als erstes Beispiel dafür wollen wir ein Prgrämmchen realisieren, das eine Rechteckwelle auf einem Pin ausgibt. Dies ist in mehrerlei Hinsicht interessant: Neben der Erfassung der Startzeit können wir die VM so auch auf Jitter und andere Ärgernisse abklopfen.

STM32Java unterteilt den Projekterstellungs-Workflow in zwei Schritte:

- Nach der Erstellung einer Plattform mit Informationen über den Mikrocontroller folgt
- die Erzeugung des eigentlichen Codeprojekts, in dem die Applikationslogik unterzubringen ist.

Beginnen Sie mir dem Anklicken von „File -> New -> Java Platform“. Als Architektur wird „STM32JavaF4“ ausgewählt, im nächsten Schritt entscheiden wir uns für die Vorlage „STM32F429I-DISCO -> Full“. Nach dem Ausfüllen des Gerätenamens erzeugt STM32Java vier Unterprojekte.

Die -bsp-Projektmappe enthält das für Keil MicroVision benötigte Board Support Package. Im -configuration-Projekt findet sich eine .platform-Datei, die die zu kompilierende Umgebung beschreibt – öffnen Sie sie durch Doppelklick im grafischen Editor und klicken Sie den Link „Build Platform“ an, um die einige Minuten in Anspruch nehmende Kompilation anzustoßen.

Nach der erfolgreichen Erzeugung – die Achtkernworkstation des Autors genehmigte sich rund zwei Minuten – öffnen wir die Willkommensseite abermals. Nach dem Anklicken der Option „Try out Java Examples“ findet sich im JPF-Auswahldialog nun die neu erzeugte Umgebung. Wählen Sie sie aus und erzeugen Sie im nächsten Schritt ein Projekt auf Basis der Vorlage „EDC -> Hello World“.

Der in HelloWorld.java befindliche Code ist für uns im Moment nicht weiter interessant. Klicken Sie stattdessen auf „Run -> Run configurations“ und öffnen Sie die Rubrik „MikroEJ Application“. Ihr Inhalt präsentiert sich wie in Abbildung 3 gezeigt. Wählen Sie nun die Option „EmbJPF“ aus und klicken Sie auf „Run“ – der Lohn der Mühen ist ein Kompilat im „Executable and Linking Format“ ELF, das im Projektverzeichnis entsteht.

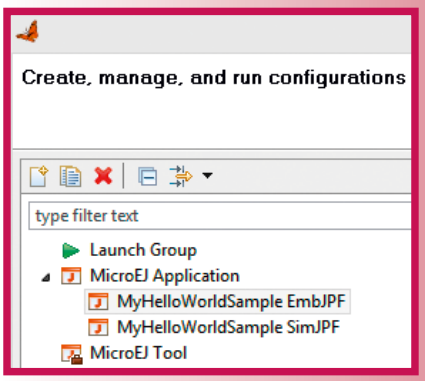

Abb. 3: MicroEJ-Applikationen können im Simulator oder auf echter Hardware leben

Zur Auslieferung des Gesamtkompilats ist bei STM-Mikrocontrollern eine vergleichsweise komplexe Toolkette erforderlich. Die Franzosen lösen dieses Problem durch Einspannen von Keil MicroVision. Im -bsp-Projekt findet sich unter „Project->MicroEJ->MDK-ARM->Project.uvproj“ ei-

ne Datei, die sich nach einem Doppelklick in „MicroVision“ öffnet.

Da STM32Java die Änderung des Projektformats in Keil 5 noch nicht nachvollzogen hat, erscheint beim Laden eine Fehlermeldung. Klicken Sie im Interesse der Kompatibilität die Option „Install Legacy Support“ an. Im daraufhin aufscheinenden Fenster wird der Button „Download Legacy Support for Cortex-M Devices“ angeklickt; die Installationsdatei schickt die benötigten Dateien auf Ihre Workstation.

Verbinden Sie die Evaluationsplatine sodann mit der Workstation. In Keil ist das Anklicken von „Project -> Build target und Flash -> Download“ notwendig. Wenn Ihr Evaluationsboard bisher mit der von STM ausgelieferten Software ausgestattet war, so äußert sich das erfolgreiche Deployment durch die Anzeige eines weißen Bildschirms.

C zu Diensten

Als nächste Aufgabe wollen wir uns der Signalerzeugung zuwenden. Dazu wird der Inhalt von HelloWorld.java folgendermaßen angepasst:

```
public class HelloWorld {
  public static native void setupGPIOs();
  public static native void emitWaveform();
  public static void main(String[] args) {
    setupGPIOs();
    while(1==1) {
      emitWaveform();
    }
  }
}
```

Aktualisieren Sie den Inhalt der Java-Objektdatei über Run und kompilieren Sie das Projekt. Keil beklagt sich im Rahmen des Linkings über die fehlenden nativen Methoden:

```
.\STM32F429i-DISCO\STM32429I-DISCO.axf: Error: L6218E: Undefined symbol
  Java_com_is2t_examples_edc_hello_HelloWorld_emitWaveform
   (referred from javaapp.o).\STM32F429i-DISCO\
   STM32429I-DISCO.axf: Error: L6218E: Undefined symbol Java_com_is2t_
     examples_edc_hello_HelloWorld_setupGPIOs
  (referred from javaapp.o).
```

Klicken Sie das Stammverzeichnis des Projekts in MikroVision rechts an und erstellen Sie eine neue Gruppe. Diese bekommt im nächsten Schritt durch einen weiteren Rechtsklick und die Option „Add New Item to Group“ eine .c-Datei eingeschrieben.

Als Erstes ist – Kenner von STMs Entwicklungsumgebungen grinsen an dieser Stelle – das Inkludieren einiger Headerdateien erforderlich:

```
#include „cpu.h“
#include „stm32f4xx.h“
#include „stm32f429i_discovery_sdram.h“
#include „stm32f4xx_fmc.h“
#include „stm32f429i_discovery.h“
```

Embedded-Prozessoren starten von Haus aus im „inerten“ Zustand. Unsere erste Aufgabe besteht darin, das GPIO-Modul mit dem Taktgeber zu verbinden und so seine Arbeitsfähigkeit zu sichern:

```
void Java_com_is2t_examples_edc_hello_HelloWorld_setupGPIOs() {
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
```

Im nächsten Schritt folgt die eigentliche Initialisierung des für die Ausgabe vorgesehenen Pins. Hierzu ist eine als **`GPIO_InitTypeDef`** bezeichnete Struktur notwendig, die per **`GPIO_INIT`** an einen der durch die GPIO-Konstanten eindeutig beschriebenen Ports weitergereicht wird:

```
  GPIO_InitTypeDef GPIO_InitStructure;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
}
```

STM stattet seine Mikrocontroller mit einer vergleichsweise komplexen GPIO-Programmierschnittstelle aus: Neben der Festlegung der Pin-Richtung muss der Programmierer auch festlegen, ob der betreffende Pin als Pish-Pull- oder als Open-Drain-Ausgang konfiguriert werden soll. Die Bestimmung der Geschwindigkeit dient der Adaptierung der Innenschaltung – je nach gewünschter Arbeitsfrequenz werden mehr oder weniger Kapazitäten zugeschaltet.

Das eigentliche Ein- und Ausschalten des Pins erfolgt sodann im Rahmen von emitWaveform. STMs Abstraktionsbibliothek stellt Entwicklern hier eine kleine Falle. Wer statt **`SetBits`** auf andere Methoden zum GPIO-Zugriff setzt, wird mit einem nicht funktionierenden Programm belohnt:

```
void Java_com_is2t_examples_edc_hello_HelloWorld_emitWaveform() {
  GPIO_SetBits(GPIOC, GPIO_Pin_0 );
  GPIO_ResetBits(GPIOC, GPIO_Pin_0 );
  GPIO_SetBits(GPIOC, GPIO_Pin_0 );
  GPIO_ResetBits(GPIOC, GPIO_Pin_0 );
  GPIO_SetBits(GPIOC, GPIO_Pin_0 );
  GPIO_ResetBits(GPIOC, GPIO_Pin_0 );
}
```

Damit ist die zweite Version des Projekts einsatzbereit. Die Analyse mit MDO (modulation domain analyzer) und Digitalspeicheroszillograf zeigt, dass die Aufrufe der nativen Methoden den primären Flaschenhals darstellen. Die Stabilität ist derweil im Großen und Ganzen brauchbar, wenn es auch zu JVM-bedingten Ausreißern kommt.

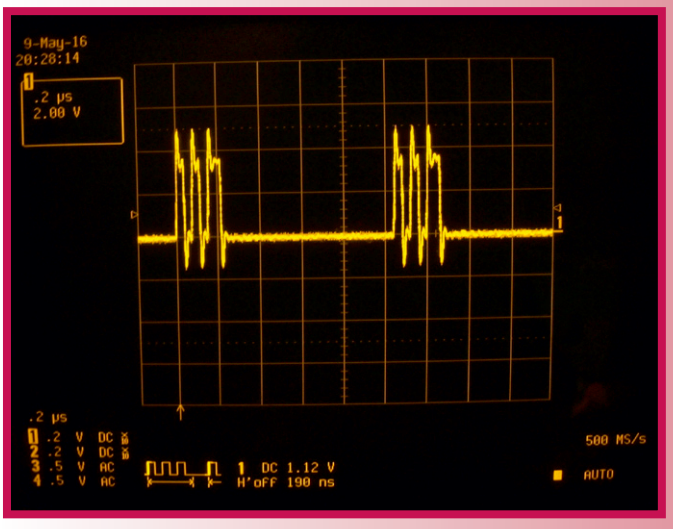

Abb. 4: JNI-Aufrufe nehmen Zeit in Anspruch

Zur Ermittlung der Startzeit der JVM wollen wir auf einen kleinen Trick zurückgreifen. Der STM32F429DISCO ist mit einem Knopf ausgestattet, der einen Neustart des Mikrocontrollers erzwingt. Das daraus entstehende Reset-Signal lässt sich durch ein auf der Unterseite der Hauptplatine befindliches Pin

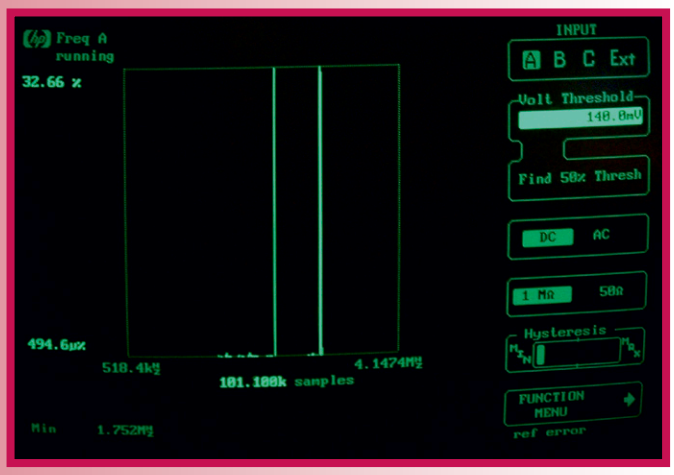

Abb. 5: Die Stabilität der generierten Wellenformen ist befriedigend

abgreifen und zum Triggern des LeCroy 9354AM nutzen. Anhand der zwischen Triggering und Auftauchen der Wellenform vergehenden Zeit lässt sich die Initialisierungsdauer der JVM abschätzen.

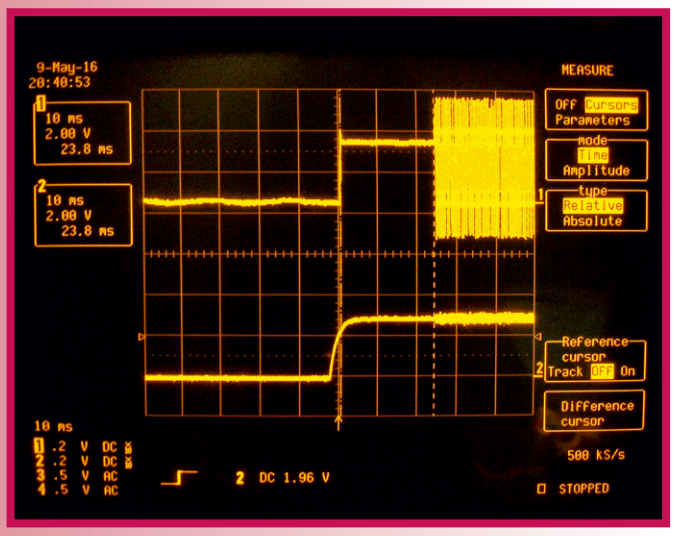

Abb. 6: Der Start der JVM nimmt etwas Zeit in Anspruch

Nutze die Macht des GUIs

STM bewerben die Grafikfähigkeiten ihrer Mikrocontroller immens: Es gibt kaum ein Evaluationsboard, das nicht mit einer mehr oder weniger großen LCD-Anzeige ausgestattet ist. Im Bereich Embedded Java stehen mit MicroUI [ESR002] und MWT [ESR011] zwei verschiedene Abstraktionsstufen zur Verfügung.

MicroUI ist im Grunde genommen ein Eventsystem, das zudem grafische Primitiva anbietet. MWT ist eine stark an AWT angelehnte Grafikbibliothek, auf deren Basis Entwickler mehr oder weniger komplexe Benutzerschnittstellen aus vorgefertigten Steuerelementen zusammenbauen können.

Da die beiden ESR-Standards in nicht allzu ferner Zukunft größere Bedeutung genießen werden, wollen wir sie in einer der nächsten Ausgaben von JavaSPEKTRUM mit einem eigenen Artikel würdigen. Für hier reicht es im Moment aus, auf die in STM32Java enthaltenen Beispiele hinzuweisen.

Mehr erfahren

STM war noch nie für umfangreiche Dokumentation bekannt. Unter `http://www.stm32java.com/resources/` finden sich in der Rubrik „Application Notes“ einige fertige Programme samt .pdf-Dateien, die einige Konzepte näher erklären.

Fazit

Wer sich heute mit Java im Embedded-Bereich befasst, steht einer Vielzahl fragmentierter Lösungen gegenüber: In der Praxis ist es in vielen Fällen zukunftsträchtiger und technisch sauberer, die Steuerung an einen dedizierten, in C zu programmierenden Echtzeitprozessor zu übertragen und diesen über einen seriellen Link anzusprechen.

An dieser Stelle sei ein Verweis auf den Kult-Comic „Golden Hammer“ [xkcd] erlaubt, er enthält mehr als nur ein Körnchen Wahrheit.

Links

[ARM] `https://de.wikipedia.org/wiki/ARM-Architektur`
[ARMKEIL] ARM Version 5.20 Evaluation Software Request, `https://www.keil.com/demo/eval/arm.htm`
[ELF] `https://de.wikipedia.org/wiki/Executable_and_Linking_Format`
[ESR002] MicroUI-1.4.1, Micro user interface Profile Specification, ESR Consortium, `http://www.e-s-r.net/download/specification/MICROUI-1.4-I.pdf`
[ESR011] MWT-1.0, Micro Widget Toolkit Profile Specification, ESR Consortium, `http://www.e-s-r.net/download/specification/MWT-1.0-D.pdf`
[HaikuVM] A Java VM for Arduino and other micros using the IeJOS runtime, `http://haiku-vm.sourceforge.net/`
[Hanna15] T. Hanna, HaikuVM: Eine virtuelle Java-Maschine für Mikrocontroller, in: JavaSPEKTRUM, 05/2015
[Kohn15] M. Kohn, Java Grinder, 20.11.2015, `https://www.mikekohn.net/micro/java_grinder.php`
[MeDo97] J. Meyer , T. Downing, Java Virtual Machine, O'Reilly, 1997
[STM32] `https://en.wikipedia.org/wiki/STM32`
[STM32JavaWebinar] `https://my.st.com/public/STe2ecommunities/mcu/Lists/STM32Java/Attachments/1/STM32Java_Webinar.pdf`
[STM32JE] STM32Java SDK free trialversion, STMicroelectronics, `http://www.stm32java.com/free-trial/`
[uJ] D. Grinberg, `http://dmitry.gr/index.php?r=05.Projects&proj=12. uJ - a micro JVM`
[xkcd] `https://www.explainxkcd.com/wiki/index.php/801:_Golden_Hammer`

Tam Hanna ist Gründer der in Pressburg (Bratislava, Slowakei) ansässigen Tamoggemon Holding k.s. Er beschäftigt sich mit der Programmierung und Anwendung von Mobilcomputersystemen.
E-Mail: tamhan@tamoggemon.com