Evaluierung und Optimierung der Energy Harvesting Testplattform für RIOT

Michel Rottleuthner

HAW-Hamburg, Fakultät Technik und Informatik, Department Informatik,
Berliner Tor 7, 20099 Hamburg, Germany
michel.rottleuthner@haw-hamburg.de
http://informatik.haw-hamburg.de

Zusammenfassung. Dieser Projektbericht zeigt die Optimierung und Evaluierung der Energy Harvesting Testplattform für RIOT. Dabei wird der Umstieg auf ein neues Evaluierungsboard, dessen Modifikation zur einfachen Verwendung in Feldtests, die Entwicklung der dazu nötigen Softwarekomponenten und die iterative Weiterentwicklung der Prototypen für Ladeelektronik und Messmodul präsentiert. Weiterhin werden Messungen zu Leistungsfähigkeit und Energiebedarf der Logging-Komponente vorgestellt. Deren Flexibilität wird mithilfe der Integration von generischen Abstraktionsebenen erhöht.

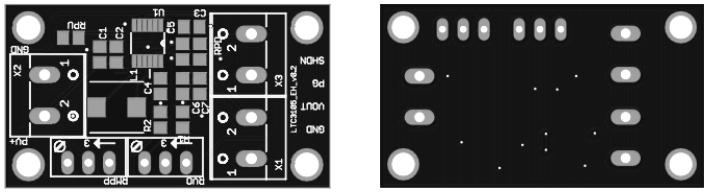
1 Einleitung

Energie-autarke Sensorknoten bieten gegenüber konventionell batteriebetriebenen Systemen die Vorteile eines wartungsarmen Dauereinsatzes und erhöhter Verfügbarkeit von Energieressourcen. Auch auf derartigen Systemen können Betriebssysteme wie RIOT [1] eingesetzt werden, was erheblich zur Vereinfachung der Entwicklung beitragen kann. Eine vorangegangene Arbeit zeigt den Aufbau einer Testplattform für Energy Harvesting (EH) mit RIOT [2]. Die Testplattform kann bereits für erste Versuche eingesetzt werden. Diese zeigen, dass das verwendete Evaluierungsboard für realistische Feldtests stark angepasst werden muss, um einen ausreichend geringen Energieverbrauch zu erreichen.

Abschnitt 2 und 3 zeigen zunächst die Verbesserungen an den Platinen für Ladeelektronik und Messmodul. Anschließend wird in Abschnitt 4, alternativ zu den zuvor genannten Modifikationen, die Auswahl und Integration eines neuen Evaluierungsboards vorgestellt. Dazu werden die benötigten Softwarekomponenten entwickelt. Als letzter Baustein wird in Abschnitt 5 auf die Logging-Komponente eingegangen und ihre Leistungsfähigkeit sowie der Energieverbrauch genauer untersucht. Das Ziel ist, Daten für die Parametrisierung der geplanten Feldtests zu gewinnen. Mit der Integration von generischen Abstraktionsebenen in die Loggig-Komponente wird gezeigt, wie diese flexibler gestaltet werden kann. Abschnitt 6 fasst die Arbeit abschließend zusammen und gibt Anregungen zu weiterführenden Arbeiten.

2 Ladeelektronik

Der Prototyp der Ladeelektronik wurde bereits mehrfach zu Demonstrationszwecken eingesetzt und hat dabei keine grundlegenden Probleme in der Funktionsweise aufgezeigt. Bei sehr schwacher Ausleuchtung der Photovoltaik-Zelle (PV-Zelle) zeigt sich jedoch, dass die Schaltung teilweise nicht in der Lage ist, den shutdown-Modus zu verlassen. Hierfür sollten laut Datenblatt des ICs nur knapp 250 mV am Eingang benötigt werden. Ein zusätzlicher Kondensator an der Eingangsseite schafft hier Abhilfe, allerdings muss dieser aufgeladen sein, bevor die Ladeelektronik aktiviert wird, um den erhöhten Strombedarf beim Aktiveren der Schaltung bereitzustellen. Wird die Ladeelektronik über den SHDN-Pin aktiviert, bevor der Kondensator ausreichend aufgeladen ist, dann kann es zu einem oszillierenden Zustand zwischen enable- und shutdown-Modus kommen. Da dieses Verhalten nur bei sehr schwacher Ausleuchtung auftritt, wird zunächst davon abgesehen einen zusätzlichen Schwellwertschalter oder ähnliches zu integrieren, welcher die Ladeelektronik erst aktiviert, sobald der Pufferkondensator ausreichend geladen ist. In einer neuen Revision der Schaltung (siehe Abbildung 1) werden Pads für optionale Pull-Widerstände (RPD und RPU) für die SHDN-Leitung integriert. Damit kann festgelegt werden, ob die Schaltung standardmäßig aktiv oder inaktiv sein soll.



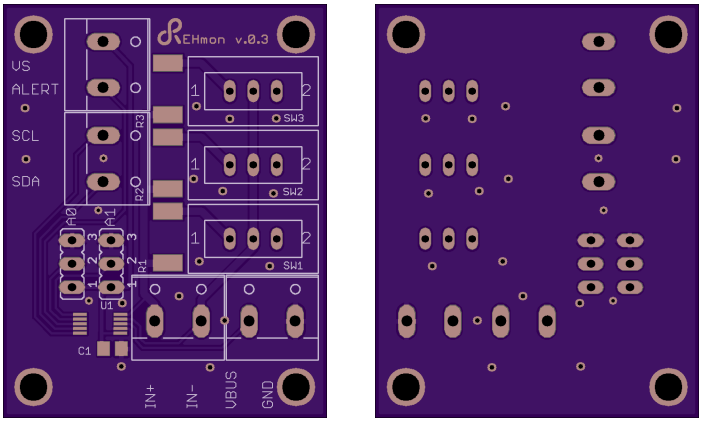
(a) Oberseite (b) Unterseite

Abb. 1: Platinenlayout des Lademoduls

3 Messmodul

Das Messmodul wird ebenfalls mit einer neuen Revision der Schaltung ausgestattet. Diese ist in Abbildung 2 zu sehen und besitzt im Vergleich zur Prototypen-Variante hochwertigere Schalter mit einem niedrigeren Übergangswiderstand aufgrund der verwendeten Silberkontaktierungen. Laut Datenblatt kann dadurch der Innenwiderstand auf ein Maximum von 10 mΩ begrenzt werden. Evaluierungsmessungen haben gezeigt, dass die Änderung des Innenwiderstands bei geschlossenem Zustand zwischen verschiedenen Schaltvorgängen nun wesentlich geringer ausfällt. Bei einem Wechsel der Schaltposition blieb die Abweichung bei allen Versuchen unterhalb von 1 mΩ. Diese Schwankung wirkt sich direkt auf den

Messfehler aus, wenn das Modul nach dem Umschalten des Messbereichs nicht erneut kalibriert wird. Selbst im Messbereich mit dem kleinsten Messwiderstand von 160 mΩ bleibt der dadurch verursachte Fehler nun unter 0,7 %. Im Messbereich für die kleinsten Ströme (750 mΩ Messwiderstand) entspricht das einem Fehler von knapp 0,13 %, was unter Betrachtung anderer Einflussfaktoren wie Temperatur und Leitungswiderständen vernachlässigbar wenig ist.



(a) Oberseite (b) Unterseite

Abb. 2: Platinenlayout des Messmoduls

4 Umstieg auf ein neues Evaluierungsboard

Das bisher zum Testen verwendete samr21-xpro[1] Board lässt sich ohne erhebliche Modifikationen an der Hardware nicht für Feldtests verwenden. Unter anderem verhindern ein fest verlöteter Embedded Debugger (EDBG) und dauerhaft aktivierte LEDs die reale Ausnutzung der Energiesparmechanismen der MCU. Um den EDBG-Chip zu deaktivieren, müssen 18 verschiedene Lötbrücken entfernt werden. Um den EDBG anschließend wieder zu reaktivieren, müssen alle Lötbrücken wieder geschlossen werden. Im Rahmen von Tests und Evaluierungsmessungen häufig neue Firmwareversionen auf das Board zu laden, erweist sich damit als nicht praktikabel. Da alle entwickelten Komponenten der Testplattform plattformunabhängig ausgelegt sind, wird der Wechsel auf ein anderes Evaluierungsboard angestrebt.

Als besonders geeignet zeigt sich das nucleo-l476 der Firma STMicroelectronics. Dieses ist werksseitig dafür ausgelegt den Flash/Debug-Chip vollständig

[1] Die Bezeichnungen der Boards beziehen sich auf die eindeutigen Namen in RIOT, siehe: `https://github.com/RIOT-OS/RIOT/tree/master/boards`

vom Evaluierungsboard abtrennen zu können. In Abbildung 3 ist das Board mit eingezeichneter Trennlinie zu sehen. Über den CN4/SWD-Header kann das abgetrennte Erweiterungsboard weiterhin über Kabel angebunden werden.

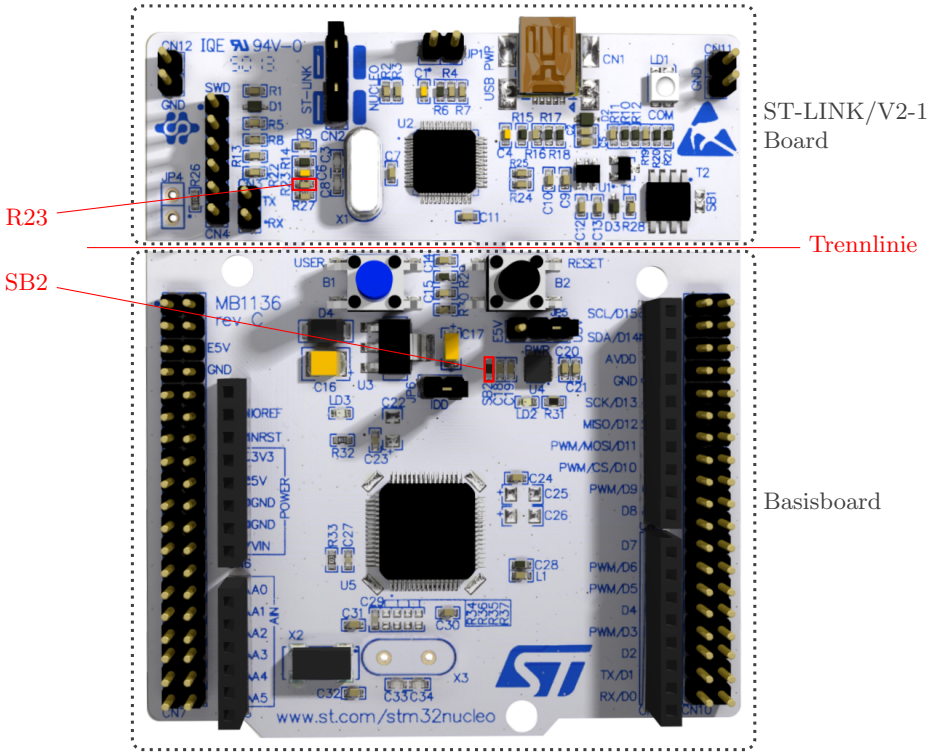


Abb. 3: STM32L476 Nucleo-64 Board Oberseite

Um die Handhabung hier noch weiter zu vereinfachen, werden drei weitere Jumper-Brücken angebracht, anstatt das Board abzutrennen. Mit den Jumper-Brücken wird die Funktion der Lötbrücken SB12 (siehe Abbildung 4) und SB2 (siehe Abbildung 3) einfach zugänglich und schnell konfigurierbar gemacht. SB12 verbindet den NRST-Pin der MCU mit dem ST-LINK-Board. SB2 verbindet die MCU mit dem 3,3 V Spannungsregler U4. Beide müssen getrennt werden, damit das Board über einen externen Spannungsregler betrieben werden kann. Laut dem Benutzerhandbuch[2] sind außer dem Auftrennen dieser beiden Lötbrücken keine weiteren Modifikationen nötig. Messungen haben gezeigt, dass bei Verwendung einer externen Spannungsquelle durch den Widerstand R23 (47,2 kΩ, siehe Abbildung 3) ein Leckstrom zum ST-LINK-Chip auftritt. Die dritte Jumper-Brücke dient dazu, diese Verbindung zu trennen. Eine Kontrollmessung zeigt,

[2] `http://www.st.com/resource/en/user_manual/dm00105823.pdf`

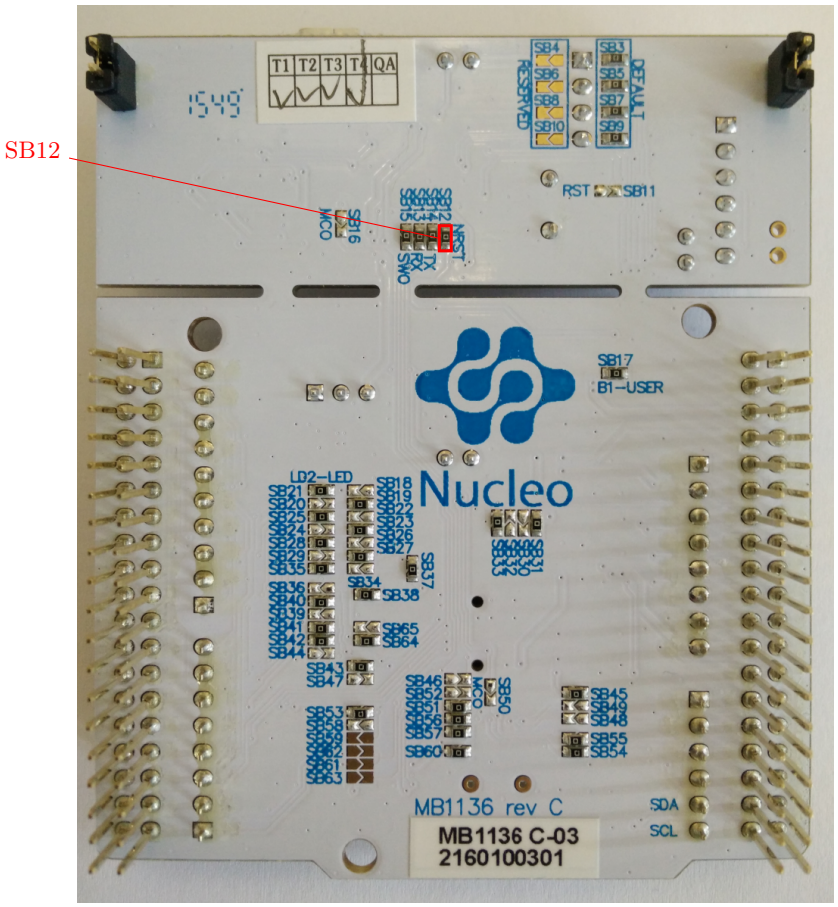


Abb. 4: STM32L476 Nucleo-64 Board Unterseite

dass mit diesen Anpassungen der Strombedarf mit aktiver Real-Time Clock (RTC) für das gesamte Board unterhalb von 1 µA liegt. Damit ist diese Plattform auch zur Verwendung in weiteren Feldtests geeignet. Werden die Jumper-Brücken wieder verbunden, kann das Board weiterhin wie im Auslieferungszustand neu bespielt werden.

Das Board besitzt kein integriertes Funkmodul, wodurch auf die Verwendung eines externen Moduls ausgewichen werden muss. Hieraus ergibt sich weiterhin die Möglichkeit, Messungen mit verschiedenen Funkmodulen durchzuführen, ohne auf eine andere MCU ausweichen zu müssen. Die externe Anbindung erlaubt außerdem eine bessere Kontrolle über die Energieversorgung des Funkmoduls. Wird das Modul nicht verwendet, kann die Versorgungsspannung des Moduls vollständig abgeschaltet werden.

4.1 Energiesparmodi für STM32L4 basierte Plattformen

Für STM32L4-basierte Plattformen existiert bisher keine Implementierung der Energiesparmodi. Im Rahmen dieser Arbeit wird diese zur existierenden Implementierung für andere STM32-Varianten in die Datei pm.c[3] integriert. Aufgrund andauernder Umstrukturierungen der Low-Power Architektur wird vorerst kein Pull-Request hierfür erstellt und zu einem späteren Zeitpunkt nachgepflegt. Der betreffende Code ist im Entwicklungszweig pm_stm32l4[4] abgelegt.

4.2 RTC Implementierung

Für das nucleo-l476 Board existiert bisher keine Implementierung für die RTC-Peripherie in RIOT. Die RTC ist für weitere Versuche nötig, um zeitgesteuert einen Interrupt zum Verlassen der Energiesparmodi auszulösen. Im Rahmen dieser Arbeit wird eine RTC-Implementierung für STM32L4-basierte Mikrocontroller erstellt. Die Implementierung ist bereits in den aktuellen master-Branch von RIOT eingepflegt[5].

Tests der Implementierung zeigen, dass im generischen Teil (stm32_common) der RTC-Funktionen mehrere Fehler existieren. Diese zeigen sich darin, dass ein Aufruf der Funktion `rtc_get_time(struct tm *time)` fehlerhafte Auswirkungen auf nachfolgende Aufrufe bewirkt und dadurch inkonsistente Ergebnisse liefert. Die Ursache hierfür ist eine fehlende Synchronisation, die aufgrund der verschiedenen Frequenzen zwischen APB- und RTC-Takt notwendig ist. Der generische Test für den RTC-Treiber maskiert diesen Fehler jedoch durch einen Aufruf von `xtimer_usleep()`. Dadurch ist die fehlende Synchronisation auch bei einer ordnungsgemäßen Ausführung des Tests zunächst nicht sichtbar. Weiterhin wird im Treiber die Spezifikation zur Lesereihenfolge der RTC_TR und RTC_DR-Register verletzt. Dadurch verursacht eine MCU-interne Funktion zum atomaren Auslesen beider Register, dass der Inhalt des RTC_DRRegisters zwischen zwei Aufrufen gesperrt bleibt und beim zweiten Aufruf zusammen mit einem neuen Wert des RTC_TR-Registers kombiniert wird. Zur Behebung der Fehler in Implementierung und Test, wird ein Pull-Request erstellt[6].

4.3 Taktkonfiguration

Der Initialisierungsablauf für die Taktkonfiguration befindet sich in der Methode `cpu_clock_init()` in der Datei `cpu/stm32l4/cpu.c` und ist durch das Schlüsselwort static ausschließlich zur Verwendung innerhalb dieser Datei vorgesehen. Nachdem die MCU den Low-Power-Mode z. B. aufgrund eines Interrupts verlässt, muss die Taktkonfiguration gegebenenfalls erneut vorgenommen werden. Dazu muss die Funktionalität der zuvor genannten Methode für andere Komponenten verfügbar gemacht werden. Der monolithische Ansatz der Methode

[3] `https://github.com/RIOT-OS/RIOT/blob/master/cpu/stm32_common/periph/pm.c`
[4] `https://github.com/MichelRottleuthner/RIOT/tree/pm_stm32l4`
[5] `https://github.com/RIOT-OS/RIOT/pull/7420`
[6] `https://github.com/RIOT-OS/RIOT/pull/7454`

`cpu_clock_init()` initialisiert Clock-Domains unabhängig von deren Verwendung. Das betrifft auch Clock-Domains, die von der RTC benötigt werden. Da dies zu Problemen wie Jitter, einem Reset oder dem Anhalten der RTC führen kann, wird die Implementierung des neuen Interfaces zur Konfiguration von Takteinstellungen, namens stmclk bevorzugt. Dieses erlaubt eine differenziertere Einstellung der Clock-Domains. Das Interface dient in RIOT dazu, die Taktkonfiguration der vielzähligen unterstützten STM32-Mikrocontroller zu vereinheitlichen. Die Konfiguration folgt auf diesen Plattformen meist einem ähnlichen Schema. Über das Interface können verschiedene Clock-Domains initialisiert, ein- und ausgeschaltet werden. Dazu stehen folgende Methoden bereit:

```
1 void stmclk_init_sysclk(void);
2 void stmclk_enable_hsi(void);
3 void stmclk_disable_hsi(void);
4 void stmclk_enable_lfclk(void);
5 void stmclk_disable_lfclk(void);
6 void stmclk_bdp_unlock(void);
7 void stmclk_bdp_lock(void);
```

Die Methode `stmclk_init_sysclk()` konfiguriert während des Bootvorgangs die zum Betrieb der MCU notwendigen Taktgeber. Entsprechend können die Methoden `*hsi()` und `*lfclk()` aufgerufen werden, um die **H**igh **S**peed **I**nternal Clock respektive **L**ow **F**requency Clock unabhängig voneinander ein und auszuschalten. Mittels `*bdp_unblock()` und `*bdp_unblock()` kann die **B**ackup **D**omain **P**rotection gesteuert werden, was z. B. zur Konfiguration der geschützten RTC-Register notwendig ist.

Für STM32L4-basierte Mikrocontroller ist das stmclk-Interface bisher nicht implementiert. Die Methoden werden anhand der Datenblattvorgaben umgesetzt und für den resultierenden Programmcode wird ein weiterer Pull-Request bereitgestellt[7].

4.4 Duty-Cycling Anwendung

Durch die Kombination der zuvor eingeführten Komponenten, lässt sich mit dem nucleo-l476 eine einfache Anwendung mit Duty-Cycling realisieren. Abbildung 5 zeigt die Stromaufnahme der MCU in einer Anwendung, die alle zwei Sekunden ein Paket an einen anderen Knoten überträgt und dazwischen in den tiefsten Energiesparmodus wechselt. Diesen Schlafmodus verlässt die MCU durch einen RTC-gesteuerten Interrupt. Beim Sendevorgang beträgt die Stromaufnahme ca. 13 mA wohingegen im Schlafmodus nur 0,9 µA benötigt werden. Diese Werte beziehen sich ausschließlich auf die MCU und beinhalten nicht die Stromaufnahme des Funkmoduls oder Verluste die im DC-DC-Wandler auftreten.

[7] `https://github.com/RIOT-OS/RIOT/pull/7469`

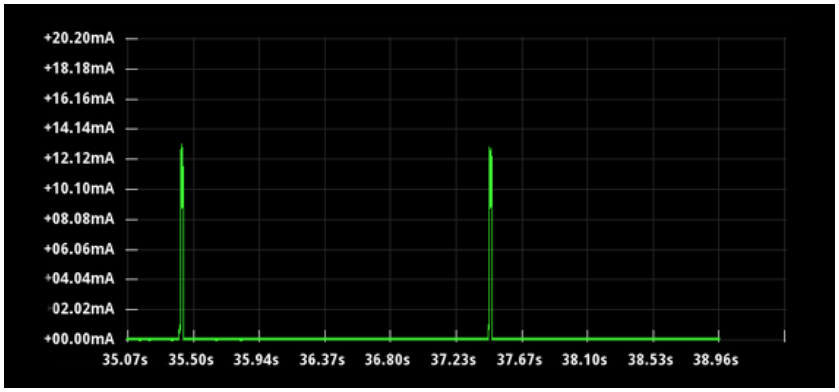


Abb. 5: Duty-Cycling mit RTC als Weakup-Trigger

5 Logging-Komponente

Die Logging-Komponente dient dazu, die Messdaten wie z. B. den Energieverbrauch des EH-Systems auf einer SD-Karte persistent zu speichern. Dazu wird der Treiber sdcard_spi verwendet. Um Limitierungen der Speicherlösung identifizieren und deren Leistungsfähigkeit beziffern zu können, werden im folgenden Abschnitt Messungen beschrieben. Außerdem wird die weitere Entwicklung und Integration von Abstraktionsschichten in die Teilmodule der Logging-Komponente erläutert. Diese generischen Abstraktionsschichten sollen die Modularität erhöhen und damit Flexibilität durch das einfachere Austauschen von Einzelkomponenten ermöglichen.

5.1 Evaluierung des Treibers sdcard_spi

Zur Evaluierung des Treibers sdcard_spi werden verschiedene Messreihen durchgeführt. Zum einen wird untersucht, wie hoch der erreichbare Durchsatz beim Lesen und Schreiben ist. Eine weitere Metrik von Interesse, ist die Dauer des Initialisierungsvorgangs. Anhand dieser Daten soll eine Abwägung bezüglich der Dimensionierung von Puffergrößen und dem auftretenden Overhead durch die Initialisierung nach einem vollständigen Abschalten der Karte ermöglicht werden. Der zweite Hauptaspekt der Messungen soll den Energieverbrauch in verschiedenen Zuständen der Karte beleuchten. Unterschieden wird hierbei zwischen Lesen, Schreiben, Initialisieren und Ruhezustand. Da verschiedene Karten sich erheblich in Ihren Eigenschaften unterscheiden können, werden fünf verschiedene Karten untersucht und miteinander verglichen. Die Karten unterscheiden sich unter anderem anhand des Herstellers, der Speicherkapazität und des SD-Karten Standards, um ein möglichst diversifiziertes Bild abzugeben.

Performance Zum Messen der Performance wird eine Testapplikation implementiert. Für alle Messungen wird das nucleo-l476 Board verwendet. Dieses läuft

mit einer Taktfrequenz von 80 MHz. Zur Initialisierung der Karte wird die Standardkonfiguration des sdcard_spi-Treibers verwendet. Diese verwendet zum Aktivieren des SPI-Modus eine Taktfrequenz von 100 kHz. Der darauf folgende Teil der Initialisierungsroutine erfolgt mit 400 kHz. Nach der Initialisierung wird die Taktfrequenz des SPI-Bus auf 10 MHz angehoben. Der Durchsatz wird bei der Übertragung einer Datenmenge von 10 MiB gemessen. Hierbei wird in beiden Richtungen ein Puffer von 10 Blockgrößen (5 KiB) verwendet. Diese Größe wird aufgrund der Annahme gewählt, dass auf den Zielplattformen in der Regel ausreichend RAM dafür zur Verfügung steht. Beim Schreiben auf die Karte wird kontinuierlich aus einem vorinitialisierten statischen Speicherbereich gelesen. Dies wird so lange wiederholt, bis die Grenze von 10 MiB erreicht ist. Der vorinitialisierte statische Speicher enthält dabei über die gesamte Größe von 5 KiB alle Werte von 0 bis 255 in gleicher Häufigkeit und zufälliger Reihenfolge. Nach jedem Übertragungsvorgang wird über den Rückgabewert geprüft, ob die Übertragung erfolgreich war, um verfälschte Ergebnisse durch übersprungene Blöcke auszuschließen. Außerdem ist die CRC-Überprüfung in beiden Richtungen aktiv.

Um die Dauer der Initialisierung zu bestimmen, muss die Karte ausgehend vom ausgeschalteten Zustand neu initialisiert werden. Um sicher zu gehen, dass der gemessene Wert keinen Ausreißer darstellt, wird die Messung über 1000 Initialisierungsvorgänge durchgeführt und daraus der Mittelwert gebildet. Damit die Karte automatisiert ausgeschaltet werden kann, wird ein MOSFET zwischen Spannungsversorgung und SD-Karte geschaltet. Das Steuersignal wird an einen Mikrocontroller-Pin angebunden, der in der Konfiguration des sdcard_spi-Treibers als Power-Pin hinterlegt wird. Nach fertiggestellter Initialisierung kann die Karte über diesen Pin abgeschaltet werden, der Treiber kümmert sich darum, die Spannungsversorgung der Karte vor der Software-seitigen Initialisierung zu aktivieren. Die Zeit wird direkt auf dem Mikrocontroller über die Differenz zwischen zwei Aufrufe der Funktion `xtimer_now()` gemessen. Damit die ausreichende Genauigkeit dieser Zeitmessung verifiziert werden kann, schaltet die Testapplikation bei Beginn und Beendigung des zu messenden Vorgangs einen GPIO-Pin des Mikrocontrollers. Dieses Signal kann mit einem externen Messgerät ausgewertet und anschließend mit der Zeitangabe der Testapplikation verglichen werden.

Tabelle 1 listet die verwendeten Kartenmodelle und die Ergebnisse der Tests für den Lese- und Schreibdurchsatz sowie für die Dauer eines Initialisierungsvorgangs. Die Ergebnisse erlauben eine erste Einschätzung bezüglich des erreichbaren Durchsatzes. Sehr deutlich zeigt sich, dass der Durchsatz auf allen Karten sowohl beim Schreiben als auch beim Lesen in der gleichen Größenordnung limitiert wird. Beim Lesen liegt die Rate im Bereich zwischen 283 und 296 KiB/s wohingegen das Schreiben etwas langsamer mit 231 bis 264 KiB/s ausgeführt wird. Die Abweichung über alle Karten liegt bei der Leserate somit unterhalb von 5 % und beim Schreiben unterhalb von 15 %. Wesentlich signifikantere Unterschiede zeigen sich bei der Initialisierungsdauer. Im langsamsten Fall benötigt die Karte 381,3 ms, während die schnellste Karte bereits nach 7,5 ms einsatzbereit ist.

Nr.	Karte	Lesen	Schreiben	Initialisieren
1	SanDisk Ultra (SL32G) 32GB	293 KiB/s	263 KiB/s	14,9 ms
2	SanDisk (SU02G) 2GB (A)	288 KiB/s	231 KiB/s	23,7 ms
3	SanDisk (SU02G) 2GB (B)	290 KiB/s	261 KiB/s	10,7 ms
4	Intenso (UHS-I) 16GB	283 KiB/s	259 KiB/s	7,5 ms
5	Kingston (SD-C01G) 1GB	296 KiB/s	264 KiB/s	381,3 ms

Tabelle 1: Schreib- und Leserate (Mittelwert bei Bulk-Übertragung von 10 MiB) und Initialisierungsdauer (Mittelwert über 1000 Initialisierungsvorgänge)

Als eine Ursache für die gleichmäßig limitierenden Geschwindigkeiten wird das Zusammenspiel des Treibers sdcard_spi mit den Low-Level Treibern für SPI vermutet. Der sdcard_spi-Treiber ruft zur Datenübertragung stets die Funktion `spi_transfer_byte` für jedes Byte auf, statt zur Übertragung von größeren zusammenhängenden Blöcken auf die Methode `spi_transfer_bytes` zurückzugreifen. Die Ursache hierfür ist, dass aus Kompatibilitätsgründen als sogenanntes Dummy-Byte bei der Kommunikation mit der Karte 0xFF übertragen werden muss. Der Low-Level SPI-Treiber überträgt bei einer reinen Leseoperation (out-Parameter == NULL) jedoch 0x00 als Dummy-Byte, womit einige Karten nicht funktionieren. Um dieses Problem zu umgehen, werden drei Möglichkeiten betrachtet. Es kann jedes Byte einzeln unter Angabe des Dummy-Bytes übertragen werden, was der momentanen Implementierung entspricht. Diese Variante bietet den Vorteil einer geringen Speicherauslastung, die auf Kosten einer reduzierten Performance erkauft wird. Alternativ kann ein statischer Speicherbereich im ROM angelegt werden, der mit den Dummy-Bytes vorinitialisiert wird. Dieser kann dann an die Bulk-Funktion `spi_transfer_bytes` übergeben werden. Damit wird ein höherer Durchsatz erreicht, allerdings steigt auch der Speicherbedarf.

Als weitere Möglichkeit könnte die Schnittstelle des Low-Level-SPI Treibers um einen Parameter für einen expliziten Dummy-Byte-Wert ergänzt werden. Diese Anpassung der Low-Level-Schnittstelle bietet aus Sicht der Performance sowohl den Vorteil eines besseren Durchsatzes, als auch eine niedrigere Speicherauslastung, da kein zusätzlicher Puffer notwendig ist. Dennoch wird von dieser Lösung abgesehen, da SD-Karten hier einen Spezialfall darstellen, der sich nicht auf die Gestaltung der Schnittstelle des SPI-Treibers auswirken sollte. Weiterhin würden sich z. B. bei der Verwendung verschiedener Geräte an einem Bus weitere Probleme ergeben, da diese nicht gezwungenermaßen die gleiche Konvention bzgl. Dummy-Bytes aufweisen. Als Lösung wird angestrebt, per define zwischen der momentanen Implementierung und einer Variante mit statischem Puffer wählen zu können. So kann je nach Anforderung entschieden werden, ob ein höherer Durchsatz oder mehr Speicher benötigt wird.

Strombedarf Der Strombedarf wird mit einem digitalen Sampling-Multimeter (Keithley DMM7510 7,5 Digits) gemessen. Dieses wird auf den 100 mA Messbereich fixiert um ein automatisches Umschalten des Messbereichs und damit

einhergehende Verzögerungen zu vermeiden. Es wird eine Abtastrate von 1000 Samples/s gewählt, da der Mittelwert über einen relativ langen Zeitraum von bis zu mehreren Minuten gemessen wird und mit einer niedrigeren Abtastrate eine höhere effektive Auflösung zur Verfügung steht. Damit nur der relevante Teil des Testablaufs gemessen wird, werden die über GPIO-Pins herausgeführten Start- und Stop-Signale des Testablaufs als Trigger für das Messgerät verwendet. Dazu werden die GPIO-Pins mit der digitalen I/O-Schnittstelle des Messgerätes verbunden. Zur Steuerung des Messgeräts wird ein einfaches Triggerflow Modell erstellt, das anhand der externen Signalflanken den Messvorgang steuert. Abbildung 6 zeigt das Modell zum Starten und Beenden des Messvorgangs. Die Wait-Zustände (1) und (3) blockieren die Ausführung des Flows bis eine steigende, respektive fallende Flanke auftritt. Die erste Digitize-Aktion (2) startet die Messung. Die zweite Digitize-Aktion (4) beendet die zuvor gestartete Messung.

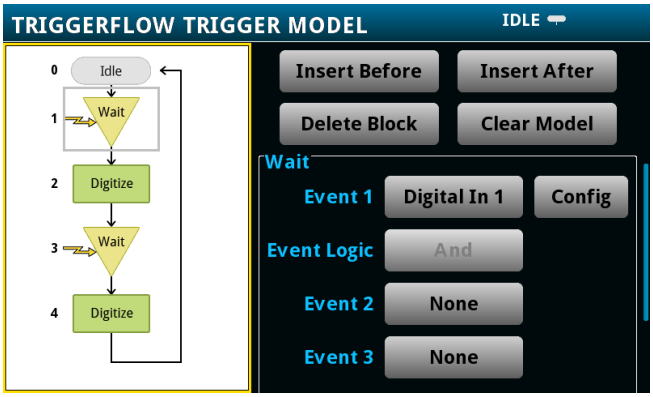


Abb. 6: Triggerflow Modell zum steuern des Messvorgangs

Tabelle 2 zeigt die Messergebnisse zur Leistungsaufnahme. Alle angegebenen Ströme wurden bei einer konstanten Versorgungsspannung von 3,3 V gemessen. Im Unterschied zu dem zuvor erläuterten limitierenden Schreibdurchsatz, zeichnet sich hier ein deutlich dynamischeres Bild ab. Der Strombedarf beim Initialisieren bewegt sich zwischen 11,22 mA und 37 mA. Beim Lesen dehnt sich diese Spanne auf 9,87 mA bis 39,05 mA aus, während Schreibvorgänge zwischen 12,64 mA und 40,35 mA fordern.

Vergleicht man die verschiedenen Zustände, zeigt sich, dass Schreiben meist nicht signifikant mehr Leistung benötigt als Lesen. Bei einer Mehrheit der Karten steigt der Energiebedarf beim Schreiben nur im niedrigen einstelligen Prozentbereich, bei den Karten 4 und 5 aber auch im Bereich von knapp 30 %.

Bei der Gegenüberstellung der Transferoperationen und dem Initialisieren fällt auf, dass das Initialisieren auf einigen Karten weniger, auf anderen Karten aber mehr Leistung benötigt. In besonderem Maße ist dies bei Karte 5 zu sehen, die beim Initialisieren knapp dreimal so viel Leistung aufnimmt.

Ein Aspekt, den die Messungen zur Leistungsaufnahme genauer beleuchten sollen, ist der auftretende Energieverbrauch im Ruhezustand. Um diesen zu Messen wird eine Testapplikation verwendet, welche die SD-Karte über die auto-init-Funktion initialisiert. Vor Beginn der Messung wird ein Block (512 B) von der Karte gelesen und überprüft, ob die Funktion ordnungsgemäß ausgeführt wird. Dies soll sicherstellen, dass die Kommunikation mit der Karte in der aktuellen Ausführungsinstanz funktioniert.

Die Messergebnisse zu diesem Versuch sind in der Spalte **Idle** zu finden und werfen zunächst Fragen auf. Der benötigte Strom steigt im Vergleich zu den Transferoperationen bei Karte 2 leicht an, bei drei der Karten wird er nur unwesentlich kleiner. Eine starke Abweichung zeigt sich bei Karte 5, welche im Ruhezustand nur noch 93 µA benötigt. Als Ursache für diesen Unterschied wird vermutet, dass der karteninterne Controller der Karten 1 bis 4 nicht direkt nach dem letzten Lesezugriff in einen sparsameren Modus wechseln kann, da für die Ausführung von internen Operationen meist das Taktsignal der SPI-Schnittstelle benötigt wird. Das Taktsignal ist bei dem sdcard_spi-Treiber allerdings nur aktiv, wenn mit der Karte kommuniziert wird, wie z. B. beim Lesen oder Schreiben von Daten. Mit einer weiteren Messung wird diese Vermutung untersucht. Der einzige Unterschied zur vorherigen Messung ist dabei, dass nach dem initialen Lesen eines Blocks 10 Dummy-Bytes an die Karte übertragen werden, wodurch auch das Taktsignal kurzzeitig weiter aktiv bleibt.

Die Ergebnisse dieser zweiten Messung zum Ruhezustand sind in der Spalte **Idle (2)** zu finden. Die Karten 1 bis 4 erreichen durch die Anpassung einen wesentlich sparsameren Betriebszustand mit einer Stromaufnahme zwischen 144 µA und 460 µA. Ebenfalls fällt auf, dass die Karte 5 nun mehr Strom aufnimmt als zuvor. Als Erklärung hierfür scheinen zwei Möglichkeiten plausibel: Der interne Controller wechselt durch die zusätzlichen Taktzyklen in einen Zustand, in dem er Verwaltungsoperationen o. ä. auf der Karte ausführt. Außerdem ist es möglich, dass die Pins, die zur Kommunikation genutzt werden, in einen anderen Konfigurationsmodus wechseln, wodurch Leckströme an den Logik-Pins entstehen. Letzteres wird als wahrscheinlicher erachtet, da sich bei mehrfacher Messdurchführung teilweise deutliche Schwankungen zeigen. Weitere Messungen könnten hier detailliertere Ergebnisse liefern. Es wird allerdings davon abgesehen, da hiermit zwar ein genaueres Verständnis der Karteneigenschaften in bestimmten Situationen nachvollzogen werden kann, sich daraus aber keine allgemeingültigen Aussagen ableiten lassen. Mehrfache Messdurchläufe zeigen bei den Karten 1 bis 4 keine signifikanten Abweichungen.

Nr.	Initialisieren	Lesen	Schreiben	Idle	Idle(2)
1	20,22 mA	22,36 mA	24,34 mA	19,03 mA	389 µA
2	30,24 mA	27,45 mA	27,16 mA	29,94 mA	460 µA
3	37 mA	39,05 mA	40,35 mA	39,76 mA	144 µA
4	11,22 mA	14,54 mA	19,38 mA	10,08 mA	291 µA
5	29,77 mA	9,87 mA	12,64 mA	93 µA	741 µA

Tabelle 2: Strombedarf verschiedener Karten-Modi

Nr.	Initialisieren	t_{Idle}
1	0,99 mW s	0,77 s
2	2,36 mW s	1,56 s
3	1,32 mW s	2,76 s
4	0,28 mW s	0,29 s
5	37,46 mW s	122,07 s

Tabelle 3: Energieverbrauch beim Initialisieren und daraus resultierende Zeitintervalle für den Ruhezustand

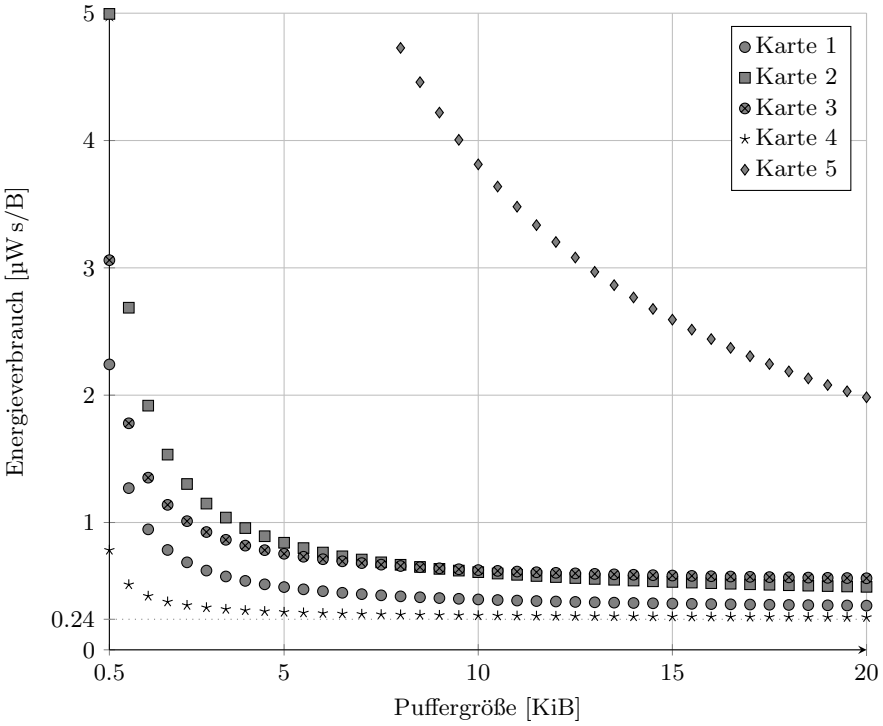


Abb. 7: Energieverbrauch in Abhängigkeit der Puffergröße

Die Karte im Ruhezustand zu belassen, wirkt sich nur dann positiv auf die Energiebilanz aus, wenn die Karte regelmäßig innerhalb eines bestimmten Zeitintervalls wiederverwendet wird. Die Obergrenze für dieses Zeitintervall wird in Tabelle 3 in der Spalte $\mathbf{t_{Idle}}$ gelistet. Diese Grenze ergibt sich aus dem Energieverbrauch der Karte beim Initialisieren und der kleinstmöglichen Leistungsaufnahme der Karte im Ruhezustand (**Idle** für Karte 5, sonst **Idle(2)**). Nur für Karte Nr. 5 liegt dieses Zeitintervall in einer Größenordnung, die für Messszenarien mit Duty-Cycle-Betreibsmodus interessant erscheint. Sobald seltener auf die Karte geschrienen wird, als das Zeitintervall $\mathbf{t_{Idle}}$ vorgibt, ist es sparsamer die Karte vollständig abzuschalten und vor einem Schreibvorgang neu zu initialisieren. Da für die Messdatenspeicherung in Feldtests auf den verwendeten Mikrocontrollern nur ein relativ kleiner Puffer bereitsteht, ist Karte 4 eindeutig zu bevorzugen. Für diese ist das Zeitintervall $\mathbf{t_{Idle}}$ nur 0,29 s lang. Dieses Zeitintervall ist für viele Messszenarien zu klein, um eine Menge von 512 B oder gar 5 KiB anzusammeln. Es wird daher davon ausgegangen, dass das Abschalten der Karte im Regelfall energieeffizienter ist, als die Karte im Ruhezustand zu belassen. Weitere Versuche könnten zeigen, ob sich dieser Sachverhalt unter Verwendung des expliziten Energiesparmodus von SD-Karten verändert. Die SD-Spezifikation garantiert allerdings nicht, dass dieser im SPI-Modus verwendet werden kann. Daher wird vorerst davon abgesehen.

5.2 Modularisierung und Transparenz zwischen SD-Karte und Applikation

Das Dateisystem FatFs wird in das Virtual-File-System „vfs“integriert, um die Austauschbarkeit von Dateisystemen zu ermöglichen. Hierfür wird ebenfalls ein neuer Pull-Request bereitgestellt[8]. Abbildung 8 zeigt die Anordnung der verschiedenen Komponenten der Logging-Applikation. Mit der vfs-Integration ist für die Applikation vollständig transparent, welches Dateisystem verwendet wird. Um diese Flexibilität auch in den unteren Schichten zu erreichen, wird die generische Datenträger Schnittstelle „mtd “in die RIOT-Implementierung der diskio Schnittstelle von FatFs integriert. Somit ist auch für FatFs transparent, welches Speichermedium verwendet wird. Entsprechend ist zwischen der mtd-Schnittstelle und dem sdcard_spi-Treiber die Komponente „mtd_sdcard“zu sehen, welche die mtd-Schnittstelle für den Treiber sdcard_spi implementiert. Für diese Implementierung wird ein separater Pull-Request erstellt[9]. In Zukunft reicht es aus, den Zugriff auf weitere Medien wie USB-Sticks über die mtd-Schnittstelle bereitzustellen, um deren Verwendung über FatFs respektive vfs zu ermöglichen.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde die Optimierung und Evaluierung der EH-Testplattform gezeigt. Es konnten sowohl die Hard- als auch die Software-Komponenten ver-

[8] `https://github.com/RIOT-OS/RIOT/pull/7104`
[9] `https://github.com/RIOT-OS/RIOT/pull/7324`

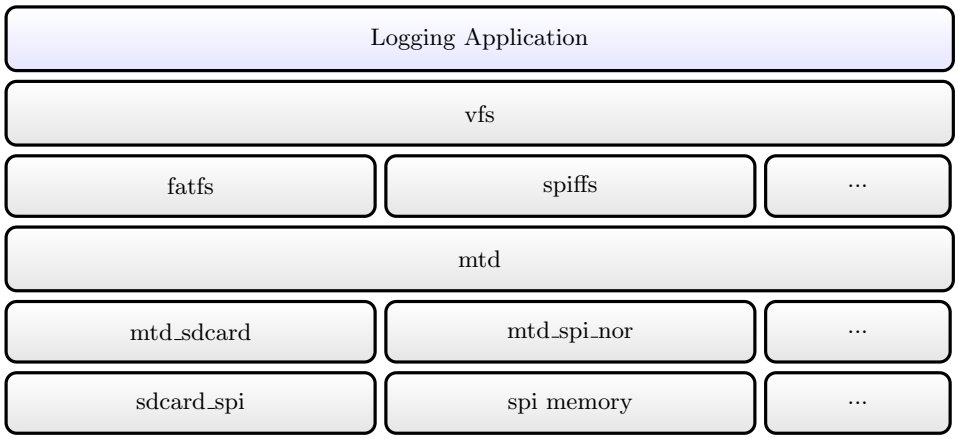


Abb. 8: Komponenten-Stack der Logging-Applikation

bessert werden. Die Messbereichsschalter des Messmoduls weisen nun geringere Abweichungen des Innenwiderstands ($< 1\,\mathrm{m}\Omega$) auf. Ein weiteres Ergebnis dieser Arbeit ist der erfolgte Umstieg auf die nucleo-l476 Plattform. Mit dieser sind Duty-Cycling Anwendungen nicht nur theoretisch verifizier- und messbar, sondern auch in Feldtests anwendbar und damit unter realistischen Bedingungen überprüfbar. Für die Plattform wurden die notwendigen Softwarekomponenten Implementiert, welche teilweise schon in den aktuellen master-Entwicklungszweig von RIOT übernommen wurden. Unter Verwendung dieser Komponenten wird im Ruhezustand eine Stromaufnahme von unter 1 µA bei aktivierter RTC erreicht. Die Leistungsfähigkeit und der Energieverbrauch der Logging-Komponente konnte außerdem untersucht und quantifiziert werden. Ohne weitere Optimierungen sind Transferraten von 231 KiB s bis 296 KiB s, bei einer Stromaufnahme zwischen 9,87 mA und 40,35 mA möglich. Die Initialisierungsdauer verschiedener Karten schwankt erheblich und bewegt sich zwischen 7,5 ms und 381,3 ms. Für weitere Arbeiten konnten weiterhin Optimierungen an der SPI-Integration und des Energieverbrauchs des Treibers sdcard_spi identifiziert werden.

Literatur

1. Baccelli, Emmanuel ; Hahm, Oliver ; Günes, Mesut ; Wählisch, Matthias ; Schmidt, Thomas C.: RIOT OS: Towards an OS for the Internet of Things. In: *Proc. of the 32nd IEEE INFOCOM. Poster.* Piscataway, NJ, USA : IEEE Press, 2013
2. Rottleuthner, Michel ; Schmidt, Thomas C.: Eine Testplattform für Energy Harvesting mit RIOT. In: *Proc. of the 16. GI/ITG KuVS Fachgespräch Sensornetze (FGSN2017).* Hamburg, Germany : HAW Hamburg, Dept. of Computer Science, Sep 2017, S. 35–38