Computer Systems and Telematics — Distributed, Embedded Systems

Bachelorarbeit

Evaluation einer Umgebung für die Skriptsprache Lua auf dem Sensorboard MSB-IOT

Richard Möhn
Matrikelnummer 4451490

Betreuer: Prof. Dr. rer. nat. Mesut Güneş
Betreuender Assistent: Dipl.-Inf. (FH), M. Sc. Michael Frey

Institut für Informatik, Freie Universität Berlin, Deutschland

9. Januar 2014

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen, die wörtlich oder sinngemäß veröffentlichten Schriften entnommen wurden, sind als solche gekennzeichnet. Die Abbildungen habe ich entweder selbst erstellt oder mit Quellennachweisen versehen. Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner Prüfungsbehörde eingereicht.

Berlin, den 24. März 2014

Zusammenfassung

Das Programmieren in Skriptsprachen bietet eine Reihe von Vorteilen, ist aber im Bereich der Embedded-Geräte nicht verbreitet. An der Freien Universität Berlin wurde das Sensorboard MSB-IOT mit vergleichsweise großzügiger Ausstattung (Cortex-M4-Prozessor, 196 kB RAM, 1 MB ROM) entwickelt und es ist zu ermitteln, welche Programmierumgebungen Benutzern zur Verfügung stehen sollen. In dieser Arbeit wird eLua, eine Umgebung für die Skriptsprache Lua, analysiert. Es wird untersucht, welche Arbeiten nötig sind, damit das MSB-IOT Lua-Skripte ausführen kann und damit diese auf beliebige Funktionen der Hardware zugreifen können. Dazu werden eLua auf das Board portiert und Vermittlungsschichten entwickelt, um zwei in C geschriebene Treiber einzubinden. Das MSB-IOT kann sich dadurch Lua-gesteuert mit WLANs assoziieren und per Funk kommunizieren. Es ist also möglich, Software für ein Sensorboard in einer Kombination von C/C++ und einer etablierten Skriptsprache zu schreiben und damit die Sprachen ihrer Stärken gemäß einzusetzen.

Abstract

Although programming in scripting languages has several advantages, it is not usually done in the realm of embedded hardware. The sensor board MSB-IOT was developed at Freie Universität Berlin. It is a comparatively powerful device (Cortex-M4 MCU with 196 kB RAM and 1 MB ROM) and it is to be considered which programming environments are best to utilise this power. In this thesis eLua, standing for Embedded Lua, is proposed. It is investigated what work it requires to make the MSB-IOT run Lua scripts and to open to these the capabilities of the hardware. For that, eLua is ported to the MSB-IOT and two drivers written in C are integrated through compatibility layers. This enables the MSB-IOT to associate with Wi-Fi networks and communicate via radio, all controlled by Lua scripts. It is thus possible to develop software for a sensor board as a combination of C/C++ and an established scripting language, thereby benefiting from the virtues of each.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quelltextverzeichnis

1 Einleitung

Skriptsprachen bieten eine Reihe von Vorteilen [1, 2]: Sie benutzen statt starrer Typsysteme Wörterbücher und Zeichenketten zur Datenrepräsentation. Sie sind dadurch flexibler und eignen sich gut, um existierende Software zu neuen Anwendungen zu verbinden. Sie haben automatische Speicherbereinigung. Sie bieten mächtige Programmierkonstrukte in dem Sinne, dass viele Anweisungen deutlich mehr Maschinenbefehle verbergen als beispielsweise in C. Lernende können so schon mit wenig Wissen nützliche Programme schreiben; Programmierer allgemein sind produktiver als mit anderen Sprachen [3].

Die Skriptsprache Lua ist mit Augenmerk auf Einfachheit und Leichtgewichtigkeit entworfen und damit für einen Menschen einfach erlern- sowie von einem leichtgewichtigen Interpreter ausführbar [4]. Lua ist so einfach, dass es benutzt wird, um Kinder das Programmieren zu lehren [5, 6, 7]. Dabei ist es allerdings nicht die einzige Programmiersprache [8]. Der Interpreter dagegen ist so leichtgewichtig, dass er als C-Bibliothek in viele andere Programme eingebettet ist [9] und schon von Systemen mit 256 kB ROM und 64 kB RAM ausgeführt werden kann. Einfachheit und Leichtgewichtigkeit begrenzen aber nicht die Benutzbarkeit von Lua, denn die Sprache verfügt über eine Reihe von Erweiterungsmechanismen [10]. Unter anderem kann man über ein einfaches Foreign Function Interface den vollen Funktionsumfang von C oder C++ ausnutzen [11].

Ein Lua-Interpreter ist auch Kernstück der Programmierumgebung eLua. Er wird vom Prozessor als einziges Programm ausgeführt [12] und ermöglicht die Embedded-Programmierung in Lua. eLua macht sich Leichtgewichtigkeit und Erweiterbarkeit von Lua zunutze, um dessen Einfachheit mit der Fähigkeit von C zu verbinden, beliebig nah an der Hardware zu programmieren. Es bringt also die Vorteile von Skriptsprachen in den Embedded-Bereich und kombiniert sie mit den Vorteilen der dort üblichen Programmiersprachen: Der Programmierer kann auf vorhandene Software zurückgreifen, kann Treiber auf natürliche Weise entwickeln, performancekritische Teile des Codes nach C/C++ auslagern und dann alles mit Lua zusammensetzen.

Das Sensorboard MSB-IOT [13] wurde vor kurzem an der Freien Universität Berlin als Plattform für Sensornetze und Internet-der-Dinge-Anwendungen entwickelt. Es verfügt über einen 32-Bit-Prozessor mit 1 MB ROM und 196 kB RAM und und besitzt als besonderes Merkmal ein WLAN-Modul. Die großzügige Ausstattung erlaubt, andere Programmierumgebungen als C oder C++ zu unterstützen. In dieser Arbeit wird eLua als Möglichkeit für eine solche Programmierumgebung betrachtet. eLua wird für das MSB-IOT portiert und anhand eines Funk-Transceivers und des erwähnten WLAN-

Moduls gezeigt, wie man Treiber für Peripheriegeräte einbindet. Die Arbeit soll damit die Grundlage schaffen sowie Orientierung bieten für die Entwicklung einer vollständigen Lua-Umgebung auf dem MSB-IOT.

1.1 Verwandte Arbeiten

Ousterhout [1] unterscheidet zwischen Skriptsprachen, beispielsweise Tcl, und Sprachen zur Systemprogrammierung, beispielsweise C, und nennt sie komplementär: Letztere eignen sich, um Komponenten von Grund auf zu bauen, während man diese Komponenten mit ersteren schnell und einfach zusammensetzen kann. Ousterhout zählt auch Java zu den Sprachen zur Systemprogrammierung, wogegen Spinellis [2] anführt, dass Java und C# (eines von »Microsoft's .NET offerings«) automatische Speicherbereinigung und umfangreiche Bibliotheken bieten. Sie wären also zwischen Skriptsprachen und Sprachen zur Systemprogrammierung einzuordnen.

In diesem von Ousterhout und Spinellis aufgespannten Spektrum bewegen sich die Möglichkeiten, eingeschränkte beziehungsweise Embedded-Hardware zu programmieren. Dabei sind C und C++ als Sprachen zur Systemprogrammierung vorherrschend [14, 15]. Für Java existiert beispielsweise die TakaTuka Java Virtual Machine (JVM) [16]. Sie erlaubt, Java-Programme auf Systemen mit weniger als 10 kB RAM auszuführen, indem sie Teile des Programmlebenszyklus auf den Host-Rechner auslagert (geteilte virtuelle Maschine). Die TakaTuka JVM entspricht der CLDC-Spezifikation (Connected Limited Device Configuration), einer Spezifikation für Java auf ressourcenarmen Geräten [17], erfordert aber das Betriebssystem TinyOS als Hardwareabstraktion.

Microsoft hat das .NET Micro Framework (.NET MF) entwickelt, das ermöglicht, eingebettete Geräte in C# zu programmieren. .NET MF hat einige Ähnlichkeiten mit eLua: Es läuft direkt auf der Hardware, ist auf beliebige Systeme portierbar und Entwickler können Erweiterungen in C und C++ schreiben und in C#-Programme einbinden. Carlson, Mittek und Pérez [18] stellen .NET MF vor und vergleichen es mit zwei anderen Umgebungen zur Embedded-Programmierung.

Dunkels [19] sieht Skriptsprachen als Möglichkeit zur Neuprogrammierung von Sensorknoten in bereits installierten Sensornetzwerken. Er beschreibt die eigens entwickelte Sprache SCript und stellt sie bestehenden Technologien für Embedded-Geräte gegenüber. Seine Zielsysteme sind allerdings MSP430-Boards mit ein bis zwei Größenordnungen weniger Speicher als das MSB-IOT. Entsprechend primitiver ist SCript gegenüber Allzweck-Skriptsprachen wie Lua.

Ein bis zwei Größenordnungen mehr Speicher als das MSB-IOT haben die Embedded-Geräte bei Bennett [3]. Er diskutiert, unter anderem anhand zweier Fallbeispiele, sinnvolle Einsatzgebiete von Skriptsprachen im Embedded-Bereich. Die Sprache seiner Wahl ist Jim Tcl, eine für ressourcenbeschränkte Geräte angepasste Implementierung

von Tcl. Tcl ist ähnlich Lua einbettbar und mit C/C++ erweiterbar [20]. Tatsächlich führt Bennett Lua als nahezu gleichwertige Alternative zu Tcl an.

Alle diese Lösungen, bis auf .NET MF, benötigen entweder ein Betriebssystem oder müssen in andere Programme eingebettet werden. Im Unterschied dazu ist eLua eigenständig: Lua-Programme werden vom direkt auf der Hardware laufenden Interpreter ausgeführt und dienen nicht der Erweiterung, sondern dem Zusammenbau von Anwendungen aus existierenden Komponenten. – Der von Ousterhout am meisten hervorgehobene Vorteil von Skriptsprachen. – Außerdem bringt eLua als Einziges eine etablierte Skriptsprache ohne Einschränkung der Funktionalität oder Änderung der Benutzerschnittstelle auf Embedded-Geräte.

1.2 Aufbau dieser Arbeit

In Abschnitt 2 werden das MSB-IOT und eLua vorgestellt. Dabei wird insbesondere auf die für diese Arbeit wichtigen Peripheriegeräte des Boards und Eigenschaften der Programmierumgebung eingegangen. Abschnitt 3 beleuchtet Aspekte der Portierung von eLua auf das MSB-IOT und der Implementierung der Kompatibilitätsschichten zwischen C und Lua. Die Ergebnisse dieser Arbeitsschritte werden in Abschnitt 4 ausgewertet. Dieser enthält auch Beispiele für Lua-Skripte, die vom MSB-IOT mit eLua ausgeführt werden können. Jeweils verbleibende Arbeit wird umrissen. Den Abschluss bilden eine Zusammenfassung der Arbeit und ein Ausblick auf die mögliche weitere Entwicklung von eLua auf dem MSB-IOT.

2 Grundlagen

2.1 MSB-IOT

2.1.1 Allgemeines

Modular Sensor Board (MSB) heißt eine an der Freien Universität Berlin entwickelte Reihe von Sensorboards. Das Attribut »modular« weist auf das Gestaltungsprinzip dieser Geräte hin: Ein Basismodul mit nur grundlegenden Funktionen kann über Buchsen- oder Steckerleisten verschiedenen Zwecken angepasst werden [21]. Diese Idee kennt man beispielsweise auch vom Arduino mit seinen Shields [22].

Abbildung 2.1 zeigt die Vorgänger des MSB-IOT, MSB-430, MSB-430H und MSB-A2. Sie waren hauptsächlich für drahtlose Sensornetzwerke gedacht. Für die Kommunikation untereinander verfügen sie über Funk-Transceiver.

Das MSB-IOT hat ebenfalls einen Funk-Transceiver, den CC1101. Das IOT in MSB-IOT deutet allerdings auf neue Aufgaben hin: Es soll auch *I*nternet *o*f *T*hings-Anwendungen dienen. Deshalb besitzt es zusätzlich einen CC3000-WLAN-Chip und kann sich so ohne Zwischenstellen mit dem Internet verbinden. C1101 und CC3000 werden in den folgenden Abschnitten vorgestellt.

Der Mikrocontroller des MSB-IOT ist ein STM32F415RG [26] von STMicroelectronics mit ARM Cortex-M4-Achitektur (32 bit), 1 MB Flash und 196 kB RAM [13]. In Abbildung 2.2 sieht man weitere Komponenten.

2.1.2 CC1101

Der CC1101 ist ein energieeffizienter Funk-Transceiver von Texas Instruments (TI) [27]. Er funkt im Sub-1 GHz-Band, wird über SPI angesteuert und hat den auf dem MSB-430H und MSB-A2 verwendeten CC1100 abgelöst. Die Unterschiede zwischen den beiden Geräten sind gering [28, 29], aber TI empfiehlt, den CC1100 nicht für neue Hardwareentwürfe zu verwenden [30].

Der CC1101 kann auf mehreren Kanälen funken, unterstützt Wake-on-Radio und eine feste Paketlänge von 255 B. Unter anderem deswegen eignet er sich für Sensornetzwerke und Internet of Things-Anwendungen. Beispielsweise wird er in einem Teil der Knoten des Senslab-Testbeds verwendet [31], sein Vorgänger CC1100 als Teil des MSB-A2 im DES-Testbed [32].

Abbildung 2.1: Die alten Sensorboards der MSB-Reihe: MSB-430, MSB-430H und MSB-A2. Das MSB-430 demonstriert das modulare Design; es ist mit einem Erweiterungs- und einem Trägermodul mit Batteriehalter und serieller Schnittstelle kombiniert. (Quellen: [23, 24, 25])

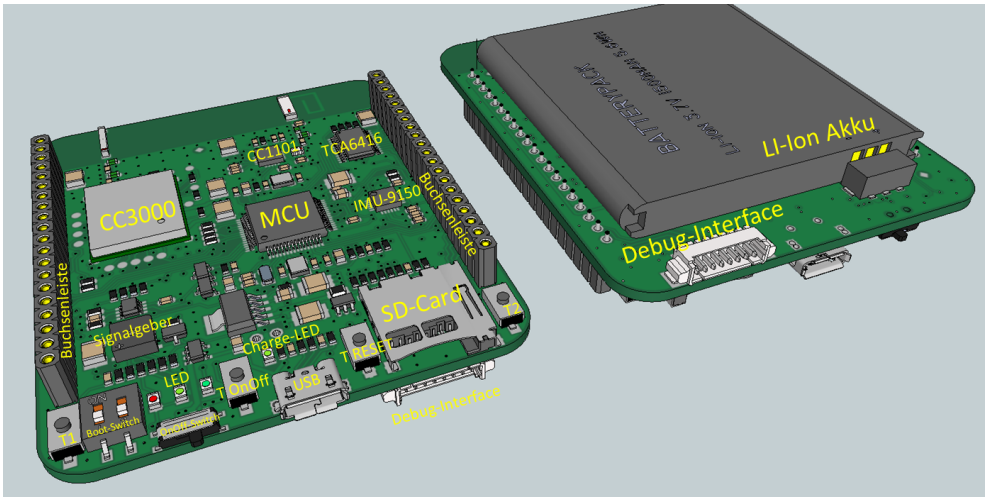

Abbildung 2.2: Das MSB-IOT. (Quelle: [13])

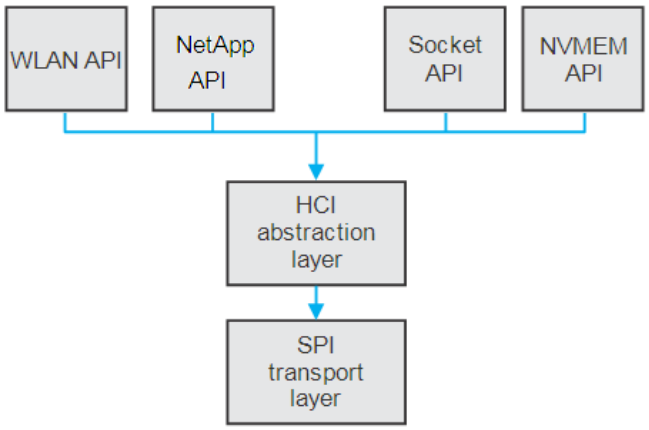

Abbildung 2.3: Schichtenarchitektur der Bibliothek für den CC3000. Über die WLAN-API baut man Verbindungen zu WLANs auf. Über die NetApp-API kann man pingen und manuell oder mit DHCP die Netzwerkadresse des Gerätes konfigurieren. Die Socket-API bietet POSIX-ähnliche TCP- und UDP-Sockets. Über die NVMEM-API kann man bestimmte Einstellungen im EEPROM des CC3000 speichern. (Quelle: [36])

2.1.3 CC3000

Der CC3000 ist ein WLAN-Modul, ebenfalls von TI [33]. Er funkt nach den Normen IEEE 802.11 b/g und verfügt über einen IPv4-Protokollstack sowie weitere Netzwerkdienste wie TCP, DNS und DHCP. Damit entlastet er den Prozessor bei Netzwerkanwendungen. Der CC3000 hat allerdings keine Möglichkeit, IPv6 zu unterstützen [34], und kann sich nur im Infrastrukturmodus verbinden. Das Modul kann also nicht an Ad-hoc-Netzen teilnehmen [35], sondern ist immer auf einen Access Point angewiesen.

Der CC3000 kommuniziert mit dem Mikrocontroller über SPI. TI stellt Entwicklern aber Bibliotheken zur Verfügung, die stark von der Hardware abstrahieren: WLAN-Verbindungsaufbau und Adressvergabe werden über Bibliotheksfunktionen gesteuert; für den Transport Layer gibt es POSIX-ähnliche Sockets. Abbildung 2.3 zeigt die Schichtenarchitektur. Die unterste Schicht muss natürlich der jeweiligen Hardware angepasst werden.

2.1.4 Entwickeln für das MSB-IOT

Um Quelltext für Cortex-M4-Prozessoren zu kompilieren, kann man sich eine eigene Toolchain bauen oder eine der existierenden benutzen [37]. Für diese Arbeit wurde die GCC ARM Embedded-Toolchain [38] benutzt, weil sie unter Open-Source-Lizenz steht und weil sie die Hard Float-ABI der Cortex-M4-Prozessoren unterstützt [39]. – Der kostenlosen Version der in der eLua-Dokumentation empfohlenen CodeSourcery-Toolchain fehlen die entsprechend kompilierten Bibliotheken [40].

Zur Übertragung von Programmen auf den Flash-Speicher besitzt der Prozessor des MSB-IOT eine JTAG-Schnittstelle [26]. Diese wird über ein Gerät namens ST-LINK/V2 von STMicroelectronics [41] mit dem PC verbunden. Die dazugehörige Software vom Hersteller läuft nur unter Windows [42]; es gibt aber auch für Linux Programme unter der Sammelbezeichnung *stlink* [43], die für diese Arbeit benutzt wurden: Mit `st-flash` kann man Binärdateien auf den ROM des STM32F415RG schreiben. `st-util` stellt einen GDB-Server bereit, über den man mit dem GDB aus der Toolchain Programme auf dem MSB-IOT debuggen kann.

Es gibt zwei Möglichkeiten mit Programmen auf dem MSB-IOT seriell zu kommunizieren: Man kann mit Hilfe einer Bibliothek von STMicroelectronics den Micro-USB-Port zu einem CDC-Gerät machen [44]. Oder man nutzt eine der USARTs des Prozessors [26]. Auf dem MSB-IOT sind die Pins der USART2 auf eine Buchsenleiste herausgeführt und können beispielsweise über ein USB-zu-UART-Kabel mit dem PC verbunden werden.

2.2 eLua

Lua ist eine Programmiersprache und eLua, Embedded Lua, eine Lua-Umgebung für Embedded-Geräte. Es besteht aus einem direkt auf der Hardware laufenden modifizierten Lua-Interpreter und einigen Zusätzen für die hardwarenahe Entwicklung [12].

2.2.1 Lua

Lua wird seit 1993 an der Päpstlichen Katholischen Universität von Rio de Janeiro (PUC-Rio) entwickelt und unter der MIT-Lizenz veröffentlicht. Es ist ein mächtige, schnelle, leichtgewichtige und einbettbare Skriptsprache (»powerful, fast, lightweight, embeddable scripting language« [4]). Schnell und leichtgewichtig ist es durch die minimalistische Gestaltung mit einfacher Syntax und zunächst geringem Funktionsumfang.

Trotzdem ist es mächtig und einbettbar, weil es eine erweiterbare Erweiterungssprache (»an extensible extension language« [10]) ist: Einerseits ist der Lua-Interpreter als C-Bibliothek [4, 45] verfügbar. Man kann ihn deshalb in andere Programme einbauen und diese so um Skripting-Funktionalität erweitern. Die Bibliothek ist in ANSI-C geschrieben [46], sodass Portabilität kein Problem ist. Andererseits kann man Lua selber erweitern: Man kann eingebaute Funktionen transparent durch eigene Funktionen ersetzen [47]. Man kann das Verhalten von Datenstrukturen ändern [11] und Lua so beispielsweise um Objektorientierung [48] erweitern. Und man kann wie bei anderen Sprachen Module einbinden. Es existiert auch eine einfache Schnittstelle, über die Entwickler Module für Lua in C/C++ schreiben können [11]. Ferner verfügt der Lua-Interpreter über einen Read-Evaluate-Print-Loop [49], sodass man Programme interaktiv entwickeln kann.

2.2.2 Besonderheiten in eLua

Wegen seiner Leichtigkeit und Portabilität eignet Lua sich zur Arbeit mit Embedded-Geräten. Es benötigt 256 kB ROM und 64 kB RAM [50] und läuft damit problemlos auf dem MSB-IOT. eLua ist also prinzipiell nicht notwendig. Für Geräte mit noch beschränkterer Hardware sind ihm aber ein Garbage Collector für Speichernotsituationen [51] und der sogenannte Lua Tiny RAM-Patch [52] fest eingebaut. Mit diesem kann man Datenstrukturen, auf die nur lesend zugegriffen wird, aus dem RAM in den ROM verlagern. Zu eLua gehören auch Module für den Zugriff auf die Hardware, beispielsweise auf GPIOs, Timer oder SPI [53].

Es gibt mehrere Möglichkeiten, Lua-Programme in eLua auszuführen: Zum einen kann man sogenannte Remote Procedure Calls benutzen. Dabei kommuniziert ein modifizierter Lua-Interpreter auf dem Host-Rechner direkt mit einer eLua-Instanz auf einem Embedded-Gerät. Der Read-Evaluate-Print-Loop von Lua steht dadurch auch für die eLua-Entwicklung zur Verfügung [54]. Zum anderen kann man Lua-Programme im ROM oder beispielsweise auf SD-Karten speichern. Nennt man eines davon `autorun.lua`, führt eLua es direkt aus [55]. Alternativ kann man über UART mit einer eLua-Shell kommunizieren und von dort aus die gespeicherten Programme aufrufen.

2.2.3 Architektur von eLua

Um viele Embedded-Plattformen unterstützen zu können, hat eLua eine Schichten-Architektur [55] (Abbildung 2.4). Ein großer Teil der obersten Schicht ist allen Plattformen gemein (»Common Code«). Er enthält beispielsweise den gepatchten Lua-Interpreter, die eLua-Shell, die generischen Module und Dateisystemtreiber.

Um plattformunabhängig zu sein, nutzt der allgemeine Code nur Funktionen der Plattformschnittstelle. Diese Schnittstelle abstrahiert von allen plattformspezifischen Details wie beispielsweise der Initialisierung des Systems oder dem Zugriff auf die GPIOs [56].

Damit eLua auf einer Plattform laufen kann, müssen Teile der Plattformschnittstelle für diese Plattform implementiert werden. Um Lua-Programmen spezielle Funktionen einer Plattform oder eines Boards, beispielsweise Zugriff auf LCDs, zur Verfügung zu stellen, kann man außerdem eigene Plattformmodule schreiben. Diese Arbeit stellt genau solche Plattformmodule für den Funk-Transceiver und das WLAN-Modul des MSB-IOT vor.

2.2.4 eLua Bauen

Ursprünglich verwendete eLua SCons als Build-System; seit Version 0.10 benutzt man Lua-Skripte [57]. Der Umstieg hat sich aber fast nur auf Entwickler von eLua selbst ausgewirkt. Beispielsweise funktioniert die Anpassung an neue Plattformen jetzt anders: Wo man eLua früher beim Portieren über `.h`-Dateien konfigurierte, geschieht das

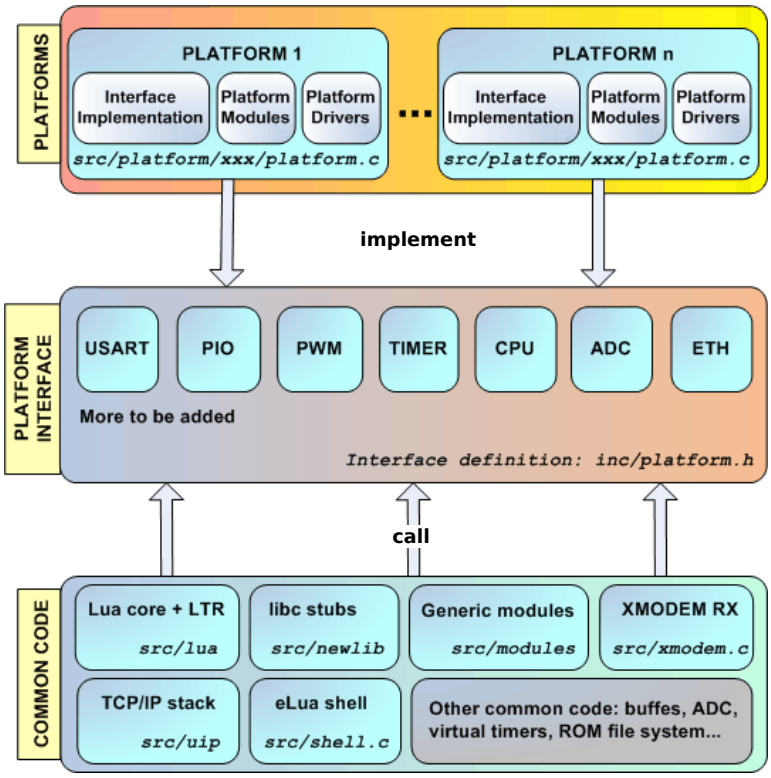

Abbildung 2.4: Architektur von eLua. Lua-Interpreter, generische Module, Dateisysteme etc. sind für alle Plattformen gleich. – Sie greifen nur auf die Schnittstelle zu, die von allen Plattformen implementiert wird. (Quelle: [55], Beschriftungen der Pfeile ergänzt.)

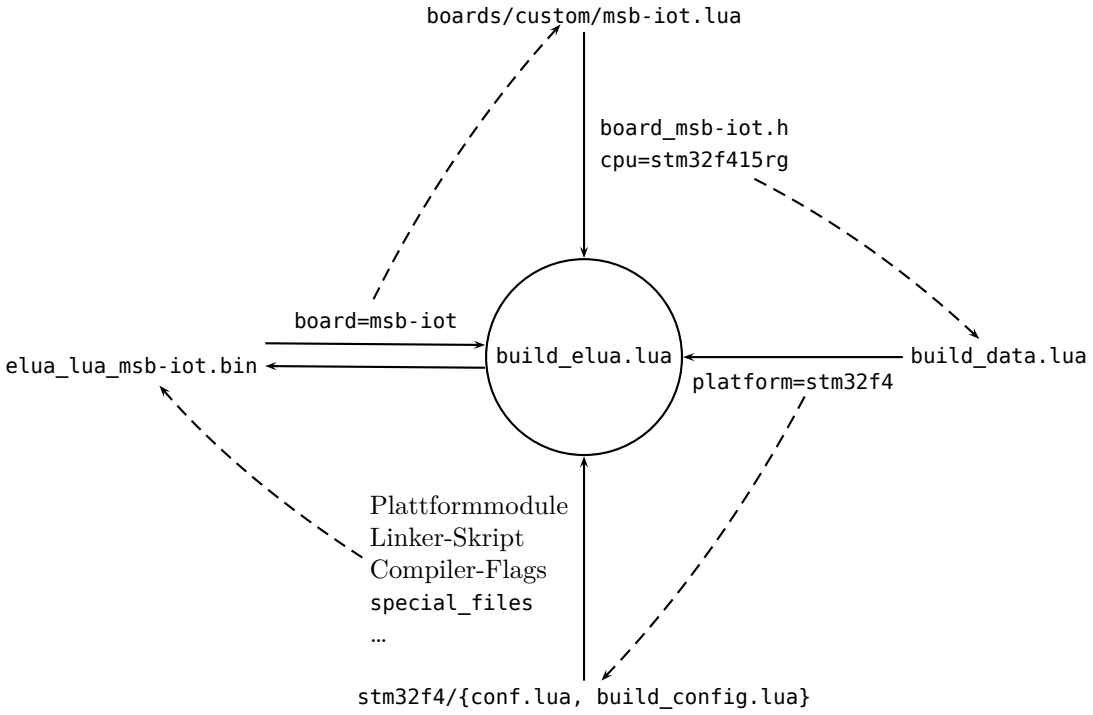

Abbildung 2.5: Der Build-Prozess von eLua: Das Build-Skript wird mit dem Parameter `board` aufgerufen, lädt die entsprechende Konfigurationsdatei, erfährt dort den Zielprozessor, bestimmt aus `build_data.lua` die Zielplattform, findet deren Konfigurationsdateien mit Informationen über einzubindenden Quelltext und Programme der Toolchain und baut damit die ausführbare Datei.

nun über Lua-Skripte, die während des Build-Prozesses eingebunden werden. Das Zusammenspiel der Konfigurationsskripte ähnelt der Architektur von eLua: Um einen Build für ein bestimmtes Board beziehungsweise Embedded-Gerät zu beginnen, ruft man `build_elua.lua` mit dem Namen des Boards als Argument auf. Um sich über das Board zu informieren, liest `build_elua.lua` dessen Konfigurationsskript ein. Unter anderem findet es heraus, welche Module benötigt werden und welche Prozessorplattform das Board benutzt. Es führt deren Konfigurationsskripte aus und baut mit den gesammelten Informationen die ausführbare eLua-Datei. Siehe auch Abbildung 2.5.

2.2.5 Portierungen und Einsatz von eLua

Auf der Status-Seite von eLua [58] findet man eine Liste der Plattformen und Boards, auf die eLua bereits portiert wurde. Darunter ist beispielsweise die mbed-Plattform; das EK-LM3S8962 verfügt über ein Display, wofür es auch ein eLua-Plattformmodul gibt [59]; die Mizar32-Boards werden sogar mit eLua ausgeliefert [60]. Diese Prototyping- beziehungsweise Evaluationsboards werden natürlich eher im Hobby- oder Bildungsbereich eingesetzt. Aber genauso wie Hardwareentwickler von jenen Boards ausgehend eigene entwickeln, können sich Softwareentwickler an bestehenden eLua-Portierungen orientieren.

Das eLua-Wiki führt außerdem eine Reihe von eLua-basierten Projekten auf [61]. Darunter sind Spiele, aber auch ein Segway-Klon und ein Roboter von der PUC-Rio. Außerdem gibt es einen Webserver – nützlich, um Hardware über den Browser zu verwalten – und eLuaBrain. eLuaBrain ist ein eigenständiger Computer, größtenteils programmiert in Lua [62] von Bogdan Marinescu, einem der eLua-Autoren [63].

2.2.6 C-Module für eLua

Die zentrale Datenstruktur in Lua ist die Tabelle [11, 64], in anderen Sprachen Hash, assoziatives Array, Dictionary oder Map genannt. Tabellen sind unter anderem Basis für gewöhnliche Arrays, Variablen- und Funktionslookups und für die in Abschnitt 2.2.1 erwähnten Erweiterungsmechanismen, also auch für Module: Wenn man ein Modul lädt, gibt es eine Tabelle zurück, die die Namen der Funktionen des Moduls auf die Funktionen (First-Class-Werte) selber abbildet. Man kann sie dann als `<tabellenname>['<funktionsname>'](<argumente>)` oder alternativ als `<tabellenname>.<funktionsname>(<argumente>)` aufrufen.

Will man ein Modul für Lua in C programmieren, muss man die Funktionen nach einem bestimmten Muster schreiben [11]. Insbesondere ist das einzige Argument dieser Funktionen ein Zeiger auf eine Stack-Datenstruktur. Diese dient dem Austausch von Argumenten und Rückgabewerten zwischen C- und Lua-Code. Außerdem muss das Modul eine Funktion des Namens `luaopen_<modulname>` haben, die das Modul initialisiert

und seine Funktionen beim Lua-Interpreter registriert. Wenn man das Modul nun als Shared Library kompiliert, kann man es einbinden als wäre es in Lua geschrieben [64].

C-Module in eLua zu benutzen funktioniert prinzipiell genauso. Mit dem Lua Tiny RAM-Patch kann man jedoch die Modultabelle bei Bedarf vom RAM in den ROM verlagern [52]. Deswegen sind die Konventionen für die `luaopen_*`-Funktion hier geringfügig anders. Außerdem muss man Module dem Build-System bekannt machen. Dazu gibt es für jede Plattform ein Skript `build_config.lua` mit der Funktion `get_platform_modules`, die eine Liste der Module für diese Plattform zurückgeben sollte. Wenn man dann noch in der der Board-Konfigurationsdatei dem Schlüssel `platform_name` einen Wert zuweist, kann man Funktionen eines Moduls als `<plattformname>.<modulname>.<funktionsname>(<argumente>)` aufrufen, ohne das Modul explizit zu laden.

Quelltext 2.1 ist ein minimales Beispiel für ein Plattformmodul. Die Signaturen aller von Lua aus aufrufbarer Funktionen müssen der der Funktion `msbiot_example_addfive` (Zeilen 8 bis 14) gleichen. Diese entnimmt dem Parameterstack ein Argument, legt ihren Rückgabewert darauf ab und zeigt die Anzahl zurückgegebener Werte mit `return 1` an. Danach stehen die Modultabelle (Zeilen 16 bis 19) und die Registrierfunktion `msbiot_luaopen_example`.

2.2.7 Präprozessor und Interrupts

Die Konfigurationsskripte des eLua-Buildsystems erlauben nicht nur, je nach Board unterschiedliche Dateien, sondern auch verschiedene Teile des Quelltextes bei der Kompilation einzubeziehen. Dazu übergeben sie dem C-Präprozessor mit der Kommandozeilenoption `-D` Makrodefinitionen bezüglich Prozessor, Prozessorplattform und Board, die dann über `#ifdef` abgefragt werden können.

Das ist beispielsweise wichtig für Interrupt Service Routines (ISRs), denn mit dem gleichen GPIO des Prozessors sind meist je nach Board verschiedene Peripheriegeräte verbunden, deren Interrupts das System unterschiedlich verarbeiten muss. eLua ermächtigt den Programmierer außerdem, Interrupt-Handler in Lua zu registrieren. In der Datei `platform_int.c` sind dazu ISRs für alle Timer-, USART- und GPIO-Interrupts definiert. Sie können mit Hilfe des Präprozessors leicht ersetzt werden, wenn, wie in dieser Arbeit bei CC1101 und CC3000, ein Hardwaretreiber über Interrupts mit Peripheriegeräten kommuniziert.

```c
#include "lua.h"
#include "lauxlib.h"

#include "lrotable.h"
#define MIN_OPT_LEVEL 2
#include "lrodefs.h"

static int msbiot_example_addfive(lua_State *L)
{
    int n = luaL_checkint(L, 1);

    lua_pushinteger(L, n + 5);
    return 1;
}

const LUA_REG_TYPE msbiot_example_map[] = {
    { LSTRKEY("addfive"),   LFUNCVAL(msbiot_example_addfive)    },
    { LNILKEY,              LNILVAL                             },
};

LUALIB_API int msbiot_luaopen_example(lua_State *L)
{
    LREGISTER(L, "example", msbiot_example_map);
}
```

Quelltext 2.1: Minimalbeispiel für ein Plattformmodul: Die Funktion `msbiot_example_addfive` kann aus Lua heraus aufgerufen werden und gibt die Summe aus ihrem Argument und 5 zurück.

3 Durchführung

Dieses Kapitel dokumentiert die Portierung von eLua auf das MSB-IOT und die Arbeiten, die Lua-Programmen den Zugriff auf Funk-Transceiver CC1101 und WLAN-Modul CC3000 jenes Boards ermöglichen. In allen Fällen konnte bestehende Software genutzt werden: Das Git-Repository von eLua enthält ein Portierung für das STM32F4 Discovery-Board. Für den CC1101 existierte bereits eine in C geschriebene Treiberbibliothek. Zum CC3000 gehört eine umfangreiche Bibliothek vom Hersteller, für die auch schon MSB-IOT-spezifische Anpassungen und Zusätze vorhanden waren.

Im Rahmen dieser Arbeit wurde die Portierung für das STM32F4 Discovery-Board für das MSB-IOT adaptiert. Außerdem entstanden Korrekturen, Plattformmodule und Testskripte für die Treiberbibliotheken und die MSB-IOT-Portierung von eLua. Sämtliche Quelltexte und eine Dokumentation des Arbeitsprozesses befinden sich in einem Git-Repository auf der beigelegten CD.

3.1 Portierung von eLua auf das MSB-IOT

eLua soll auf das MSB-IOT portiert und damit Folgendes erreicht werden: Ein Interpreter läuft auf dem Board und kann Lua-Skripte ausführen. Lua-Skripte können über die in eLua integrierten Module auf die Hardware, insbesondere auf GPIOs und Timer, zugreifen. Entwickler können über USB und UART mit der eLua-Shell kommunizieren. Sie können außerdem Plattformmodule für das MSB-IOT schreiben und beliebige als Quelltext vorliegende Bibliotheken einbinden. Plattformmodule und Bibliotheken sollen nur für das MSB-IOT kompiliert werden und nicht für andere Boards.

eLua auf ein Board zu portieren bedarf im Allgemeinen dreier Schritte [65]: Die Portierung auf die Prozessorplattform des Boards, die Anpassung an den konkreten Prozessor und die Konfiguration für das Board selber. Hauptansatzpunkt für den ersten Schritt ist die Plattformschnittstelle. – Sind deren Funktionen für eine Prozessorplattform implementiert, funktionieren automatisch Interpreter, Shell und die Module für den Hardwarezugriff [55]. Für viele Prozessorplattformen existieren allerdings schon eLua-Portierungen; so auch für die des MSB-IOT: eLua läuft schon auf dem STM32F4 Discovery-Board [66, 67]. Damit ist insbesondere die Plattformschnittstelle für alle Prozessoren der STM32F4-Reihe implementiert. Diese Prozessoren sind sich so ähnlich [26, 68, 69], dass der STM32F415RG des MSB-IOT lediglich geringste Anpassungen erfordert.

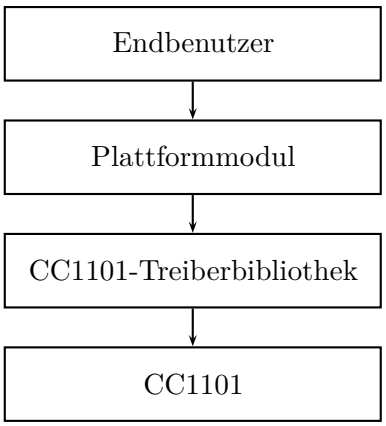

Abbildung 3.1: Architektur der Ansteuerung des CC1101: Aus C-Code heraus kann man über den CC1101-Treiber Pakete mit dem CC1101 senden und empfangen. Der Treiber kommuniziert mit dem CC1101 über SPI. Die Vermittlungsschicht zwischen C und Lua bildet ein Plattformmodul.

Um im letzten Schritt eLua für ein bestimmtes Board einzurichten, müssen Entwickler grundsätzlich nur eine Konfigurationsdatei schreiben [70]. Hier müssen sie die Zusammensetzung des Boards angeben, wie beispielsweise die Frequenz externer Taktquellen. Außerdem können sie neben anderen Dingen einstellen, welche Module mitkompiliert werden sollen und über welche Schnittstelle die eLua-Shell zu erreichen ist. Die Konfigurationsdatei des MSB-IOT ist an die des STM32F4 Discovery-Boards angelehnt, weil die beiden sich nicht nur in den Prozessoren, sondern auch im restlichen Aufbau ähneln [13, 67].

Die letzte Anforderung war, dass die nachfolgend beschriebenen Plattformmodule und die von ihnen benutzten Bibliotheken nur für das MSB-IOT und nicht für andere Boards der STM32F4-Plattform kompiliert werden. Zu diesem Zweck wurden die Build-Skripte dieser Plattform um Kontrollstrukturen ergänzt, die je nach Board unterschiedliche Dateien in den Build-Prozess einbeziehen.

3.2 Ansteuerung des CC1101

Ziel ist, aus Lua-Programmen heraus Pakete mit dem CC1101 senden und empfangen zu können. Abbildung 3.1 zeigt die Architektur der resultierenden Software; der Raum zwischen CC1101 und Endbenutzer könnte aber auch anders gefüllt werden als dort zu sehen. – Prinzipiell braucht man nur ein Modul, das auf der einen Seite aus Lua aufrufbare Funktionen anbietet und auf der anderen Seite mit dem CC1101 kommuniziert. Man hat also Gestaltungsfreiheit über die Schnittstelle zu Lua, die Kommunikation mit dem CC1101 und das Innere des Moduls.

3.2.1 Kommunikation mit dem CC1101

Man kann den CC1101 so einrichten, dass er einen seiner Ausgänge auf 1 schaltet, wenn er ein Paket empfangen hat [27]. Der Treiber könnte diesen Ausgang wiederholt abfragen und gegebenenfalls das Paket entgegennehmen. Insbesondere im Akkubetrieb ist das ungünstig, weil der Prozessor durchgehend beschäftigt ist, ohne Ergebnisse zu liefern (aktives Warten). Besser ist es, ihn so zu konfigurieren, dass die Pegeländerung am CC1101 einen Interrupt auslöst. Das erlaubt dem Prozessor, in einen Schlafzustand überzugehen und erst auf den Interrupt hin wieder aufzuwachen und das Paket abzuholen.

Pakete Abholen und ein Großteil der übrigen Kommunikation erfolgt über die SPI-Schnittstelle des CC1101. In eLua gibt es ein Modul für SPI [71]; den Treiber direkt in Lua zu implementieren, wäre also möglich. Ob das im Hinblick auf Performanz und Energieverbrauch sinnvoll ist, müsste man untersuchen. – Zumindest könnte man aber einen Prototyp in Lua entwickeln und gegebenenfalls in C nachbauen. Letztendlich bestimmt aber die Kommunikation mit dem CC1101 die Komplexität des Treibers, weshalb Lua statt C die Entwicklung kaum erleichtern oder beschleunigen würde. Außerdem verlöre man den Vorteil wieder, den der im vorigen Absatz beschriebene Interrupt-gesteuerte Betrieb bringt. In eLua kann man nämlich derzeit nur auf Interrupts reagieren, während der Interpreter Instruktionen ausführt [72]. Und während der Interpreter Instruktionen ausführt, kann der Prozessor nicht schlafen. Insgesamt erscheint es also am günstigsten, den Treiber in C zu programmieren oder, wie in diesem Fall, einen bereits vorhandenen zu nutzen.

3.2.2 Ergänzung des Treibers

Der vorhandene Treiber bietet Funktionen zum An- und Abschalten des CC1101, Einstellen von Sendeleistung, Funkkanal und einer primitiven Adresse sowie zum Senden von Paketen, nicht aber zum Empfangen. – Eintreffende Pakete werden in der ISR entgegengenommen und auf einer seriellen Schnittstelle ausgegeben. Für aufrufenden Code wäre aber eine blockierende Empfangsfunktion günstig: Eine Funktion, die kürzlich eingegangene Pakete zurückgibt und, falls keine eingegangen sind, blockiert bis welche eingehen.

Dazu ist ein Puffer nötig, wohinein die ISR schreibt und wo heraus die Empfangsfunktion liest. Weil die ISR die Empfangsfunktion unterbrechen kann, muss dieser Puffer sicher gegenüber nebenläufigen Zugriffen sein. Bei einer Kapazität von einem Paket bräuchte man Sperrsynchronisation; das ist aber ohnehin nicht sinnvoll, weil die Empfangsfunktion ein Paket fertig bearbeitet haben müsste, bevor das nächste eintrifft.

Für Puffer höherer Kapazität besitzt der Prozessor einen DMA-Controller [26], der die Pakete direkt vom CC1101 holen und in den Arbeitsspeicher schreiben könnte.

Die Hauptrecheneinheit würde dieser Aufgaben entbunden. Der DMA-Controller der STM32F4-Prozessoren unterstützt Warteschlangen und Doppelpufferung. Andererseits ist der erforderliche Code nicht portabel, man benötigt einen weiteren Interrupt und muss darauf achten, dass es keine Kollisionen mit anderen DMA-Kanälen gibt. Die Komplexität des Systems erhöht sich also deutlich.

Deshalb wurde vorerst auf DMA verzichtet und der Treiber mit einer in Software implementierten Warteschlange versehen. Sie ist auch ohne Sperrsynchronisation sicher gegenüber nebenläufigen Zugriffen.

3.2.3 Plattformmodul

Den ergänzten Treiber können zwar in C, nicht aber in Lua geschriebene Programme nutzen. Deshalb gibt es, wie in Abbildung 3.1 gezeigt, eine zusätzliche Schicht zwischen diesem Treiber und dem Endbenutzer: Das Plattformmodul ist ein Adapter, der auf der einen Seite eine Lua-Schnittstelle bietet und auf der anderen Seite C-Funktionen aufruft. Der Mechanismus dazu ist von Lua vorgegeben, wie in Abschnitt 2.2.6 beschrieben; Hauptaufgabe der Funktionen des Plattformmoduls ist, Daten zu konvertieren: Sie müssen die vom Lua-Programm übergebenen Argumente in eine geeignete Form überführen, um damit die Funktionen der Treiberbibliothek aufzurufen. Deren Rückgabewerte werden wiederum Lua-gerecht verpackt.

Die API des Plattformmoduls ist gegenüber der der Treiberbibliothek vereinfacht: Die Funktionen zum Hochfahren des CC1101 und Einstellen von Kanal, Sendeleistung und Adresse sind zu einer Initialisierungsfunktion zusammengefasst. Das spart dem Programmierer zwei Funktionsaufrufe und macht außerdem klarer ablesbar, wie der CC1101 gerade konfiguriert ist. Mit dieser API kann man auch beliebig viele Pakete auf einmal senden und empfangen, bei der Treiberbibliothek immer nur eines. Die Funktion zum Ausschalten des Transceivers gibt es weiterhin.

3.3 Ansteuerung des CC3000

3.3.1 Zielsetzung

Die Aufgabenstellung beim CC3000 ähnelt der beim CC1101: Es existiert eine in C geschriebene Treiberbibliothek [36, 73] für das WLAN-Modul. Auch Lua-Programme sollen WLAN-Verbindungen aufbauen können. Also ist ein Plattformmodul als Vermittlungsschicht über dem Treiber nötig (Abbildung 3.2).

Der Unterschied zwischen CC1101 und CC3000 ist aber enorm: Der CC1101 kann Pakete von Gerät zu Gerät senden, stellt also nach dem TCP/IP-Modell Funktionen des Link Layers zur Verfügung. Der CC3000 ist dagegen ein Netzwerkprozessor und implementiert den Link Layer (WLAN-Scan und Verbindung mit WLAN-Access Points

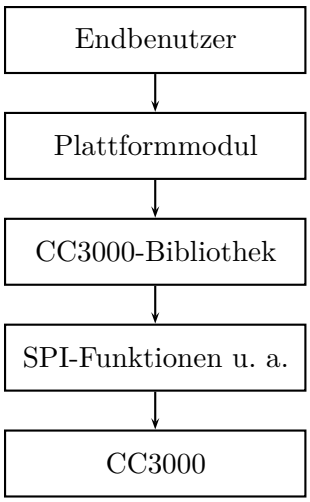

Abbildung 3.2: Architektur der Ansteuerung des CC3000: Aus C-Code heraus kann man sich über die zum CC3000 gehörende Bibliothek mit WLANs verbinden, das Modul konfigurieren, über Sockets Daten austauschen etc. Diese Bibliothek kommuniziert mit dem CC3000 über SPI. Die plattformspezifischen Details dieser Kommunikation verbirgt eine Abstraktionsschicht. Das Plattformmodul vermittelt zwischen der Lua-Welt des Endbenutzers und der C-Welt der Bibliothek des CC3000.

verschiedener Sicherheitsstufen), den Internet Layer (manuelle oder automatische Registrierung im Netzwerk), den Transport Layer (TCP- und UDP-Sockets zur Interprozesskommunikation) und den Application Layer (DNS) [33]. Auf alle diese Funktionen bietet die Treiberbibliothek Zugriff und ist entsprechend umfangreich. Gemäß dem Anspruch dieser Arbeit, Grundlage und Orientierung für weitere Entwicklungen zu sein, soll das Plattformmodul deshalb vorerst nur einen Teil der Möglichkeiten des CC3000 an Lua weiterreichen, gleichzeitig aber Ausgangspunkt für eine schrittweise Erweiterung sein. Es implementiert daher einen vertikalen Schnitt durch die Funktionalität der Treiberbibliothek beziehungsweise den TCP/IP-Stack: Mit ihm können sich Lua-Programme mit WPA2-WLANs verbinden, per DHCP eine IP-Adresse erhalten und über UDP-Sockets mit entfernten Prozessen kommunizieren.

3.3.2 Gestaltung der API

Der größte Teil der API des Plattformmoduls bezieht sich auf Sockets zur Interprozesskommunikation. Die meisten Programmiersprachen verfügen über eine Implementierung dazu und sich an einer solchen zu orientieren ist sinnvoll, damit Programmierer Gewohntes vorfinden. In C orientiert man sich an POSIX- beziehungsweise BSD-Sockets, wie man auch der Treiberbibliothek des CC3000 ansieht. In Lua gibt es

verschiedene Ansätze: Man kann sich C zum Vorbild nehmen und das Lua-Äquivalent zur POSIX-API konstruieren; so ist es beim *net*-Modul von eLua [74]. Oder man kann wie LuaSocket [75] auf die Abstraktionsmechanismen von Lua setzend etwas Neues schaffen: LuaSocket nutzt C-Module, um die grundlegenden Netzwerkfunktionen an Lua durchzureichen, und umgibt sie mit einer objektorientierten API. Das in dieser Arbeit vorgestellte Modul soll aber bloß die Machbarkeit eines WLAN-Moduls nachweisen und Grundlage für weitere Entwicklungen sein. Es implementiert daher die simple API des genannten *net*-Moduls. Ein zusätzlicher Vorteil dieses Ansatzes ist, dass Nutzer dieses Moduls ohne Codeänderungen auch das Modul für den CC3000 verwenden können, sobald der Chip initialisiert und verbunden ist.

3.3.3 Bemerkungen zur Implementierung

Die Treiberbibliothek wurde nur in geringem Umfang geändert: Zwei blockierende Funktionen kapseln das Warten auf eine WLAN-Verbindung und Adresszuweisung per DHCP. Außerdem wurde ein sehr primitives Logging-System eingeführt. Mit diesem kann man die ständigen Debugausgaben des Treibers ein- und auch ausschalten, damit sie beispielsweise nicht die Kommunikation mit der eLua-Shell stören.

Das Plattformmodul, mit dessen API sich schon der vorherige Abschnitt befasst hat, dient wie beim CC1101 (vergleiche Abschnitt 3.2.3) als Adapter zwischen C-Bibliothek und Lua-Programmen. Interessant ist hier die Verwendung von sogenannten *Benutzerdaten* (»full userdata« [11]): Die Funktion `connect` benötigt als Argument den Rückgabewert von `packip`. Dabei handelt es sich aber um ein Struct (Quelltext 3.1, Zeilen 10 bis 17), das in Lua nicht durch einen gewöhnlichen Datentyp repräsentiert werden kann. Es wird daher als Benutzerdatum verpackt (Zeile 9). Auf dessen Inhalt hat das aufrufende Lua-Programm keinen Zugriff; er kann nur von anderen Funktionen des Plattformmoduls weiterverwendet werden (Zeilen 31 und 35).

3.3.4 Probleme

Gegenwärtig kann man mit dem CC3000 keine WLAN-Verbindung aufbauen, solange die USB-Schnittstelle aktiv ist. – Die Funktion zur Initialisierung des Moduls blockiert dann aus unbekannter Ursache. Der Urgrund für dieses Versagen ist, dass drei große Programmkomponenten ohne ausreichende Kenntnis ihres Inneren zusammengesetzt wurden: eLua, der USB-Treiber von STMicroelectronics und die Treiberbibliothek des CC3000. Den Defekt in dem resultierenden Konglomerat zu finden, war im Rahmen dieser Arbeit nicht möglich. Wahrscheinlich handelt es sich aber um eine Wechselwirkung aller drei Teile, denn beliebige zwei funktionieren zusammen: eLua mit USB-Kommunikation, ein gleichzeitig auf USB und CC3000 zugreifendes Testprogramm und eLua mit CC3000, bei dem über UART statt USB kommuniziert wird. Diese letzte Konstellation entspricht zwar einem Ziel dieser Arbeit, eLua und CC3000 zusam-

```
/*  [...]
 * Returns:
 * 1    an internal representation of the given IPv4 address
 */
static int msbiot_cc3000_packip(lua_State *L)
{
    [...]
    // Fill the IP address part of a sockaddr struct with the given value
    sockaddr *ip_addr = lua_newuserdata(L, sizeof(sockaddr));
    ip_addr->sa_family = AF_INET;
    int err = sscanf(
        ip_string,
        "%hhd.%hhd.%hhd.%hhd",
        &( ip_addr->sa_data[2] ),
        &( ip_addr->sa_data[3] ),
        &( ip_addr->sa_data[4] ),
        &( ip_addr->sa_data[5] )
    );
    [...]
    return 1; // The userdata is pushed onto the stack when it's created.
}

/*  [...]
 * 2 dest_ip    the IP address of whom the data shall be sent to as
 *              returned by msbiot_cc3000_packip()
 *  [...]
 */
static int msbiot_cc3000_connect(lua_State *L)
{
    [...]
    sockaddr *dest_addr = (sockaddr *) lua_touserdata(L, 2);
    [...]

    // Set up connection/association
    int conn_err = connect(sock_fd, dest_addr, sizeof(sockaddr));
    [...]
}
```

Quelltext 3.1: Dieser Ausschnitt aus dem Plattformmodul für den CC3000 demonstriert den Umgang mit der Lua-API und sogenannten Benutzerdaten.

menzubringen, ist aber nicht optimal: Um die eLua-Shell nutzen zu können, brauchen Entwickler zusätzlich zum Micro-USB- ein USB-zu-UART-Kabel, das im Vergleich zu jenem nicht gebräuchlich ist. Den Defekt zu finden, wäre also wünschenswert; die nachfolgend dokumentierten Erkenntnisse sollen bei der weiteren Suche helfen. Die dieser Arbeit beigefügte CD enthält noch detailliertere Informationen.

Man führt das Versagen herbei, indem man in der Board-Konfigurationsdatei USB CDC als Weg der seriellen Kommunikation auswählt. Für das Debugging sollte außerdem die Compileroption `-Os` ausgeschaltet sein, da sie gelegentlich zu schwer erklärbarem Verhalten führt.

Die Pin-Überschneidung zwischen CC3000 und USB-Subsystem ist als Versagensursache ausgeschlossen: Das Modul löst zwar im Zuge der Kommunikation mit dem Prozessor Interrupts auf Pin PA10 aus [76], der auch mit einem Kontakt der Micro-USB-Buchse verbunden ist [26]. Dieser dient aber nur der Unterscheidung zwischen Micro-USB-A- und -B-Steckern [77], sodass die Initialisierung des Pins aus dem USB-Treiber entfernt werden kann. Das ermöglicht in dem Testprogramm die Koexistenz von WLAN- und USB-Verbindung, nicht aber in eLua.

Entfernt man allerdings aus dem USB-Treiber die Initialisierung des `OTG_FS_IRQn`-Interrupts, tritt das Versagen nicht mehr auf. Das und verschiedene weitere Versuche lassen vermuten, dass Konflikte zwischen Interrupts des USB- und des WLAN-Subsystems das Versagen verursachen. Erster Ansatzpunkt sind dabei die Interrupt-Prioritäten. Entwickler sollten deren Konfiguration für die Cortex-M4-Architektur [78] verstehen: Jeder Interrupt hat eine Präemptions- und eine Subpriorität (`PreemptionPriority` beziehungsweise »group priority« und `SubPriority`). Interrupts mit einer höheren Präemptionspriorität können Interrupts mit einer niedrigeren Präemptionspriorität unterbrechen (Nested Interrupts). Die Subpriorität beeinflusst nur die Reihenfolge, in der wartende Interrupts abgearbeitet werden: Ein Interrupt mit höherer Subpriorität kann einen Interrupt mit niedrigerer Subpriorität überholen, aber nicht unterbrechen. Für Präemptions- und Subpriorität eines Interrupts stehen insgesamt vier Bit zur Verfügung und man kann festlegen, wie viele Bit für welche Prioritätsart vergeben werden.

eLua setzt die Anzahl der Bits für die Präemptionspriorität auf 0. Das heißt, die Präemptionspriorität wird ignoriert, Interrupts können einander nicht mehr unterbrechen und werden nur nach der Subpriorität geordnet. Bei der Integration des USB-Treibers wurde nicht beachtet, dass dieser die Anzahl der Bits für die Präemptionspriorität wieder auf 1 setzt. Die Treiberbibliotheken für CC3000 und CC1101 wurden anscheinend ohne Beachtung der Prioritätsmechanismen programmiert. Entsprechende Korrekturen beheben allerdings nicht das Problem mit dem WLAN-Verbindungsaufbau.

4 Auswertung

4.1 Funktionstest für eLua auf dem MSB-IOT

In Abschnitt 3.1 werden die Anforderungen an die eLua-Portierung aufgezählt: Der Interpreter soll auf dem MSB-IOT Lua-Skripte ausführen. Diese können auf die Hardware zugreifen. Die Kommunikation über USB und UART und der Mechanismus für Plattformmodule funktionieren. Die in Quelltext 4.1 gezeigte eLua-Shell-Sitzung demonstriert, wie diese Anforderungen umgesetzt wurden.

In den Zeilen 2 bis 9 sieht man den Aufruf eines Shell-Befehls. In Zeile 11 wird eine der ausgegebenen Dateien ausgeführt. Es handelt sich dabei um ein Skript, das eine LED des MSB-IOT blinken lässt, bis der Benutzer auf der UART-Schnittstelle ein Zeichen eingibt. So wird die Funktionstüchtigkeit dreier Module für den Hardwarezugriff überprüft: Timer, GPIOs und serielle Kommunikation über UART.

Schließlich wird in Zeile 13 der interaktive Lua-Interpreter gestartet. Die in Zeile 16 aufgerufene Funktion wurde in Abschnitt 2.2.6 in dem Beispiel für ein minimales Plattformmodul definiert.

4.2 CC1101

4.2.1 Funktionstest

Das Plattformmodul zum CC1101 wurde geschaffen, damit Lua-Programme über diesen Funk-Transceiver Daten in Form von Paketen mit anderen Geräten austauschen können. Der für den CC1101 vorhandene Treiber bietet auch Funktionen, mit denen man den Funkkanal, eine Art Adresse und die Sendeleistung des Transceivers setzen kann. Das Plattformmodul ermöglicht Lua-Programmen, auch diese Einstellungen vorzunehmen.

Um die Funktionstüchtigkeit des Plattformmoduls zu prüfen, wurde ein MSB-430H als Funkgegenstelle programmiert. Es sendet im Sekundentakt fortlaufend nummerierte Pakete und gibt empfangene Pakete auf der UART-Schnittstelle aus. Auf dem MSB-IOT läuft ein Lua-Skript, das eingehende Pakete ebenfalls ausgibt und dem Absender dann eine Antwort schickt. Da die Sende- und Empfangsfunktionen für das Senden und Empfangen mehrerer Pakete auf einmal ausgelegt sind, ruft das Lua-Skript sie mehrmals mit aufsteigender Paketanzahl auf. Die in Quelltext 4.2 gezeigten Ausschnitte aus dem Programm demonstrieren die API des Plattformmoduls.

```
1  eLua v0.9-199-g4d27084  Copyright (C) 2007-2013 www.eluaproject.net
2  eLua# ls
3  /rom
4    wlan-test.lua                1006 bytes
5    uart-test.lua                79 bytes
6    wlan.lua                     1764 bytes
7    blink.lua                    672 bytes
8    radio-test.lua               1887 bytes
9  Total on /rom: 5408 bytes
10
11 eLua# lua /rom/blink.lua
12 Press CTRL+Z to exit Lua
13 eLua# lua
14 Press CTRL+Z to exit Lua
15 Lua 5.1.4  Copyright (C) 1994-2011 Lua.org, PUC-Rio
16 > = msbiot.example.addfive(4)
17 9
18 >
```

Quelltext 4.1: Eine Sitzung mit der eLua-Shell. Zuerst werden die verfügbaren Dateien aufgelistet, dann eine davon ausgeführt und schließlich im interaktiven Lua-Interpreter eine Funktion eines Plattformmoduls aufgerufen

Der Empfangs- und Sendelauf wird einmal mit voller und einmal mit reduzierter Sendeleistung ausgeführt, um diese Einstellung zu erproben. Daraufhin verringert sich wie erwünscht der RSSI-Wert auf dem MSB-430H. Dass bei unpassender Kanal- und Adresskonfiguration keine Kommunikation zustande kommen kann, wird durch Aufrufe von Funktionen des Plattformmoduls im interaktiven Lua-Interpreter sichergestellt.

4.2.2 Verbleibende Arbeit

Das Plattformmodul reicht alle Funktionen des Treibers an Lua weiter. Das heißt, Lua-Programme können Pakete verschicken und empfangen, Funkkanal und Sendeleistung einstellen sowie Pakete auf eine primitive Weise adressieren. Der Treiber bietet aber bei Weitem nicht Zugriff auf die gesamte Funktionalität des CC1101 [27]: Um Energie zu sparen, kann man den CC1101 beispielsweise in den Wake-on-Radio-Modus versetzen. Netzwerkstandards wie 802.15.4 verlangen eine minimale Paketgröße von 127 B [79], die zwar der CC1101 unterstützt, aber nicht der Treiber. Dieser müsste also weiterentwickelt und das Plattformmodul entsprechend angepasst werden.

Aber auch das Plattformmodul bedarf der Erweiterung: Aufrufender Code muss empfangene Pakete selbst mit einer sogenannten Metamethode versehen (Quelltext 4.2, Zeilen 4 bis 11). Diese erstellt bei Bedarf eine das Paket repräsentierende Zeichenkette, damit es beispielsweise durch `print` richtig dargestellt wird. Die Sendemethode erwartet

```
local cc1101 = msbiot.cc1101

-- Metatable for nice behaviour of packets under print()
local packet_mtable = {
    __tostring  = function(p)
        return "Source:      "    .. p[cc1101.PKG_SOURCE] .. "\n"
               .. "Destination: " .. p[cc1101.PKG_DEST]   .. "\n"
               .. "RSSI:        " .. p[cc1101.PKG_RSSI]   .. "\n"
               .. "Data:        " .. p[cc1101.PKG_DATA]
    end,
}

-- Function for sending some packets to and fro
function test_cc1101(channel, id, msg, ...)
    cc1101.init(channel, id, ...)

        [...]
        -- Receive a number of packets
        local received = cc1101.receive(n)

            [...]
            -- Print one
            setmetatable(packet, packet_mtable)
            print(packet)

            -- Prepare answer
            table.insert(to_send, {
                [cc1101.PKG_SOURCE] = id,
                [cc1101.PKG_DEST]   = packet[cc1101.PKG_SOURCE],
                [cc1101.PKG_DATA]   = msg .. total_sent,
            } )
            [...]

        -- Send away the prepared packets
        cc1101.send(to_send)
    end

    -- Switch the chip off
    cc1101.shut_down()
end
[...]
```

Quelltext 4.2: Auszüge aus dem Testskript für das CC1101-Modul.

Pakete in Form von Lua-Tabellen, die der Aufrufer selber zusammenbauen muss. Deren Schlüssel sind noch dazu aus dem Plattformmodul importierte numerische Konstanten, was in Lua sowohl syntaktisch als auch semantisch unidiomatisch ist. Um die Programmierschnittstelle sicherer und benutzerfreundlicher zu machen, sollte daher zumindest eine objektorientierte Abstraktion für Pakete eingeführt werden. Weitergehend könnte man einen Netzwerkstack auf dem CC1101 aufbauen und in Lua verfügbar machen.

4.3 CC3000

4.3.1 Funktionstest

Wie in Abschnitt 3.3.1 beschrieben, sollten Lua-Programme auf dem MSB-IOT per UDP mit entfernten Prozessen kommunizieren können, nachdem sie eine Verbindung mit einem WPA2-WLAN aufgebaut und per DHCP eine Netzwerkadresse bezogen haben. Um die Funktionstüchtigkeit der Implementierung zu prüfen, wurde ein WLAN-Access-Point mit DHCP-Server gestartet. Auf dem gleichen Rechner lief auch ein Programm, das auf eingehende UDP-Nachrichten wartet und dem Sender jeweils eine Antwort zurückgibt. Das in Listing 4.3 aufgeführte Lua-Skript nutzt das in Abschnitt 3.3.2 beschriebene Plattformmodul, um eine UDP-Nachricht zu schicken und zu empfangen. Man erkennt, dass die Funktionsaufrufe der POSIX-Socket-API prinzipiell sehr ähnlich, aber deutlich abstrakter sind. Außerdem könnte man den Inhalt der Variablen `wlan` zwischen den Aufrufen `wlan.socket()` und `wlan.close()` gegen ein beliebiges anderes Modul austauschen, das die API des *net*-Moduls von eLua implementiert.

4.3.2 Verbleibende Arbeit

Wie in Abschnitt 3.3.1 beschrieben, leitet das Plattformmodul nur einen vertikalen Schnitt durch die Funktionalität der Treiberbibliothek des CC3000 weiter. Damit es allgemein benutzbar wird, müssen zumindest noch die CC3000-Bibliotheksfunktionen für WLAN-Scan, DNS und TCP-Sockets an Lua weitergereicht werden. Sehr sinnvoll wären außerdem der Zugriff auf WLANs anderer Sicherheitsstufen, manuelle IP-Adresskonfiguration und eine Art `select`. Schließlich könnte man noch den Zugriff auf spezielle Funktionen des CC3000 ermöglichen, wie Smart Config und den eingebauten permanenten Parameterspeicher [33]. Die in Abschnitt 3.3.2 erwähnte Möglichkeit, das C-Modul mit einem Lua-Modul mit abstrakterer API zu umgeben, sollte dabei in Betracht gezogen werden.

```lua
local wlan  = msbiot.cc3000

--[[
Die with message msg if err is something else than wlan.ERROR_OK.
--]]
local function die_on_failure(err, msg)
    if err ~= wlan.ERR_OK then
        error(msg)
    end
end

local server_ip     = "192.168.3.3"
local server_port   = 20000

-- Initialise WLAN chip and connect to AP
wlan.init("bombusnet", "milchbart")

-- Set up connection to server
local sock = wlan.socket(wlan.SOCK_DGRAM)
die_on_failure(
    wlan.connect(sock, wlan.packip(server_ip), server_port),
    string.format("Cannot connect to %s:%u.", server_ip, server_port)
)

-- Greet the server
local n_sent, err = wlan.send(sock, "Hello, server!")
die_on_failure(err, "Cannot send.")
print(n_sent .. " bytes sent.")

-- Print what the server has to say
local data, err = wlan.recv(sock, 1024)
die_on_failure(err, "Error receiving.")
print( string.format('The server sent "%s"', data) )

-- Shut down the socket and the WLAN chip
die_on_failure( wlan.close(sock), "Cannot close socket." )
wlan.shut_down()
```

Quelltext 4.3: Lua-Skript für das MSB-IOT zur Kommunikation mit einem entfernten Prozess über UDP

5 Zusammenfassung

In dieser Arbeit wurde eLua auf das MSB-IOT portiert und damit die Möglichkeit geschaffen, Lua-Programme auf diesem Board auszuführen. Außerdem wurden zwei Plattformmodule entwickelt, durch die solche Lua-Programme komfortabel auf Peripheriegeräte des MSB-IOT zugreifen können. Das eine Peripheriegerät, der CC1101, erlaubt dem MSB-IOT, Daten per Funk im Sub-1 GHz-Band beispielsweise mit anderen Sensorboards auszutauschen. Über das andere, den CC3000, kann es sich mit WLANs verbinden.

Beim CC3000 muss man allerdings Einschränkungen hinnehmen: Seine mangelnde IPv6-Unterstützung ist rückschrittlich. Für Mesh-Netzwerke, ein Forschungsthema an der Freien Universität Berlin, ist er ungeeignet, weil er den WLAN-Ad-Hoc-Modus nicht unterstützt. – Dafür könnte man sich eine Anwendung im Bereich Sensornetze vorstellen: Viele Sensorknoten können mit Funk-Transceivern wie CC1100 oder CC1101 nur untereinander kommunizieren. Das MSB-IOT mit CC1101 und CC3000 kann als Basistation andere Sensorknoten mit dem Internet verbinden. – Konflikte in der Software verhindern bislang außerdem den gleichzeitigen Betrieb der USB-Schnittstelle und des CC3000-Moduls des MSB-IOT.

Nichtsdestotrotz ist eLua ein interessanter neuer Weg der Embedded-Programmierung. Wie man es benutzt, um Anwendungen für den MSB-IOT in Lua zu entwickeln, wurde in dieser Arbeit gezeigt. Außerdem wurde anhand von Beispielen die Möglichkeit vorgestellt, Lua zu erweitern, um das Vorhandensein von Treibersoftware auszunutzen. Und wie in der Einleitung erwähnt, können Entwickler diesen Mechanismus verwenden, um beliebige Programmkomponenten in C oder C++ zu schreiben und dann mit Lua-Skripten zusammenzusetzen.

Ausblick

Diese Arbeit zeigt, was Entwickler mit eLua machen können, und bildet damit den ersten Schritt auf dem Weg zu einer vollständigen Umgebung für eine Skriptsprache auf dem MSB-IOT. Der nächste Schritt ist, herauszufinden, ob eLua auch in realen Anwendungen besteht. Dazu sind Fragen zu klären wie: Lässt der Speicherbedarf des Lua-Interpreters daneben noch andere Programme zu? Kann Lua über den CC3000 eintreffende Daten überhaupt schnell genug verarbeiten? Verhindert der Stromverbrauch des Lua-Interpreters, den MSB-IOT oder andere Geräte ohne dauerhafte Energiezufuhr zu betreiben?

Die ersten zwei Fragen sind wegen der großzügigen Hardwareausstattung von Geräten wie dem MSB-IOT vermutlich unkritisch. Ressourcenintensive Komponenten können, wie bereits gezeigt, ohnehin in C/C++ implementiert werden. Der Stromverbrauch könnte deutlich problematischer sein, da die derzeitige eLua-Portierung für STM32F4-Prozessoren deren Energiesparfunktionen nicht nutzt. Hier müsste man sich also mit den Innereien von eLua und mit der Frage beschäftigen, ob deren Konstruktion mit dem Energiesparen überhaupt vereinbar ist.

Wenn man eLua dann immer noch für geeignet hält, gilt es, das Entwickeln so einfach wie von Skriptsprachen gewohnt zu machen: Lua-Skripte müssen per USB oder WLAN auf die SD-Karte geladen und von dort ausgeführt werden können. Und sie müssen die auf dem MSB-IOT vorhandenen Geräte über eine abstrakte Programmierschnittstelle ansteuern können. Schließlich liegt die Stärke von Skriptsprachen nicht darin, Software von Grund auf zu schaffen, sondern darin, existierende Komponenten zu verbinden [1].

In der vorliegenden Arbeit wurde nicht nur mit der Portierung von eLua auf das MSB-IOT die entscheidende Voraussetzung für eine solche Programmierumgebung geschaffen. Die vorgestellten Plattformmodule sind auch die ersten zwei Komponenten, die Programmierer zu neuen Anwendungen verbinden können. Sie und die Portierung sind Grundlage und Orientierungspunkt für alle weiteren Entwicklungen.

Glossar

ABI Application Binary Interface. Schnittstelle zwischen Softwarekomponenten auf der Ebene von Maschinenbefehlen. Legt beispielsweise Aufrufkonventionen fest.

CDC Communications Device Class. USB-Geräteklasse für Kommunikationsgeräte wie Telefone und Modems.

GDB GNU Debugger.

GPIO General-Purpose Input/Output. Frei programmierbarer Pin an einem elektronischen Bauelement.

ISR Interrupt Service Routine.

JVM Java Virtual Machine.

.NET MF .NET Micro Framework.

RSSI Receive Signal Strength Indicator. Maßzahl für die Stärke eines empfangenen Funksignals.

SPI Serial Peripheral Interface. Serielles Bussystem.

UART Universal Asynchronous Receiver Transmitter. Schaltung zur Realisierung einer seriellen Schnittstelle.

USART Universal Synchronous Asynchronous Receiver Transmitter. Wie UART, erlaubt aber zusätzlich zur asynchronen auch synchrone Datenübertragung.

Literaturverzeichnis

[1] J. K. Ousterhout, »Scripting: Higher-Level Programming for the 21st Century,« *Computer*, Bd. 31, Nr. 3, S. 23–30, März 1998. Adresse: http://www.stanford.edu/~ouster/cgi-bin/papers/scripting.pdf (besucht am 09. 01. 2014).

[2] D. Spinellis, »Java Makes Scripting Languages Irrelevant?« *Software*, Bd. 22, Nr. 3, S. 70–71, 2005.

[3] S. Bennett, »Effective Scripting in Embedded Devices,« WorkWare Systems, Apr. 2010. Adresse: http://workware.net.au/papers/embedded-scripting.pdf (besucht am 09. 01. 2014).

[4] Lua.org. (27. März 2013). Lua: about, Adresse: http://www.lua.org/about.html (besucht am 09. 01. 2014).

[5] (12. Nov. 2013). Lua for Kids, A guide for teaching kids how to code, Adresse: http://luaforkids.com/ (besucht am 09. 01. 2014).

[6] K. Chaykowski, »Twelve-Year-Old Programmers Help Fuel IPhone Game Frenzy: Tech,« *Bloomberg*, 29. Aug. 2012. Adresse: http://www.bloomberg.com/news/2012-08-29/twelve-year-old-programmers-help-fuel-iphone-game-frenzy-tech.html (besucht am 09. 01. 2014).

[7] B. Randel. (8. Juli 2013). Teaching kids Lua programming with ComputerCraft, Adresse: http://blog.nocturne.net.nz/education/2013/07/08/teaching-lua-with-computercraft/ (besucht am 09. 01. 2014).

[8] M. Kohl. (2009). Programming for kids? Adresse: http://happynerds.net/ (besucht am 09. 01. 2014).

[9] P. Merrell. Where Lua Is Used, Adresse: https://sites.google.com/site/marbux/home/where-lua-is-used (besucht am 09. 01. 2014).

[10] R. Ierusalimschy, L. H. de Figueiredo und W. C. Filho, »Lua – an extensible extension language,« *Software: Practice and Experience*, Bd. 26, Nr. 6, S. 635–652, 1996. Adresse: http://www.lua.org/spe.html (besucht am 09. 01. 2014).

[11] R. Ierusalimschy, L. H. de Figueiredo und W. Celes, *Lua 5.1 Reference Manual*, 13. Nov. 2012. Adresse: http://www.lua.org/manual/5.1/manual.html (besucht am 09. 01. 2014).

[12] eLua Project. (2011). Overview – What is Lua? Adresse: http://www.eluaproject.net/overview (besucht am 09. 01. 2014).

[13] A. Liers, Freie Universität Berlin. (2013). Mikroprozessor Praktikum – Hardwareplattform, Adresse: http://hwp.mi.fu-berlin.de/intern/STM32/02000000.php (besucht am 09. 01. 2014). Nur über FU-Netz abrufbar.

[14] VDC Research. (29. Juni 2011). 2011 Embedded Engineer Survey Results – Programming languages used to develop software, Adresse: http://blog.vdcresearch.com/embedded_sw/2011/06/2011-embedded-engineer-survey-results-programming-languages-used-to-develop-software.html (besucht am 09. 01. 2014).

[15] D. Saks. (5. Apr. 2012). Unexpected trends, Adresse: http://www.embedded.com/electronics-blogs/programming-pointers/4372180/Unexpected-trends (besucht am 09. 01. 2014).

[16] F. Aslam, »Challenges and Solutions in the Design of a Java Virtual Machine for Resource Constrained Microcontrollers,« Dissertation, Universität Freiburg, März 2011. Adresse: http://sourceforge.net/p/takatuka/code/2645/tree/docs/PhD-dissertation.pdf?format=raw (besucht am 09. 01. 2014).

[17] *Connected Limited Device Configuration Specification*, Version 1.1.1, Sun Microsystems, Inc., Nov. 2007. Adresse: http://download.oracle.com/otn-pub/jcp/cldc-1.1.1-mrel-eval-oth-JSpec/cldc-1_1_1-mrel-spec.zip (besucht am 09. 01. 2014).

[18] J. D. Carlson, M. Mittek und L. C. Pérez, »Exploring the Microsoft .NET Micro Framework for Prototyping Applied Wireless Sensor Networks,« University of Nebraska-Lincoln, Department of Electrical Engineering, Paper, 2013. Adresse: http://mc2.unl.edu/uploads/papers/eitc2013/exploring-netmf.pdf (besucht am 09. 01. 2014).

[19] A. Dunkels, »A Low-Overhead Script Language for Tiny Networked Embedded Systems,« Swedish Institute of Computer Science, Technical Report, Sep. 2006. Adresse: http://dunkels.com/adam/dunkels06lowoverhead.pdf (besucht am 09. 01. 2014).

[20] Tcl Developer Xchange. Features and Benefits, Adresse: http://www.tcl.tk/about/features.html (besucht am 09. 01. 2014).

[21] M. Baar, E. Köppe, A. Liers und J. Schiller, »Poster and Abstract: The ScatterWeb MSB-430 Platform for Wireless Sensor Networks,« in *SICS Contiki Hands-On Workshop*, (Kista, Schweden), Swedish Institute of Computer Science (SICS), März 2007.

[22] Arduino. Arduino – Products, Adresse: http://arduino.cc/en/Main/Products (besucht am 09. 01. 2014).

[23] Freie Universität Berlin. (3. Sep. 2012). Modular Sensor Board 430, Adresse: http://www.mi.fu-berlin.de/inf/groups/ag-tech/projects/Z_Finished_Projects/ScatterWeb/modules/mod_MSB-430.html (besucht am 09. 01. 2014).

[24] Freie Universität Berlin. (3. Sep. 2012). Modular Sensor Board 430-H, Adresse: http://www.mi.fu-berlin.de/inf/groups/ag-tech/projects/Z_Finished_Projects/ScatterWeb/modules/mod_MSB-430H.html (besucht am 09. 01. 2014).

[25] Freie Universität Berlin. (3. Sep. 2012). MSB A2, Adresse: http://www.mi.fu-berlin.de/inf/groups/ag-tech/projects/Z_Finished_Projects/ScatterWeb/modules/mod_MSB-A2.html (besucht am 09. 01. 2014).

[26] *STM32F415xx, STM32F417xx*, Rev. 4, Datasheet – production data, STMicroelectronics, 4. Juni 2013. Adresse: http://www.st.com/web/en/resource/technical/document/datasheet/DM00035129.pdf (besucht am 09. 01. 2014).

[27] *CC1101, Low-Power Sub-1 GHz RF Transceiver*, Rev. H, Texas Instruments, 9. Okt. 2012. Adresse: http://www.ti.com/lit/ds/symlink/cc1101.pdf (besucht am 09. 01. 2014).

[28] *CC1101, Low-Cost, Low-Power Sub-1 GHz RF Transceiver (Enhanced CC1100)*, Rev. D, Texas Instruments, 22. Mai 2008.

[29] E. Simensen, »Upgrade from CC1100 to CC1101,« Texas Instruments, Design Note DN009, 14. Juli 2009. Adresse: http://www.ti.com/lit/an/swra145a/swra145a.pdf (besucht am 09. 01. 2014).

[30] Texas Instruments. (2013). CC1100, Highly Integrated MultiCh RF Transceiver Designed for Low-Power Wireless Apps, Adresse: http://www.ti.com/product/cc1100 (besucht am 09. 01. 2014).

[31] Senslab. (27. März 2013). Getting Started, Adresse: http://wiki.senslab.info/documentation/getting_started/prerequisites (besucht am 09. 01. 2014).

[32] Freie Universität Berlin. (28. Okt. 2013). DES-Testbed – Hardware, Adresse: http://www.des-testbed.net/hardware (besucht am 09. 01. 2014).

[33] *TI SimpleLink CC3000 Module – Wi-Fi 802.11b/g Network Processor*, Texas Instruments, Nov. 2012. Adresse: http://www.ti.com/lit/gpn/cc3000 (besucht am 09. 01. 2014).

[34] M. Maurer und Aaron L, TI E2E Community. (13. Mai 2013). CC3000 with dual mode stack IPv4/IPv6 or IPv6 only? Adresse: http://e2e.ti.com/support/low_power_rf/f/851/t/266918.aspx (besucht am 09. 01. 2014).

[35] TI E2E Community. (27. Juni 2013). CC3000 and Ad-Hoc Mode, Adresse: http://e2e.ti.com/support/low_power_rf/f/851/t/159852.aspx (besucht am 09. 01. 2014).

[36] *CC3000 Host Programming Guide*, Texas Instruments Wiki, 23. Aug. 2013. Adresse: http://processors.wiki.ti.com/index.php?title=CC3000_Host_Programming_Guide&oldid=159326 (besucht am 09. 01. 2014).

[37] eLua Project. (2011). eLua – eLua Toolchains, Adresse: http://www.eluaproject.net/doc/master/en_toolchains.html (besucht am 09. 01. 2014).

[38] GCC ARM Embedded Maintainers. (26. Sep. 2013). GNU Tools for ARM Embedded Processors, Adresse: https://launchpad.net/gcc-arm-embedded (besucht am 09. 01. 2014).

[39] GCC ARM Embedded Maintainers, *Release notes for GNU Tools for ARM Embedded Processors 4.7 – Q3 2013*, 26. Sep. 2013. Adresse: https://launchpadlibrarian.net/151488439/release.txt (besucht am 09. 01. 2014).

[40] Mentor Graphics. (8. März 2013). How do I get the compiler to generate VFP instructions? Adresse: https://sourcery.mentor.com/sgpp/lite/arm/portal/kbentry27 (besucht am 09. 01. 2014).

[41] *ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32*, Rev. 3, Data brief, STMicroelectronics, 14. Dez. 2012. Adresse: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/data_brief/DM00027105.pdf (besucht am 09. 01. 2014).

[42] *STM32 ST-LINK Utility software description*, Rev. 13, User manual, STMicroelectronics, 4. Nov. 2013. Adresse: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/CD00262073.pdf (besucht am 09. 01. 2014).

[43] texane. (31. Okt. 2013). stlink, stm32 discovery line linux programmer, Adresse: https://github.com/texane/stlink (besucht am 09. 01. 2014).

[44] *STM32F105xx, STM32F107xx, STM32F2xx and STM32F4xx USB On-The-Go host and device library*, Rev. 3, User manual, STMicroelectronics, 8. März 2012. Adresse: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/CD00289278.pdf (besucht am 09. 01. 2014).

[45] Lua.org. (5. Okt. 2013). Lua: getting started, Adresse: http://www.lua.org/start.html (besucht am 09. 01. 2014).

[46] Lua.org. (25. Juli 2013). Lua: FAQ, Adresse: http://www.lua.org/faq.html (besucht am 09. 01. 2014).

[47] lua-user wiki. (25. Mai 2013). Environments Tutorial, Adresse: http://lua-users.org/wiki/EnvironmentsTutorial (besucht am 09. 01. 2014).

[48] lua-user wiki. (24. Sep. 2013). Object Orientation Tutorial, Adresse: http://lua-users.org/wiki/ObjectOrientationTutorial (besucht am 09. 01. 2014).

[49] lua-user wiki. (27. Aug. 2013). Tutorial Examples, Adresse: http://lua-users.org/wiki/TutorialExamples (besucht am 09. 01. 2014).

[50] T. Müller, *Lua*, Poster, 9. Feb. 2007. Adresse: http://www.schulze-mueller.de/download/lua-poster-090207.pdf (besucht am 09. 01. 2014).

[51] eLua Project. (2011). EGC (Emergency GC) in eLua, Adresse: http://www.eluaproject.net/doc/master/en_elua_egc.html (besucht am 09. 01. 2014).

[52] eLua Project. (2011). LTR (Lua Tiny RAM) in eLua, Adresse: http://www.eluaproject.net/doc/master/en_arch_ltr.html (besucht am 09. 01. 2014).

[53] eLua Project. (2011). eLua – Generic modules, Adresse: http://www.eluaproject.net/doc/master/en_refman_gen.html (besucht am 09. 01. 2014).

[54] eLua Project. (2011). eLua – Generic Info, Adresse: http://www.eluaproject.net/doc/master/en_using.html (besucht am 09. 01. 2014).

[55] eLua Project. (2011). eLua internals – Overview, Adresse: http://www.eluaproject.net/doc/master/en_arch_overview.html (besucht am 09. 01. 2014).

[56] eLua Project. (2011). eLua – Platform interface, Adresse: http://www.eluaproject.net/doc/master/en_arch_platform.html (besucht am 09. 01. 2014).

[57] eLua Project. (2011). eLua – Building eLua, Adresse: http://www.eluaproject.net/doc/master/en_building.html (besucht am 09. 01. 2014).

[58] eLua Project. (2011). eLua – Status, Adresse: http://www.eluaproject.net/doc/master/en_status.html (besucht am 09. 01. 2014).

[59] eLua. (2011). eLua reference manual – LM3S disp module, Adresse: http://www.eluaproject.net/doc/v0.9/en_refman_ps_lm3s_disp.html (besucht am 09. 01. 2014).

[60] Simplemachines. (2011). Mizar32 Software, Adresse: http://simplemachines.it/index.php/modules-menu/mizar32-software/elua-faq (besucht am 09. 01. 2014).

[61] eLua Wiki. (8. Juli 2013). Projects, Adresse: http://wiki.eluaproject.net/Projects (besucht am 09. 01. 2014).

[62] B. Marinescu. (30. Jan. 2012). eLuaBrain, A technical blog built around an eLua based stand alone computer, Adresse: http://eluabrain.blogspot.de/ (besucht am 09. 01. 2014).

[63] eLua Project. (2011). eLua – Authors, Adresse: http://www.eluaproject.net/overview/authors (besucht am 09. 01. 2014).

[64] R. Ierusalimschy, *Programming in Lua*, 2. Aufl. Lua.org, März 2006.

[65] eLua Project. (2011). eLua – Adding a new port, Adresse: http://www.eluaproject.net/doc/master/en_arch_newport.html (besucht am 09. 01. 2014).

[66] eLua Wiki. (24. Jan. 2012). eLua on STM32F4DISCOVERY, Adresse: http://wiki.eluaproject.net/STM32F4DISCOVERY (besucht am 09. 01. 2014).

[67] STMicroelectronics. (2013). STM32F4DISCOVERY, Discovery kit for STM32F407/417 lines – with STM32F407VG MCU, Adresse: http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419 (besucht am 09. 01. 2014).

[68] STMicroelectronics. (2013). STM32F4 Series, Adresse: http://www.st.com/web/en/catalog/mmc/FM141/SC1169/SS1577 (besucht am 09. 01. 2014).

[69] STMicroelectronics. (2013). STM32F407VG, High-performance and DSP with FPU, ARM Cortex-M4 MCU with 1 Mbyte Flash, 168 MHz CPU, Art Accelerator, Ethernet, Adresse: http://www.st.com/web/catalog/mmc/FM141/SC1169/SS1577/LN11/PF252140 (besucht am 09. 01. 2014).

[70] eLua Project. (2011). eLua – Configuring the image, Adresse: http://www.eluaproject.net/doc/master/en_configurator.html (besucht am 09. 01. 2014).

[71] eLua. (2011). eLua reference manual – SPI module, Adresse: http://www.eluaproject.net/doc/v0.9/en_refman_gen_spi.html (besucht am 09. 01. 2014).

[72] eLua Project. (2011). eLua interrupt handlers, Adresse: http://www.eluaproject.net/doc/master/en_inthandlers.html (besucht am 09. 01. 2014).

[73] *SimpleLink API Reference Manual*, Version 1.11.1, Texas Instruments, 1. Okt. 2013. Adresse: http://software-dl.ti.com/ecs/simplelink/cc3000/public/doxygen_API/v1.11.1/html/index.html (besucht am 09. 01. 2014).

[74] eLua. (2011). eLua reference manual – net module, Adresse: http://www.eluaproject.net/doc/v0.9/en_refman_gen_net.html (besucht am 09. 01. 2014).

[75] D. Nehab. (3. Okt. 2007). LuaSocket: Network support for the Lua language, Adresse: http://w3.impa.br/~diego/software/luasocket/home.html#download (besucht am 09. 01. 2014).

[76] A. Liers, Freie Universität Berlin. (2013). Mikroprozessor Praktikum – CC3000-WLAN Modul, Adresse: http://hwp.mi.fu-berlin.de/intern/STM32/02020000.php (besucht am 09. 01. 2014). Nur über FU-Netz abrufbar.

[77] M. Rodda und J. Fahllund, Hrsg., *Universal Serial Bus Micro-USB Cables and Connectors Specification*, Revision 1.01, USB Implementers Forum, Inc., 4. Apr. 2007. Adresse: http://www.usb.org/developers/docs/usb_20_070113.zip (besucht am 09. 01. 2014).

[78] *STM32F3xxx and STM32F4xxx Cortex-M4 programming manual*, Rev. 3, STMicroelectronics, 4. Sep. 2012. Adresse: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/programming_manual/DM00046982.pdf (besucht am 09. 01. 2014).

[79] *IEEE Standard for Local and metropolitan area networks, Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE Std 802.15.4-2011. IEEE Computer Society, 5. Sep. 2011. Adresse: `http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf` (besucht am 09. 01. 2014).

Inhalt des Datenträgers

Die beigefügte CD enthält

- heruntergeladene Versionen der Quellen, für die im Literaturverzeichnis eine URL angegeben ist,
- den Quelltext von eLua mitsamt aller im Rahmen dieser Arbeit gemachten Änderungen und Ergänzungen in Form eines Git-Repositories,
- eine frühe Version des Abschnitts 3.3.4 mit mehr Hinweisen zur Fehlersuche beim CC3000,
- diese Arbeit als PDF-Datei.