Ralf
Jesse

ARM Cortex-M3 Mikrocontroller

Einstieg und Praxis

Kapitel 1

Orientierung

In diesem Buch wird eine *Entwicklungsumgebung* (*Entwicklungsplattform* = *IDE*, *Integrated Development Environment*) für Cortex-M3-Mikrocontroller eingesetzt, die auf der frei verfügbaren Eclipse-Plattform basiert. Doch diese Plattform ist bei Weitem nicht die einzige, die für die Programmierung von Cortex-M3-Mikrocontrollern herangezogen werden kann. Im Gegenteil: Die Zahl der verfügbaren Plattformen ist nahezu unüberschaubar! Je nach Anspruch (und finanziellen Möglichkeiten) haben Sie die freie Wahl zwischen

- kommerziellen IDEs, deren Anschaffung aber unter Umständen mit erheblichen Kosten verbunden ist,
- herstellergebundenen, aber kostenlosen, IDEs und
- freien Entwicklungsumgebungen.

In den folgenden Abschnitten dieses Kapitels gebe ich Ihnen einen kleinen Überblick über die am Markt verfügbaren Entwicklungsumgebungen. Dieser Überblick erhebt bei Weitem keinen Anspruch auf Vollständigkeit! Es werden aber die meiner Meinung nach am weitesten verbreiteten Vertreter dieser drei Gruppen kurz vorgestellt.

1.1 Kommerzielle IDEs

Viele Entwickler von Mikrocontrollerschaltungen (hier sind besonders die im industriellen Umfeld tätigen Unternehmen gemeint) legen großen Wert auf einen guten Support durch die Hersteller von Entwicklungsumgebungen. In diesem Umfeld spielen Lizenzkosten, die bis zu 5.000 Euro betragen können, eine eher untergeordnete Rolle, zumal die Hersteller auch Mehrbenutzerlizenzen mit großzügigen Rabatten anbieten, was den Durchschnittspreis pro Lizenz beim Kauf mehrerer Lizenzen erheblich reduzieren kann. In den meisten Fällen bieten die kommerziellen Anbieter auch Evaluierungsversionen ihrer Software an. Diese sind nach erfolgter Registrierung in der Regel kostenlos verfügbar und entsprechen mit Einschränkungen häufig den Vollversionen. Diese Einschränkungen betreffen beispielsweise den Nutzungszeitraum oder die Größe des erzeugten Codes. Fast niemals dürfen diese Softwareversionen produktiv, das heißt kommerziell, eingesetzt werden. Auch die Nutzung für Ausbildungszwecke ist hiermit häufig untersagt, da beispielsweise Lehrer durch das Erteilen des Unterrichts Geld

verdienen: Diese Arbeit wird von den meisten Herstellern durch sogenannte Educational Licenses zu Vorzugspreisen unterstützt. Über den Unterricht hinausgehende kommerzielle Nutzung dieser Lizenzen ist aber auch dann untersagt. Einige Beispiele für kommerzielle Entwicklungsumgebungen sowie die Links zu den Herstellerseiten finden Sie in den folgenden Abschnitten.

1.1.1 Keil µVision

Die Firma Keil, früher ein eigenständiges Unternehmen, wurde inzwischen von der Firma ARM übernommen. Bei *Keil µVision* handelt es sich um eine IDE, die von Haus aus die Chips unterschiedlicher Hersteller direkt unterstützt. Nach der Entscheidung für einen Chip können Sie die für Ihr Projekt benötigten Komponenten eines Chips durch einfaches Anklicken in das Projekt übernehmen, wobei die Initialisierung und Konfiguration dieser Komponenten über eine bequem zu nutzende Benutzeroberfläche durchgeführt werden kann. Als Folge dieser Auswahl »baut« die IDE dann das Gerüst Ihres Projekts auf, sodass Sie sich »nur noch« um die eigentlichen Funktionalitäten kümmern müssen. Für jeden von dieser IDE unterstützten Chip steht unmittelbar eine ausführliche Dokumentation zur Verfügung: Die häufig mühsame Suche nach der korrekten Dokumentation entfällt somit. Eine Vielzahl von Beispielanwendungen erleichtert die Entwicklung eigener Projekte. Abgerundet wird die Software durch eine Debugging-Software, die es ermöglicht, den Zustand des Mikrocontrollers bis hinein in die internen Register der CPU zu kontrollieren. Weitere Informationen finden Sie unter `http://www.keil.com`.

1.1.2 IAR Workbench

Genauso mächtig und umfangreich ist die Workbench der Firma *IAR*, was bei professionellen Entwicklern ebenfalls zu einer großen Beliebtheit dieser IDE geführt hat. Neben der Unterstützung der gesamten Produktpalette von ARM-Bausteinen (angefangen beim ARM7TDMI bis hin zum Cortex-A 15) gehört auch hier eine vollständige Dokumentation aller unterstützten Bausteine. Mehr als 3.100 Anwendungsbeispiele lassen das Herz jedes Entwicklers höher schlagen. Die Bedienung der Workbench ist vergleichbar strukturiert wie bei Keil und ebenso einfach und intuitiv möglich. Dass die IAR Workbench ebenfalls über eine integrierte Debugger-Lösung verfügt, versteht sich von selbst. Weitere Informationen finden Sie unter `http://www.iar.com`.

1.1.3 Sourcery Codebench

Diese Software von der Firma *Mentor* basiert auf umfangreichen Erweiterungen für Eclipse unter Einsatz der GNU Compiler Collection. Die Sourcery Codebench gibt es für sehr viele verschiedene Zielplattformen: Für unseren Anwendungsfall

wäre die Version *ARM EABI* die richtige Variante. Der Einsatz kostenloser Basis-Werkzeuge, wie der genannten Eclipse-IDE und der GNU Compiler Collection, ermöglicht es der Firma Mentor, die Codebench etwas preisgünstiger anzubieten. So kostet die Professional-Version als Einzelplatzlizenz 2.800 US-Dollar (Nettopreis). Es existiert auch eine kostenlose Version: Sourcery Codebench lite. Hierbei handelt es sich aber um eine kommandozeilenbasierte Version, sodass der Komfort einer grafischen Entwicklungsumgebung (z.B. Eclipse oder NetBeans) nur durch umfangreiche Konfigurationsarbeiten erreicht werden kann. Weitere Informationen finden Sie unter `http://www.mentor.com`.

1.1.4 Atollic TrueSTUDIO

Für ca. 1.000 Euro ist das ebenfalls Eclipse-basierte *Atollic TrueSTUDIO* erhältlich. Hierbei handelt es sich um die Professional-Version, die kostenlose Version Atollic TrueSTUDIO Lite ist ebenfalls verfügbar. Neben der Unterstützung von mehr als 1.300 ARM-basierten Mikrocontrollern werden noch ungefähr 1.000 Beispielanwendungen mitgeliefert, sodass auch hier mehr als ausreichend Vorlagen für die Umsetzung eigener Projektideen genutzt werden können.

Es existieren noch weitere kommerzielle Entwicklungsumgebungen. Die vier oben genannten sind aber die – meiner Einschätzung nach – am weitesten verbreiteten. Dies äußert sich auch bei Recherchen im Internet durch die Vielzahl von privat geführten Foren, in denen umfangreicher Support geleistet wird.

1.1.5 CrossWorks for ARM

CrossWorks for ARM ist ebenfalls eine auf Eclipse basierende kommerzielle Entwicklungsplattform, die von der Firma *Rowley Associates* entwickelt wird. Wie die bereits vorgenannten kommerziellen Entwicklungsumgebungen, liefert auch CrossWorks for ARM sozusagen »alles aus einem Guss«. Rowley Associates bietet die IDE in drei Varianten an, die sich nur im Lizenzmodell unterscheiden. Hierbei handelt es sich um

- eine persönliche Lizenz, die sich an Hobby-Programmierer und Schüler bzw. Studenten wendet. Wie es der Name bereits andeutet, ist diese Lizenz an eine Person (und nicht an einen Computer, der von mehreren Personen genutzt wird) gebunden. Bei einem Preis von 150 US-Dollar ist diese Version sehr günstig, wobei die kommerzielle Nutzung ausgeschlossen ist.
- eine Lizenz zu Ausbildungszwecken für Lehrer und Dozenten. Diese kostet 300 US-Dollar und darf ausschließlich für den Zweck der Ausbildung genutzt werden. Im Gegensatz zur persönlichen Version ist diese Lizenz an einen Computer gebunden, der von verschiedenen Personen genutzt werden darf. Die Entwicklung darüber hinausgehender kommerzieller Anwendungen ist nicht zulässig.

- eine Profi-Lizenz zu einem Preis von 1.500 US-Dollar. Auch diese Lizenz ist wieder personengebunden. Die kommerzielle Nutzung ist mit dieser Version gestattet.

1.2 Herstellergebundene IDEs

Viele Hersteller von Mikrocontrollern haben erkannt, dass sie den Nutzen ihrer Produkte für die Anwender erheblich steigern können, wenn sie ihren Kunden eine eigene kostenlose Entwicklungsumgebung anbieten. Natürlich sind diese nicht so universell einsetzbar wie die kommerziellen Varianten: Sie beschränken sich im Regelfall auf die eigene Produktpalette, was ich allerdings für absolut legitim halte. Eine kleine Auswahl der bekanntesten herstellergebundenen Entwicklungsumgebungen finden Sie in den folgenden Abschnitten.

1.2.1 ATMEL Studio 6

Da das von mir gewählte Entwicklungsboard den ATMEL-Chip SAM3S4B einsetzt, beginne ich bei der Beschreibung auch mit dem von ATMEL bereitgestellten *ATMEL Studio 6*. Bei dieser IDE handelt es sich um eine Entwicklungsumgebung, die sehr übersichtlich aufgebaut ist und eine besonders komfortable Fehlersuche (Debuggen) ermöglicht: Mit dieser IDE kann man während der Laufzeit »in den Prozessor« hineinschauen und sehen, was sich dort gerade abspielt. Das ATMEL Studio 6 basiert auf *Microsofts Visual Studio*, und bietet somit alle Möglichkeiten, die viele Entwickler im Microsoft-Umfeld in anderen Situationen kennen- und schätzen gelernt haben. Der möglicherweise einzige Nachteil im Vergleich zu den meisten anderen IDEs ist, dass sie ausschließlich für Windows-Betriebssysteme von Microsoft verfügbar ist. Bei einem Marktanteil von ca. 90% muss dieser Nachteil aber nicht gravierend sein. Wer mit einem Windows-Betriebssystem arbeitet und einen ATMEL-Chip einsetzt, sollte diese Entwicklungsumgebung auf jeden Fall in Betracht ziehen. Dies gilt umso mehr, als man nicht zwingend auf ein Evaluierungsboard von ATMEL angewiesen ist und beispielsweise selbst entwickelte Boards einsetzen kann. Da bereits ein Browser in die IDE integriert ist, kann man nahtlos und ohne Einsatz weiterer Programme auf Internetressourcen zugreifen. Der Link zu dieser IDE lautet `http://www.ATMEL.com/tools/ATMELstudio.aspx`.

1.2.2 Texas Instruments StellarisWare

StellarisWare ist eine Entwicklungsumgebung, die von der Firma *Texas Instruments* für die Programmierung sämtlicher Stellaris-MCUs (MCU = Microcontroller Unit) kostenlos und ohne bekannte Einschränkungen zur Verfügung gestellt wird. Auch hier stehen sämtliche denkbaren und für den effizienten Einsatz notwendigen Hilfsmittel einschließlich der Dokumentation bereit. Zum Einsatz dieser Ent-

wicklungsumgebung kann ich aber keine weiteren Informationen liefern, da in diesem Buch die MCU von ATMEL verwendet wird.

1.2.3 STMicroelectronics STVD

Nicht zuletzt durch den Aspekt, dass *STMicroelectronics* als einer der ersten Anbieter von Cortex-MCUs auf dem Markt aktiv wurde, erfreuen sich deren Mikrocontroller großer Verbreitung und Beliebtheit. Auch viele Hilfestellungen, die in den einschlägigen Internet-Foren zu finden sind, sind auf MCUs dieses Herstellers optimiert. Mit *STVD*, dem ST Visual Developer, bietet STMicroelectronics ebenfalls eine Entwicklungsumgebung an, die – teilweise und in Abhängigkeit von der gewählten MCU – nur eingeschränkt genutzt werden kann. Ich habe mich selber nicht mit Controllern von STMicroelectronics beschäftigt und muss daher für weitere Informationen auf Internet-Foren verweisen. Hier tut sich meiner Meinung nach die deutschsprachige Webseite `http://www.mikrocontroller.net` besonders positiv hervor.

1.3 Freie IDEs

Neben den vielen kommerziellen bzw. herstellergebundenen IDEs sind in den vergangenen Jahren vermehrt unabhängige Entwicklungsumgebungen auf den Markt gelangt, die kostenlos und uneingeschränkt nutzbar sind. Nachfolgend werden nur einige wenige dieser IDEs kurz angerissen: Der Markt ist einfach zu groß, um einen vollständigen Überblick liefern zu können.

1.3.1 CooCox CoIDE

Bei der *CooCox CoIDE* handelt es sich um eine auf Eclipse basierende kostenlose und gut gewartete Entwicklungsumgebung. Bei der Erstellung eines neuen Projekts erfolgt die Basis-Konfiguration wahlweise über den eingesetzten Mikrocontroller oder über die Auswahl eines der (leider bisher noch wenigen) Evaluierungsboards der großen Chip-Hersteller. Im nachfolgenden Dialog können weitere erforderliche Einstellungen für das geplante Projekt vorgenommen werden. Im Wesentlichen handelt es sich hierbei um die Aktivierung von CMSIS, die Verwendung der C-Standardbibliotheken sowie die Entscheidung, den Startup-Code für das Board automatisch oder manuell zu erzeugen. Ein kleiner Nachteil dieser IDE liegt darin, dass sie noch sehr jung ist und somit nur eine relativ geringe Auswahl an Mikrocontrollern direkt unterstützt. Falls diese IDE für Sie in Betracht kommt, folgen Sie bitte dem Link `http://www.coocox.org/`.

1.3.2 NetBeans for C Developers

Ähnlich wie Eclipse startete *NetBeans* als reine Entwicklungsumgebung für die Programmierung in der Programmiersprache *Java*. Aus dem ursprünglich von

tschechischen Studenten entwickelten Xelfi (der Name hatte nicht nur zufälligerweise eine gewisse Ähnlichkeit mit dem Pascal-Entwicklungssystem Delphi von der Firma Borland) wurde durch eine Entscheidung von Sun Microsystems Inc. im Jahre 1999 durch die Verknüpfung mit *Forte for Java* schließlich NetBeans. Nun würde uns (es sei denn, wir entwickelten unsere Software in Java) NetBeans an dieser Stelle nicht weiter interessieren, wenn Freiwillige nicht durch ihre Beiträge zu NetBeans schließlich auch die Programmierung in C und C++ implementiert hätten. Was die generelle Unterstützung der Programmiersprachen C und C++ betrifft, ist NetBeans absolut gleichwertig zu Eclipse; die Unterstützung beim sogenannten *Cross Compiling* »hinkt« meiner Ansicht nach Eclipse aber etwas hinterher. Auf der Webseite `http://www.netbeans.org` finden Sie weitere Informationen. Da NetBeans vollständig in der Programmiersprache Java geschrieben wurde, ist diese Entwicklungsplattform für alle Betriebssysteme verfügbar, für die eine *Java-Laufzeitumgebung*, das sogenannte *Runtime Environment*, vorhanden ist.

1.3.3 Code::Blocks

Code::Blocks ist eine kostenlose IDE (Open Source, GPLv3), die sehr gut gepflegt und inzwischen in Version 12.11 vorliegt. Sie ist in C++ geschrieben und für alle wichtigen Betriebssysteme verfügbar. Sie ist durch Plug-ins erweiterbar und unterstützt eine Vielzahl von Compilern. Mehr Informationen finden Sie unter `http://www.codeblocks.org`.

1.3.4 emIDE

emIDE verwendet das gleiche Logo wie Code::Blocks: Auf den ersten Blick hat sich mir aber nicht erschlossen, worin sich Code::Blocks und emIDE unterscheiden. Wenn Sie mehr erfahren wollen: Der Link lautet `http://www.emide.org`.

1.3.5 Eclipse für C/C++-Entwickler

Last, but not least, soll die Entwicklungsplattform *Eclipse* vorgestellt werden, die bei vielen Entwicklern im ARM-Segment große Beliebtheit und entsprechend große Unterstützung erfährt. Eclipse wurde ursprünglich von der Firma International Business Machines oder kurz IBM entwickelt. Allerdings hieß das System damals noch nicht Eclipse: Der Kern des neuen Systems basierte zumindest teilweise auf IBMs Visual Age. IBM startete die Entwicklung vor dem geschäftlichen Hintergrund, seinen Marktanteil an IDEs auszubauen. Als zweiter Aspekt wird auch gerne genannt, dass IBM die sogenannte »open community« besser unterstützen wollte. In Zusammenarbeit mit weiteren in der Softwarewelt sehr bekannten Unternehmen, wie z.B. *Borland*, *QNX Software Systems*, *Red Hat* oder *Rational Software*, entstand schließlich die Entwicklungsumgebung, die heute als Eclipse bekannt ist. Der für dieses Buch besonders wichtige Teil ist der von der Firma QNX beigetragene Teil: die Entwicklungsumgebung, die die Programmierung in C bzw. C++ mit Eclipse erst möglich gemacht hat.

Im weiteren Verlauf dieses Kapitels werde ich mich mit der Installation von Eclipse für die C/C++-Entwicklung und den erforderlichen Vorbereitungen befassen. Die Wahl fiel auf Eclipse, weil diese Plattform weltweit von sehr vielen Entwicklern eingesetzt wird, was einen besonders guten Support verspricht. Dies äußert sich nicht zuletzt in IDE-Erweiterungen, sogenannte Plug-ins, mit denen die Entwicklung von ARM-basierten Projekten erheblich vereinfacht wird.

Hinweis

In Abschnitt 1.3.2 wurde der Begriff *Cross-Compiling* eingeführt. Hierunter versteht man die Entwicklung einer Software für eine Zielplattform, die sich von der Entwicklerplattform unterscheidet. Alle oben genannten Entwicklungsumgebungen führen demnach dieses Cross-Compiling durch, da die Entwicklerplattform in den meisten Fällen ein Intel- oder AMD-basiertes System ist, während die Zielplattform in diesem Buch eine Cortex-M3-Umgebung darstellt, die binär nicht kompatibel zueinander sind.

1.4 Vorbereitende Arbeiten

Bevor Sie mit der Programmierung eigener Projekte für den Cortex-M3-Mikrocontroller von ATMEL beginnen können, müssen Sie einige vorbereitende Arbeiten erledigen:

1. Sie müssen die Hardware bestellen, also das Evaluierungsboard bzw. Development Kit, wie es auch häufig genannt wird (für dieses Buch habe ich das *Olimex-Board SAM3-P256* gewählt).
2. Nicht zwingend erforderlich, aber dennoch empfehlenswert, ist die Beschaffung eines In-Circuit-Emulators, der gleichzeitig die Schnittstelle zum Debugger darstellt.
3. Der nächste Schritt gilt dem Download und der Installation der Entwicklungsumgebung.
4. Auch die Compiler-Toolchain muss beschafft und installiert werden.

Hinweis

Obwohl in diesem Buch das Entwicklungs-Board Olimex SAM3-P256 mit dem ATMEL-Chip AT SAM3S4B eingesetzt wird, sollten die folgenden Erläuterungen in diesem Kapitel auf sämtliche Entwicklungsboards und Mikrocontroller-Chips anderer Hersteller 1:1 übertragbar sein. Sie müssen dann anstelle der C-Quelltexte, die von Olimex für das SAM3-P256 und von ATMEL für den ATSAM3S4B bereitgestellt werden, die entsprechenden Quelltexte der jeweiligen Hersteller verwenden.

1.4.1 Hardware

Auf die Lieferzeit des Entwicklungsboards haben Sie keinen Einfluss: Sie wissen nicht, ob das Board zum gewünschten Zeitpunkt lieferbar ist. Selbst wenn das Board bei dem Händler Ihres Vertrauens auf Lager ist, sind Sie immer noch von der Auftragsbearbeitung und der Schnelligkeit des Zustelldienstes abhängig. Daher sollte die Beschaffung dieser Komponenten der erste Schritt sein.

Das Olimex-Board SAM3-P256

Bei dem Olimex-Board SAM3-P256 handelt es sich um eine ausgereifte Entwicklungsplatine zur Programmierung des Cortex-M3-Chips *ATSAM3S4B* von der Firma ATMEL. Dieses Board ist trotz seines vergleichsweise günstigen Beschaffungspreises nahezu universell einsetzbar: Neben zwei RS-232-Anschlüssen, einem USB-Anschluss, zwei Tastern, zwei LEDs (und einer weiteren LED zur Anzeige der Betriebsbereitschaft) bietet es ein Potenziometer (Trimmer) und ein als TH1 bezeichnetes temperaturabhängiges Bauelement, einen sogenannten Thermistor. Die beiden letztgenannten Bauelemente (Trimmer und Thermistor) sind unmittelbar an den AD-Wandler des SAM3S4BA angeschlossen, sodass die wesentlichen Features für Experimente abgedeckt sind. Darüber hinaus sind über 40 sogenannte General-Purpose-Anschlüsse auf ein Erweiterungsfeld hinausgeführt, auf dem Sie zusätzliche Hardware-Komponenten montieren können. Der auf dem Board eingesetzte Spannungsregler kann mit einem Strom von bis zu 800 mA belastet werden.

Vorsicht

Das Board kann generell über einen USB-Anschluss des Entwicklungscomputers elektrisch versorgt werden. Ein solcher Anschluss darf aber gemäß den Spezifikationen nur mit maximal 500 mA belastet werden. Sollte der Strombedarf Ihrer Schaltung diesen Maximalwert überschreiten, müssen Sie eine externe Stromversorgung, z.B. ein einfaches stabilisiertes Steckernetzteil, einsetzen.

Zusätzlich verfügt das Board über einen sogenannten *Standard-JTAG*-Anschluss, an den beispielsweise der weiter oben erwähnte SAM-ICE angeschlossen werden kann. JTAG ist die Abkürzung von *Joint Test Action Group* und bezeichnet den Standard *IEEE 1149.1*, der eine Sammlung von Testverfahren für den Debug-Prozess innerhalb der Schaltung beschreibt (daher auch die Bezeichnung In-Circuit-Emulation bzw. der Abkürzung ICE).

Trotz aller Universalität gibt es für Cortex-M3-Chips auch Anwendungen, die mit diesem Board nicht unmittelbar möglich sind. Hierbei kann es sich beispielsweise um den Einsatz von Ethernet-Adaptern, LC-Displays, Relais, GPS-Empfängern

oder anderen Komponenten handeln. Um den Preis niedrig zu halten, hat sich Olimex entschlossen, auf solche Komponenten zu verzichten, da diese nicht von jedem Anwender genutzt werden. Um sich aber die Option offenzuhalten, solche Komponenten nachrüsten zu können, besitzt dieses Board einen sogenannten *UEXT*-Anschluss, an den diese Komponenten angeschlossen werden können. UEXT ist hier die Abkürzung von *Universal EXTension*. Dieser UEXT-Anschluss enthält neben einer 3.3-Volt-Spannungsversorgung drei serielle Schnittstellen: jeweils einen asynchronen Bus, einen I²C-Bus und einen SPI-Anschluss. Auf diese Weise ließe sich beispielsweise auch ein sogenanntes MIDI-Interface, was in der Welt der Musiker von großer Wichtigkeit ist, realisieren.

In Anhang B finden Sie in Abschnitt B.1.1 Informationen zur Beschaffung dieses Entwicklungsboards. Um nicht zu viel Zeit zu verlieren, sollten Sie sich nun daran machen, das Entwicklungsboard zu bestellen.

In-Circuit-Emulatoren

Es existieren verschiedene Möglichkeiten, Software in den SAM3S4BA zu übertragen und auch zu debuggen, die sich im Anschaffungspreis, aber auch in ihrer Leistungsfähigkeit, teilweise erheblich voneinander unterscheiden. Ich verwende in diesem Buch den ATMEL SAM-ICE, will aber einige Alternativen zumindest vorstellen.

ATMEL SAM-ICE

Wenn hier der In-Circuit-Emulator ATMEL SAM-ICE genannt wird, so sollte darauf hingewiesen werden, dass es sich um ein Produkt der deutschen Firma SEGGER Microcontroller GmbH & Co. KG handelt, das von ATMEL lizenziert wurde. Aufgrund technischer Maßnahmen kann dieser ICE ausschließlich für Mikrocontroller der Firma ATMEL eingesetzt werden. Natürlich kann zu einem höheren Beschaffungspreis auch die Vollversion dieses ICE eingesetzt werden: Die Beschränkung auf ATMEL-Produkte entfällt dann. Das Vollprodukt mit der Bezeichnung J-Link ist in unterschiedlich leistungsfähigen Ausführungen erhältlich.

Hinweis

Die Beschaffung dieses nicht ganz preiswerten ICE ist nicht zwingend erforderlich. Sie ist aber empfehlenswert, um den AT SAM3S4BA (und andere Mikrocontroller von ATMEL) effizient programmieren zu können.

Für den im Vergleich zu anderen ICE relativ hohen Anschaffungspreis von ca. 100 bis 120 Euro (für die ATMEL-Variante, der Original-JLink von Segger ist erheblich teurer) erhält man ein problemlos out-of-the-box einsetzbares Werkzeug.

Hinweis

Inzwischen vertreibt Segger auch eine sogenannte Education-Version des ICE, für den die Beschränkung auf ATMEL-Controller nicht zutrifft. Der Preis ist sehr günstig und liegt bei ca. 50 Euro.

Wiggler

Der Wiggler ist die vermutlich preiswerteste Lösung, die Sie finden werden. Das Problem liegt aber darin, dass dieses Low-Cost-Interface (ca. 10 Euro) einen Parallel-Port am Entwicklungssystem benötigt, der auf zumindest halbwegs aktuellen Computern nicht mehr vorhanden ist und durch USB-Ports ersetzt wurde.

ICprog OpenOCD

Hierbei handelt es sich um einen Open-Source-Adapter, der PC-seitig an einen USB-Port angeschlossen wird und auf der Mikrocontroller-Seite über einen Standard-JTAG-Adapter verfügt. Als Software wird hier OpenOCD verwendet. Der Vorteil des günstigen Preises relativiert sich etwas, wenn man die aufwendigen Konfigurationsarbeiten, die für jedes Projekt neu durchgeführt werden müssen, berücksichtigt. Ob da ein Anschaffungspreis von ca. 30 Euro wesentlich günstiger im Vergleich zum ATMEL SAM-ICE ist, muss jeder Nutzer für sich selbst entscheiden. In diesem Buch wird der ICprog OpenOCD nicht eingesetzt.

Informationen zu den Beschaffungsquellen finden Sie in Abschnitt B.1.3.

»Debugging light« – Eine weitere Debugging-Methode

Es existiert noch eine weitere Debugging-Methode, die sehr einfach und kostengünstig einsetzbar ist. Im günstigsten Fall benötigen Sie nur ein einfaches RS232-Kabel, ein sogenanntes *Nullmodemkabel*, das auf der einen Seite an einen möglicherweise noch verfügbaren seriellen Port des Entwicklungssystems und auf der anderen Seite an die Schnittstelle RS232_0 des Olimex-Boards angeschlossen wird. Ein solches Kabel benutzt nur drei Drähte der insgesamt neun Anschlüsse der verwendeten Sub-D-Buchsen: RxD, TxD und Masse, wobei mit RxD die Empfangsleitung und mit TxD die Sendeleitung bezeichnet werden. RxD (Anschluss 2) ist auf der anderen Seite des Kabels mit dem TxD-Anschluss verbunden und TxD entsprechend mit RxD auf der anderen Seite. Man sagt hierzu auch, dass die Anschlüsse gekreuzt sind. Ein solches Kabel ist ab ca. 1,50 Euro im Elektronikhandel beschaffbar.

Hinweis

Der Begriff »Nullmodemkabel« wurde so genannt, weil auf den Einsatz eines früher gängigen Modems (Kurzform für Modulator / Demodulator) verzichtet wird.

Bei modernen PCs wurden die früher üblichen und gängigen RS232-Ports schon vor einigen Jahren durch USB-Ports ersetzt. Die Zubehörindustrie hat darauf mit Adaptern reagiert, die einen herkömmlichen RS232-Port über einen der im Regelfall mehreren USB-Anschlüsse zur Verfügung stellt. Ein einfacher Treiber sorgt dann für die USB- nach RS232-Umsetzung. Solche Adapter sind in den verschiedensten Ausführungen ab 10 Euro erhältlich.

Den Einsatz dieser Debugging-Methode werden Sie in Kapitel 4 kennenlernen. So viel sei aber hier schon einmal angemerkt: Das »Debugging Light« reduziert sich auf die Nutzung der `printf()`-Funktion von C und ist nicht besonders komfortabel, da das untersuchte Projekt nach jeder Änderung neu kompiliert, gelinkt und an das Entwicklungsboard übertragen werden muss.

Hinweis

Nicht zuletzt mit der Veröffentlichung des *Arduino*-Projekts oder *RasperryPi* ist die Anwendung von Mikrocontrollern auch für interessierte Hobby-Elektroniker wieder sehr interessant und vor allem erschwinglich geworden. Ebenfalls sehr interessant erscheint mir der von der deutschen Firma *TinkerForge* (`www.tinkerforge.com`) gewählte Ansatz: Um den Anwendern die Beschaffung und den Aufbau erforderlicher Elektronikkomponenten zu erleichtern und das Löten zu vermeiden, bietet TinkerForge ein Modulsystem verschiedener Komponenten mitsamt der dazugehörigen Software an. Dies ist zwar etwas teurer als der Selbstbau, bietet aber dafür den Vorteil, dass die Komponenten getestet und funktionsfähig sind. Noch besser ist hierbei, dass sowohl die Hardware wie auch die Software unter der sehr freien MIT-Lizenz (GPL V2.0+) stehen und auch in kommerziellen Projekten frei verwendet werden dürfen. Da das TinkerForge-Projekt auf den nächstgrößeren ATMEL-Mikrocontroller AT91SAM3S4C setzt (im Gegensatz zum in diesem Buch beschriebenen AT91SAM3S4B), sollte die Anwendung der folgenden Themen ohne weitere Änderungen ermöglichen.

1.4.2 Software

In diesem Abschnitt werden die Software-Tools vorgestellt, die in diesem Buch benutzt werden. An erster Stelle steht hier natürlich die Entwicklungsumgebung Eclipse, aber auch andere Software-Sammlungen, die für die Entwicklung von größter Bedeutung sind, werden hier aufgeführt. Auf Anleitungen zur Basis-Installation wird verzichtet, da es sich dabei um äußerst einfache und sehr gut beschriebene Vorgänge handelt, die im Allgemeinen auch noch in eine Installationssoftware eingebettet sind.

ATMEL SAM-BA In-System Programmer

Eines vorweg: Bei dem Programm ATMEL *SAM-BA* handelt es um eine Sammlung von Tools, die die In-System-Programmierung von ARM-basierten ATMEL-Controllern der Serien SAM3, 4, 7 und 9 unterstützt.

Wer sich nicht gleich zu Beginn seiner Einarbeitung in diesen Mikrocontroller mit der Entscheidung für oder gegen einen der im Abschnitt 1.4.1 aufgeführten In-Circuit-Emulatoren befassen möchte, ist aber dennoch auf eine Möglichkeit angewiesen, seine Software in den Chip bzw. auf das Board zu übertragen. Mit dem kostenlos von ATMEL zur Verfügung gestellten *SAM Boot Assistant* (dies ist die ausgeschriebene Form von SAM-BA) ist dies auf einfache Art und Weise möglich. Sie können festlegen, ob der entwickelte Binärcode in den Flash-Speicher oder in das SRAM des Controllers übertragen werden soll. Soll die Software in den integrierten Flash-Speicher des Mikrocontrollers übertragen werden, so muss eine Möglichkeit bestehen, ältere Software aus dem Flash zu löschen: Auch diese Funktion bietet SAM-BA. Darüber hinaus besteht die Möglichkeit, von Ihnen zu bestimmende Bereiche des Flash-Speichers vor dem Überschreiben zu schützen, weil sichergestellt ist, dass die dort gespeicherte Software funktioniert und wiederverwendet werden kann.

Hinweis

Dieses Buch ist überwiegend unter Einsatz von Microsoft Windows 7 Home Premium 64 Bit unter eingeschränkten Benutzerrechten entstanden. Obwohl SAM-BA in diesem Fall scheinbar korrekt startet, kann man diese Software nur mit Administratorrechten ausführen. Als weiteren Nachteil empfinde ich, dass SAM-BA für jede Nutzung neu gestartet werden muss: Programmieren des Flash-Speichers, anschließendes Löschen und Neubeschreiben ist nach meiner Erfahrung leider nicht möglich!

Eines noch: SAM-BA ist sowohl für Windows- wie auch für Linux-basierte Entwicklungsplattformen verfügbar. Eine ausführlichere Beschreibung von SAM-BA finden Sie in Kapitel 3.

Java Runtime Environment

Da Eclipse in der Programmiersprache *Java* geschrieben wurde, wird zwingend mindestens eine *Java-Laufzeitumgebung*, das sogenannte *Java Runtime Environment*, benötigt. Sie können selbstverständlich auch das vollständige Java 2 Software Development Kit (J2SDK) installieren: Dies ist aber nur erforderlich, wenn Sie zusätzlich auch in Java programmieren wollen. Es existieren zwei voneinander unabhängige Bezugsquellen. Im ersten Fall wird die Weiterentwicklung von Java seit der Übernahme von Sun Microsystems Inc. durch die Firma Oracle fortgeführt. Im zweiten Fall existiert eine »echte Open-Source-Version« von Java, die

standardmäßig von Linux-Distributionen aufgrund der Open-Source-Bewegung uneingeschränkt nutzbar ist (allerdings sind mir auch seitens der Oracle-Version keine gravierenden lizenzrechtlichen Einschränkungen bekannt). Beide Versionen sind meines Wissens kompatibel zueinander und können auch nebeneinander eingesetzt werden.

Die OpenJDK-Version können Sie von `http://openjdk.java.net`, die Oracle-Version können Sie von `http://www.oracle.com/technetwork/java/javase/downloads/index.html` herunterladen, wobei bei der Oracle-Version eine Bestätigung der Lizenzbestimmungen erforderlich ist.

Wenn nicht bereits geschehen, ist die Installation einer der genannten Java-Laufzeitumgebungen der erste Schritt beim Aufbau der Entwicklungsumgebung für den Cortex-M3.

Yagarto (für Windows und Mac OS X)

Yagarto ist die Abkürzung von *Yet Another GNU ARM Toolchain*, die von *Michael Fischer* entwickelt und gepflegt wird. Sie stellt alle zur erfolgreichen Programmierung von ARM-basierten Mikrocontrollern erforderlichen Hilfsmittel, wie z.B. den *GNU-Assembler arm-none-eabi-as*, den *GNU-Compiler arm-none-eabi-gcc*, den *GNU-Linker arm-none-eabi-ld* und den *GNU-Debugger arm-none-eabi-gdb* in einer auf *Windows*- und *Mac OS X*-Betriebssystemen nativen Form, das heißt, ohne Einsatz eines *Linux-Wrappers*, wie z.B. *Cygwin* oder *MinGW*, zur Verfügung. Darüber hinaus hat Michael Fischer auch native Windows-Versionen der GNU-Programme *arm-none-eabi-objcopy*, *arm-none-eabi-objdump*, *arm-none-eabi-size* usw. entwickelt. Mit den Yagarto-Tools hat Michael Fischer darüber hinaus einige der wichtigsten generell für die Software-Entwicklung nützlichen Programme nativ für Windows und Mac OS X entwickelt. Hierzu zählen u.a. die UNIX- bzw. Linux-Kommandos *make*, *arm-none-eabi-cp*, *rm*, *sh* und *touch*. Die Installation der Yagarto-Toolchain und der Yagarto-Tools ist äußerst einfach gehalten, da eine vollständig dialogorientierte Installationssoftware zur großflächigen Distribution eingesetzt wurde. Seit dem Umzug der Webseite können Sie die Yagarto-Toolchain und die Yagarto-Tools unter

`http://www.emb4fun.de/arm.html`

kostenlos herunterladen.

Hinweise zur Installation unter Windows

Achten Sie bei der Installation darauf, dass der Name des Installationsverzeichnisses keine Leerzeichen enthält. Dies ist wichtig, da die Möglichkeit besteht, dass die Tools sonst nicht funktionieren. Aktivieren Sie auch die Checkbox im Installer, die den Suchpfad (Umgebungsvariable PATH) um das Installationsverzeichnis der beiden Software-Pakete erweitert. Dies erleichtert die weitere Vorgehensweise erheblich.

Wenn Sie die Yagarto-Toolchain und die Yagarto-Tools verwenden wollen, sollten Sie sie installieren, bevor Sie Eclipse installieren. Obwohl auch die umgekehrte Reihenfolge möglich ist, könnte es sein, dass bei der vorgeschlagenen Vorgehensweise die Konfiguration etwas vereinfacht wird.

EABI oder Non-EABI?

EABI ist die Abkürzung von *Embedded Application Binary Interface* und stellt das aktuelle Format dar, das für die Entwicklung von ARM-basierter Software verwendet wird. Unter einem Application Binary Interface versteht man allgemein die Verknüpfung einer Applikation mit einem Betriebssystem. Für ARM-basierte Software bietet sich hier z.B. ein Mini-Linux an, wie man es beispielsweise in Getränkerückgabestationen (Pfandbon-Erstellung) vorfindet. In solchen Fällen verwendet man demnach die EABI-Version der Entwicklungstools. In diesem Buch werden Sie aber ohne Linux (oder einem vergleichbaren Betriebssystem) arbeiten. Aus diesem Grund werden Sie bei der Bearbeitung Ihrer Projekte mit der Non-EABI-Version der Entwicklungstools arbeiten.

Launchpad

Anstelle der nur für Windows und Mac verfügbaren Yagarto-Toolchain können Sie auch die von der Firma ARM gepflegte und ebenfalls GCC-basierte *launchpad*-Toolchain verwenden. Diese Toolchain ist für Windows, Mac OS X und natürlich (GNU eben) auch für Linux verfügbar.

Hinweis

Stand Oktober 2013 betrachtet Michael Fischer die Entwicklung von Yagarto als vollendet. Es ist zu befürchten, dass die aktuelle Version nicht weiter gepflegt wird. Obwohl ich sämtliche Beispiele zunächst unter Verwendung der Yagarto-Toolchain entwickelt habe, habe ich mich dazu entschlossen, alle Beispielprojekte unter Einsatz der nachfolgend vorgestellten Launchpad-Toolchain neu zu schreiben.

Herunterladen können Sie diese von der Webseite `https://launchpad.net/gcc-arm-embedded`. Der große Vorteil dieser Toolchain ist, dass im Gegensatz zu Yagarto sämtliche Tools zur Verfügung stehen. Unter Windows ist die Installation dieser Toolchain vermutlich am einfachsten, steht hier doch ein vollwertiger Installer zur Verfügung. Ein weiterer Vorteil liegt darin, dass für alle genannten Betriebssysteme immer die gleiche Version verfügbar ist. Für Mac OS X und für Linux ist die Installation nicht ganz so einfach, da hier nur ein komprimiertes Archiv, ein sogenannter Tar-Ball, bereitgestellt wird. Sie erkennen dies an der Dateiextension `.tar.bz2`. Mit den betriebssystemeigenen Hilfsmitteln sollte die

Installation aber kein Problem darstellen. Beachten Sie bei diesen Betriebssystemen nur, dass Sie den Installationspfad inklusive dem Verzeichnis `bin` in den Suchpfad aufnehmen. Unter Linux (möglicherweise auch unter Mac OS X) müssen Sie darüber hinaus berücksichtigen, dass Sie die 32-Bit-Versionen der beiden Bibliotheken `glibc` und `libncurses` installieren, die für das Arbeiten mit `launchpad` erforderlich sind.

Sofern noch nicht geschehen, sollten Sie an dieser Stelle prüfen, ob die Tools aus dem Segment »Software Development« bereits installiert sind. In der Linux-Shell reicht es hierfür aus, einfach das Kommando `arm-none-eabi-gcc --version` einzugeben. Zum Zeitpunkt, als dieses Kapitel entstand, war die aktuelle Version 4.7.4.

Der Installer von Launchpad bietet Ihnen in der Windows-Version ebenfalls an, den Suchpfad zu aktualisieren: Markieren Sie die entsprechende Checkbox.

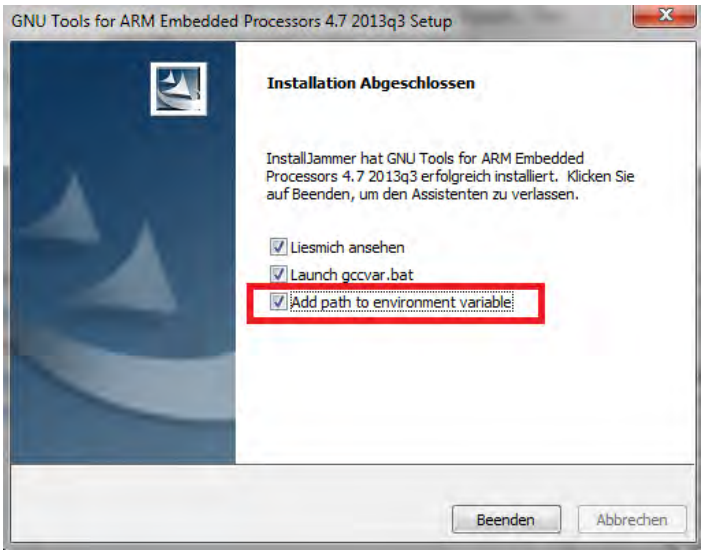

Abb. 1.1: Installation von Launchpad

Problem: Wechseln der Toolchains und wieder zurück

Haben Sie sich bereits für eine der genannten Toolchains entschieden und wollen nun eine andere ausprobieren, ist größte Vorsicht geboten! Eigene Erfahrungen, aber auch die Erfahrungen anderer (z.B. im Stackoverflow-Forum) weisen darauf hin, dass ein Bug im CDT die Rückkehr zur ursprünglichen Toolchain nicht zulässt. Die Folge sind Fehlermeldungen, die nur schwer nachvollziehbar sind.

MinGW

Wenn Sie als Betriebssystem Windows verwenden, empfehle ich Ihnen die Installation von *MinGW* (*Minimal GNU Tools for Windows*): Damit stehen Ihnen sämtliche GNU-Tools für die Entwicklung von ARM-Projekten zur Verfügung. Für die Installation steht ein ausgereiftes Setup-Programm zur Verfügung. Vergessen Sie nicht, das `bin`-Verzeichnis von MinGW in den Suchpfad aufzunehmen. MinGW können Sie von der Internetseite `http://www.mingw.org` kostenlos herunterladen.

Entwickeln Sie Ihre Software stattdessen unter Linux, ist die Installation im Allgemeinen nicht erforderlich, da sämtliche Tools im Bereich Software Development der von Ihnen bevorzugten Linux-Distribution verfügbar sind.

Eclipse

Hinweis

Die Entwickler von Eclipse und erforderlicher Plug-ins haben mir, während dieses Buch entstand, gleich mehrere »Streiche« gespielt: Das kleinste Problem war hier noch die Veröffentlichung von Eclipse 4.3 (Kepler). Gravierender war die End-of-Life-Meldung bezüglich der CDT GNU ARM C/C++ Development Tools, die von CDT GNU ARM C/C++ Cross Compiler Support abgelöst wurden: Dies führte dazu, dass KEINES der ursprünglich unter Eclipse Indigo unter Verwendung der Yagarto-Toolchain entwickelten Projekte mehr funktionierte. Daher habe ich mich dazu entschlossen, sämtliche Beispiele neu zu entwickeln, und habe in diesem Zusammenhang auch die Yagarto-Toolchain durch Launchpad ersetzt. Die ersten drei Kapitel habe ich entsprechend modifiziert, die weiteren Kapitel entstanden erst nach diesem Update.

Der sichtbare Rahmen für die weitere Entwicklung von Software für das Olimex-Board wird von der Entwicklungsumgebung Eclipse ausgefüllt. Eclipse steht für die Betriebssysteme Windows, Mac Cocoa und Linux jeweils in einer 32-Bit- und einer 64-Bit-Version unter der folgenden Internetadresse bereit (Abbildung 1.2):

```
http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/
keplersr1
```

Die beiden eingerahmten Bereiche in Abbildung 1.2 weisen auf die Eclipse-Version hin, die in diesem Buch verwendet wurde.

Eclipse unter Windows installieren

Obwohl für Eclipse keines der oben genannten Installationsprogramme zur Verfügung steht, ist die Installation dennoch sehr einfach: Sie beschränkt sich nämlich

auf das Entpacken einer ungefähr 135 MByte großen Archiv-Datei. Mit dem kostenlos erhältlichen Programm 7zip (das Programm ist unter der Internetadresse `http://www.7-zip.de` ladbar) können Sie Eclipse in ein Verzeichnis Ihrer Wahl entpacken, von wo es sofort ausführbar ist (die ausführbare Datei `Eclipse.exe` befindet sich dort im Unterverzeichnis `bin`). Das Erste, was Sie dort sehen, ist die Aufforderung zur Festlegung des sogenannten *Workspace*. Mit Workspace ist ein Bereich auf der Festplatte gemeint, in dem Eclipse später sämtliche Entwicklungsprojekte anlegt (siehe Abbildung 1.3). Um zu vermeiden, dass Sie bei jedem neuen Projekt nach dem Workspace gefragt werden, sollten Sie die Checkbox USE THIS AS THE DEFAULT AND DO NOT ASK AGAIN abhaken.

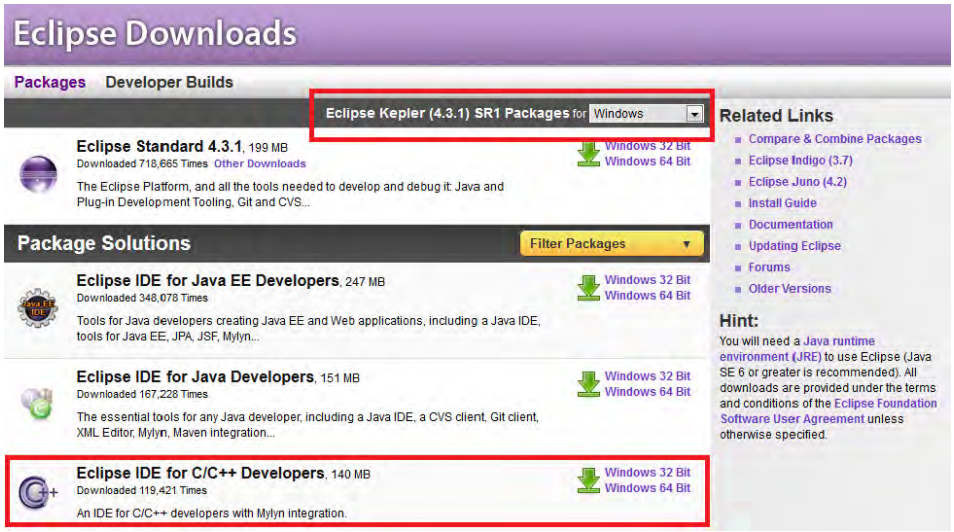

Abb. 1.2: Beschaffung von Eclipse

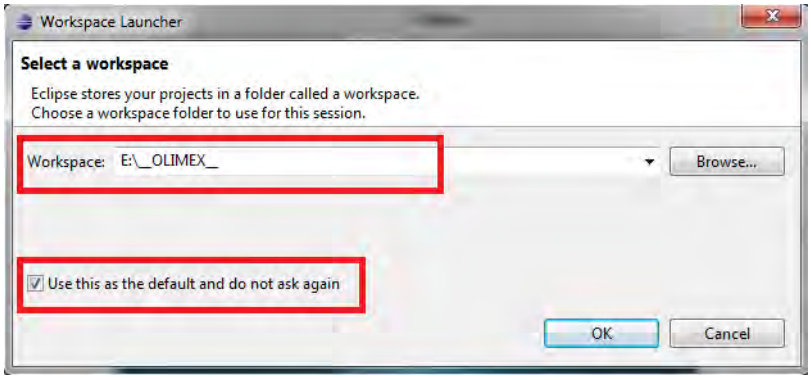

Abb. 1.3: Der erste Kontakt mit Eclipse

Vorschlag zur Partitionierung der Festplatte

Das Verzeichnis `E:\__OLIMEX__` ist das Verzeichnis, in dem ich den Workspace angelegt habe. Die Defaulteinstellung dieses Dialogs weist hingegen auf ein Verzeichnis der Partition `C:\` hin. Sie sehen also: Die Wahl des Workspace-Verzeichnisses liegt ganz bei Ihnen. Generell ist es empfehlenswert, eine strikte Trennung zwischen Betriebssystem-Partition (dies ist unter Windows im Allgemeinen die Partition `C:\`), einer Partition für Programme (bei mir ist dies Partition `D:\`) und einer Datenpartition (bei mir: Partition `E:\`) einzuführen. Diese Vorgehensweise bietet erhebliche Vorteile bei der Sicherung und Wiederherstellung sämtlicher Daten. Getrennte Sicherungen von Betriebssystem, Anwendungen und Daten lassen sich auf diese Weise leicht, schnell und vor allem ohne Seiteneffekte wiederherstellen.

Der Begrüßungsbildschirm von Eclipse heißt Sie anschließend willkommen (Abbildung 1.4).

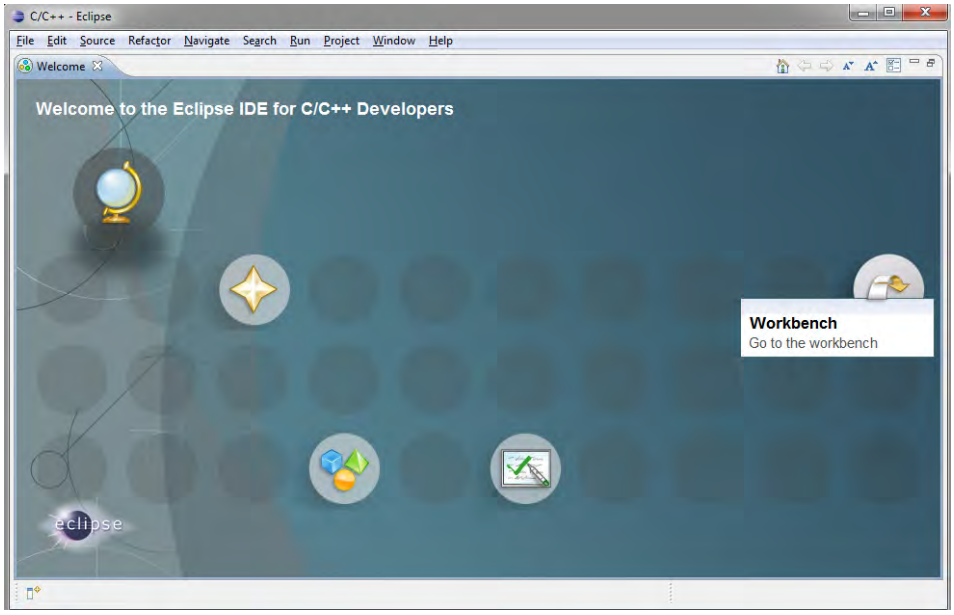

Abb. 1.4: Begrüßungsbildschirm von Eclipse

Durch Anklicken der WORKBENCH: GO TO THE WORKBENCH-Schaltfläche schaltet Eclipse dann in den Projektbereich um, der in Abbildung 1.5 gezeigt wird.

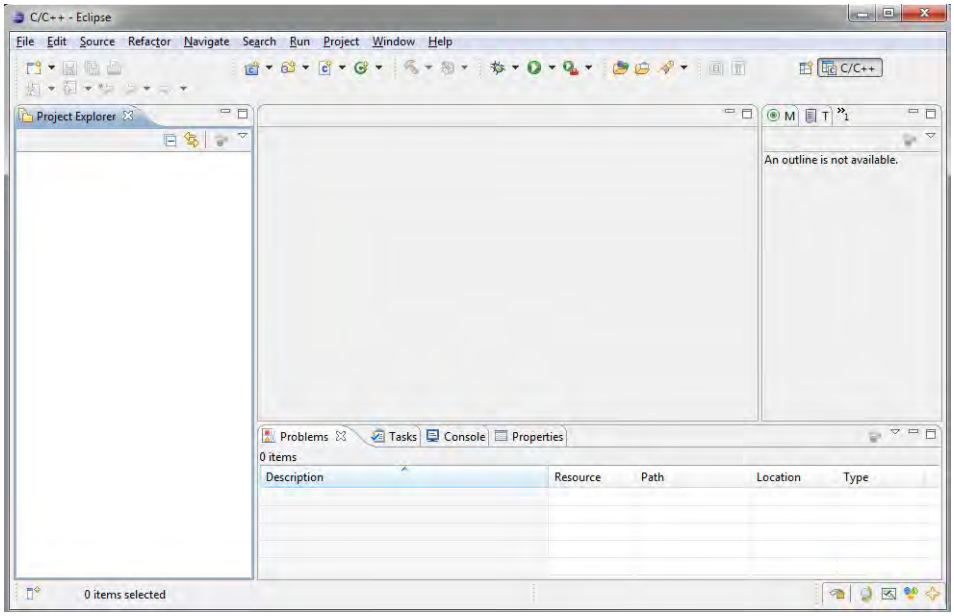

Abb. 1.5: Der Arbeits- bzw. Projektbereich von Eclipse

Literatur-Hinweis

In diesem Buch wird auf die Bedienung von Eclipse nur dann näher eingegangen, wenn dies zwingend erforderlich ist, z.B. beim Erstellen des ersten Projekts, bei der Konfiguration und Nutzung des Debuggers oder bei der Installation zusätzlicher Plug-ins. Allgemeine Literatur existiert bereits zur Genüge und soll deshalb hier nicht wiederholt werden! Wenn Sie alle Möglichkeiten nutzen wollen, die Eclipse C- oder C++-Entwicklern bietet, sollten Sie sich mit dem ausgesprochen guten Buch »Eclipse für C/C++-Programmierer« von Sebastian Bauer, erschienen im dpunkt.verlag, näher befassen: Hier finden Sie fundierte Informationen und Beispiele (allerdings, wie es auch für dieses Buch gilt, keinen C-Lehrgang!) zu allen Themen, die in der modernen Software-Entwicklung in Verbindung mit Eclipse von Bedeutung sind.

Eclipse unter Linux und Mac OS X installieren

Zur Installation von Eclipse unter Mac OS X kann ich mich leider nicht auslassen, da mir keine Entwicklungsplattform mit diesem Betriebssystem zur Verfügung steht. Aufgrund des UNIX-ähnlichen Betriebssystems ist aber davon auszugehen, dass die Installation ähnlich verläuft wie auf einem UNIX- bzw. Linux-Betriebssystem.

Die Installation von Eclipse unter Linux erfolgt auf eine vergleichbare Weise, wie sie bereits bei der Installation der Compiler-Toolchain beschrieben wurde. Alle gängigen Linux-Distributionen verwenden Paketmanager, die die Installation von neuer Software auf Mausklick ausführen. Geben Sie in das Suchfeld des jeweiligen Paketmanagers (sofern Sie die grafische Installationsweise bevorzugen) als Suchbegriff *Eclipse* ein, und ein weiterer Mausklick auf »Ausführen« sorgt für das Herunterladen von Eclipse, die Auflösung von Abhängigkeiten und abschließend den Prozess der Installation. Die Abbildungen, die Ihnen bei der Installation von Eclipse begegnen, sollten identisch mit denen in Abbildung 1.2 bis Abbildung 1.5 sein.

Beschaffung und Installation erforderlicher Plug-ins

Wenn Sie diesen Punkt erreicht haben, sind Sie in der Lage, Software in C oder C++ für das Betriebssystem Ihrer Wahl zu entwickeln. Für die Programmierung eines Cortex-M3-Prozessors (oder eines anderen ARM-basierten Mikrocontrollers) reicht dies aber noch nicht ganz: Es fehlt noch eine als Plug-in bezeichnete Erweiterung für Eclipse, die für das bereits erwähnte Cross-Compiling, also das Kompilieren von Software für eine andere Plattform, dient. In den folgenden Schritten werden Sie dieses Plug-in aus dem Internet herunterladen und in Eclipse installieren. Durch Klicken auf den Menüeintrag HELP:INSTALL NEW SOFTWARE (siehe Abbildung 1.6) öffnet sich der in Abbildung 1.7 gezeigte Dialog, der Ihnen eine Auswahl der verfügbaren Software anzeigt. Dass dieser Dialog derzeit keine nützlichen Hinweise anzeigt, liegt daran, dass der Plug-in-Manager von Eclipse noch keine Informationen über verfügbare Software hat. Diese Informationen werden Sie dem Plug-in-Manager aber in den nächsten Schritten mitteilen.

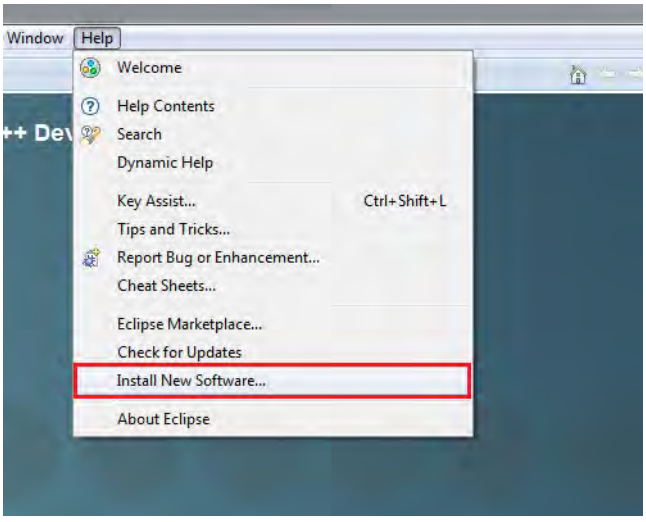

Abb. 1.6: Neue Software zu Eclipse hinzufügen

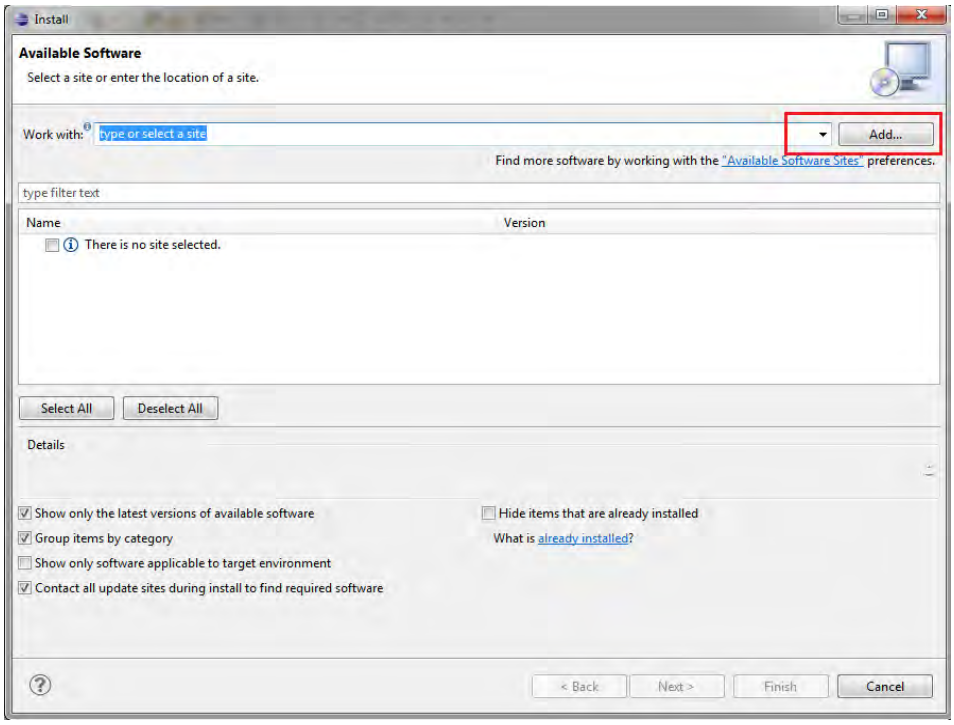

Abb. 1.7: Dialog zur Anzeige verfügbarer Software

Tipp

Von Zeit zu Zeit werden Plug-ins erweitert oder verbessert. Daher bietet es sich an, das Plug-in aus dem Internet zu beziehen und auf die lokale Speicherung zu verzichten. Konfigurieren Sie Eclipse aber so, dass ein Update erst nach ausdrücklicher Bestätigung durch Sie durchgeführt wird. Ansonsten könnte es Ihnen passieren, wie es mir ergangen ist: Nach einem Update funktionierte gar nichts mehr. Im Zweifelsfall sollten Sie zunächst die einschlägigen Foren im Internet verfolgen oder Updates im Rahmen einer parallelen Installation von Eclipse testen.

Einige Pakete, wie z.B. das nachfolgend beschriebene C/C++ Development Tool SDK, werden von eclipse.org direkt verwaltet und gepflegt: Für solche Software können Sie die in Abbildung 1.7 hervorgehobene Schaltfläche ADD... ignorieren. Es gibt aber auch Software-Pakete, die nicht von eclipse.org verwaltet werden: In solchen Fällen (ich komme weiter unten hierauf zurück) wird diese Schaltfläche aber benötigt. Jetzt geben Sie in dem WORK WITH:-Eingabefeld einfach den Text »Kepler« ein (die nachfolgende Internetadresse wird automatisch ergänzt). Abbildung 1.8 zeigt das Ergebnis.

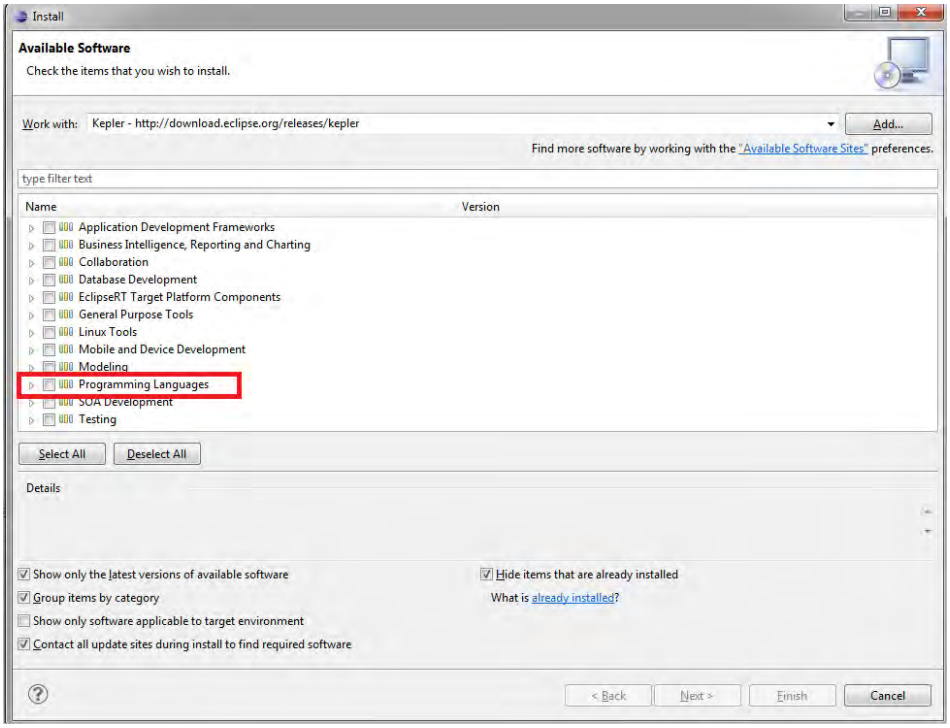

Abb. 1.8: Verfügbare Software und Plug-ins

Wählen Sie hier die mit PROGRAMMING LANGUAGES bezeichnete Checkbox. Der in Abbildung 1.9 abgebildete Dialog zeigt Ihnen eine Auswahl der unterstützten Programmiersprachen an. Markieren Sie hier den Eintrag C/C++ DEVELOPMENT TOOLS SDK.

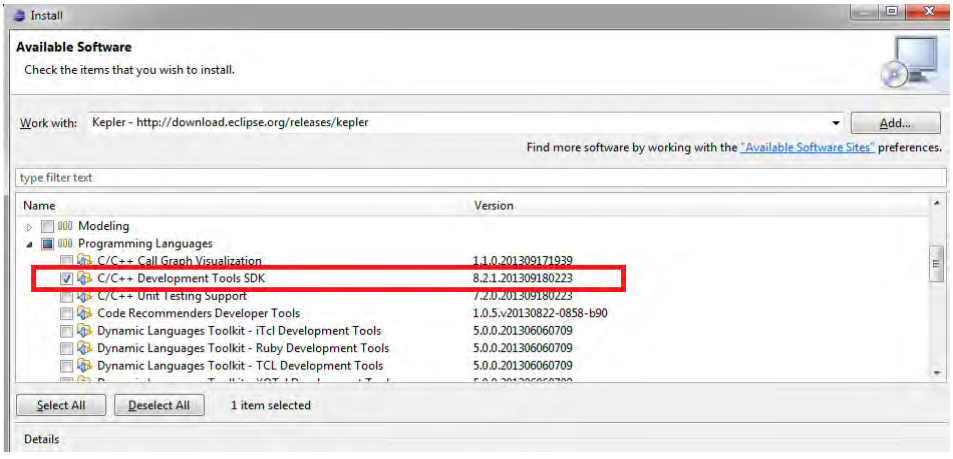

Abb. 1.9: Auswahl des C/C++ Development Tools SDK

Abbildung 1.10 zeigt Ihnen, welche Tools bzw. Plug-ins für die weitere Arbeit benötigt werden.

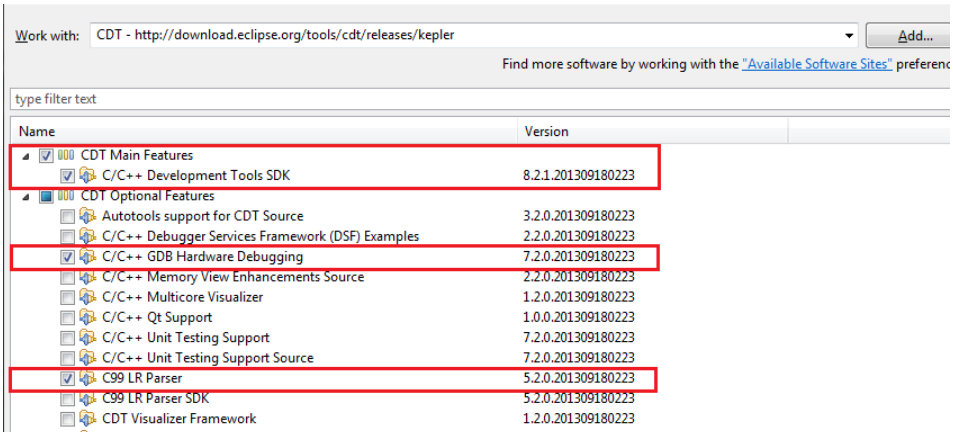

Abb. 1.10: Benötigte Tools aus dem C/C++ Development Tools SDK

Hinweis

In meiner Entwicklungsumgebung habe ich noch weitere Software installiert. Die in Abbildung 1.10 angezeigten Pakete sind für die weitere Arbeit zwingend erforderlich. Hilfreich (und bei mir installiert) sind aber Tools, wie z.B. Subversive zur Versionsverwaltung mit Subversion (SVN).

Für die Beschaffung und Installation des nächsten zwingend erforderlichen Software-Pakets müssen Sie die in Abbildung 1.11 gezeigte Schaltfläche Add... anklicken. Abbildung 1.11 zeigt zwei Eingabefelder, die Sie entsprechend selber ausfüllen müssen.

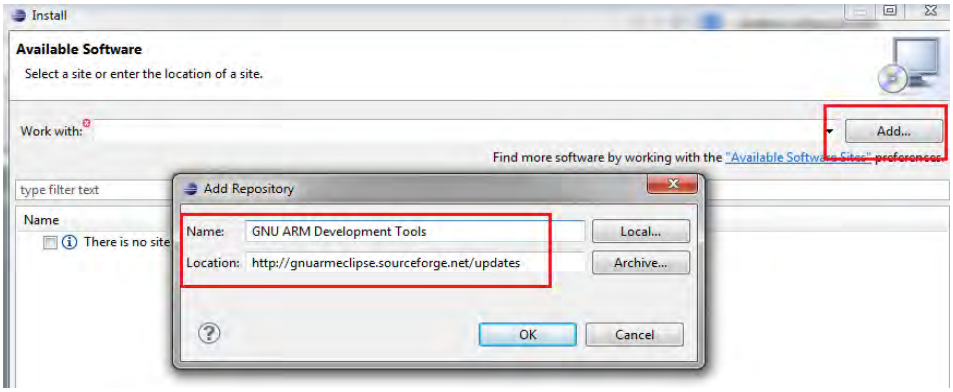

Abb. 1.11: Hinzufügen der Launchpad-Unterstützung

In dem mit NAME: bezeichneten Eingabefeld sind Sie noch völlig frei, was dort eingetragen wird. Ich empfehle aber einen aussagekräftigen Namen. Das Eingabefeld mit der Bezeichnung LOCATION: muss aber genauso ausgefüllt werden, wie es in Abbildung 1.11 gezeigt wird:

`http://gnuarmeclipse.sourceforge.net/updates`.

Hinweis

Hiermit wird nicht Launchpad installiert. Stattdessen wird Eclipse nur bekannt gemacht, dass die Launchpad-Toolchain für die Softwareentwicklung angeboten werden soll!

Nach kurzer Zeit des Datenabgleichs und dem Auflösen möglicher Abhängigkeiten erscheint der in Abbildung 1.12 gezeigte Dialog.

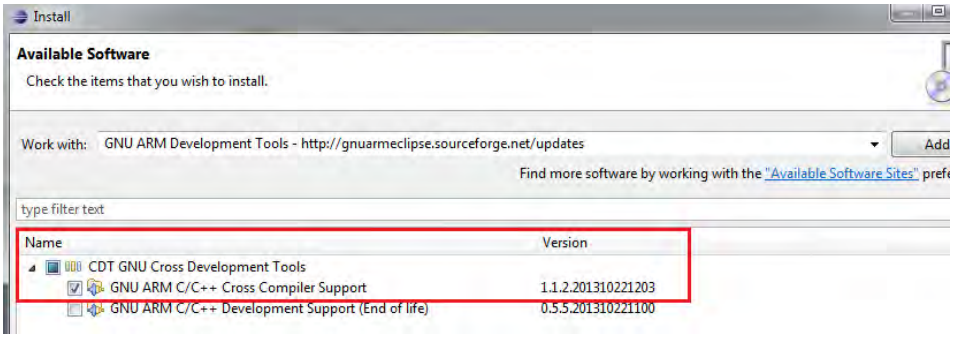

Abb. 1.12: Installieren der CDT GNU Cross Development Tools

Sie können erkennen, dass hier zwei Einträge angezeigt werden, wobei ich für die Abbildung den zweiten bewusst nicht in den Rahmen aufgenommen habe: Bei dem zweiten Eintrag handelt es sich nämlich um den nächsten »Streich«, den mir die Entwickler gespielt haben. Dieses mit End-of-Life markierte Softwarepaket wird nicht nur nicht mehr weitergepflegt, von einem weiteren Einsatz wird sogar abgeraten. Der Einsatz des neuen Pakets führte bei mir aber zu derart gravierenden Problemen, dass ich dieses und das folgende Kapitel komplett neu schreiben und die Beispielprojekte neu entwickeln musste.

Jeder einzelne der beschriebenen Installationsschritte erfordert die Anerkennung der Lizenzbedingungen. In vielen Fällen folgt noch eine Warnung (siehe Abbildung 1.13) vor nicht-signierter Software, die Sie aber durch einen Klick auf die OK-Schaltfläche akzeptieren können. Nach jedem dieser Installationsschritte sollte Eclipse neu gestartet werden: Eclipse lässt Ihnen zwar die Möglichkeit, den Neustart zu verschieben, ich habe selber aber immer den Neustart gewählt, um mögliche Seiteneffekte auszuschließen.

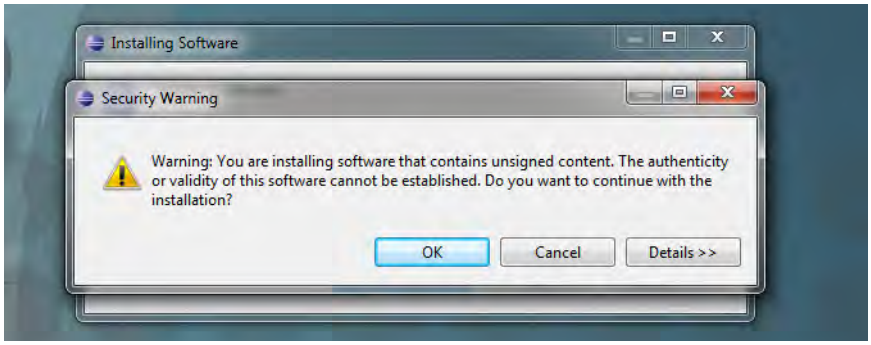

Abb. 1.13: Warnung vor nicht-signierter Software

Zur Kontrolle sollten Sie jetzt noch überprüfen, dass Sie keine der in Abbildung 1.14 gezeigten Plug-ins übersehen haben.

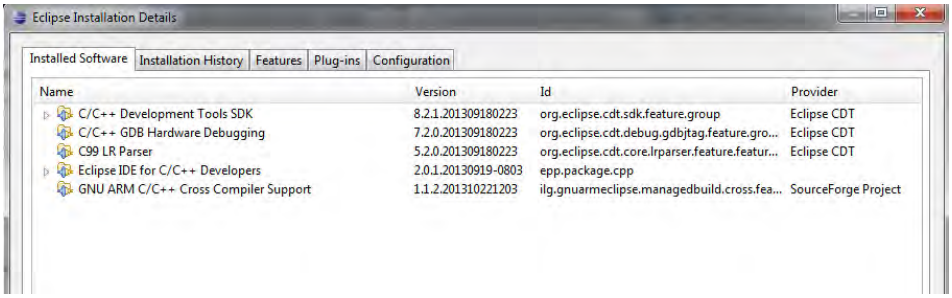

Abb. 1.14: Minimalinstallation

Abbildung 1.15 zeigt schließlich, dass die neu installierten Software-Pakete Eclipse nun vollständig bekannt sind. Diese Pakete sind jetzt, wie bereits zu Abbildung 1.7 ausgeführt, über das Eingabefeld WORK WITH: erreichbar. Der besondere Vorteil ist, dass Eclipse nun in der Lage ist, die Bereitstellung von Updates zu diesen Paketen zu überwachen und durch eine entsprechende Meldung anzukündigen.

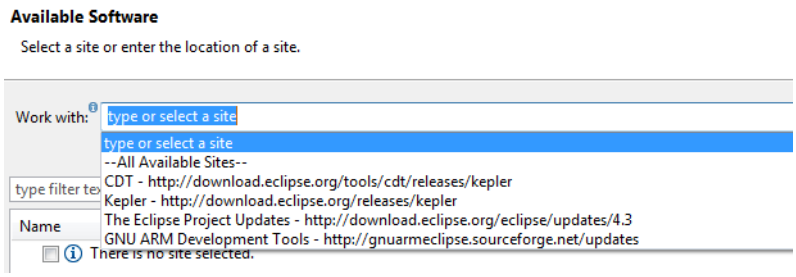

Abb. 1.15: Die Installation ist abgeschlossen.

Hiermit ist der erste Teil der »Inbetriebnahme« von Eclipse für die ARM-Cross-Kompilation abgeschlossen. In Kapitel 2 wird nach einigen einleitenden Erläuterungen zur Firma ARM und zu CMSIS der sehr wichtige Aspekt der Beschaffung bzw. Erzeugung zwingend erforderlicher Bibliotheken für die GNU-ARM-Toolchain behandelt.