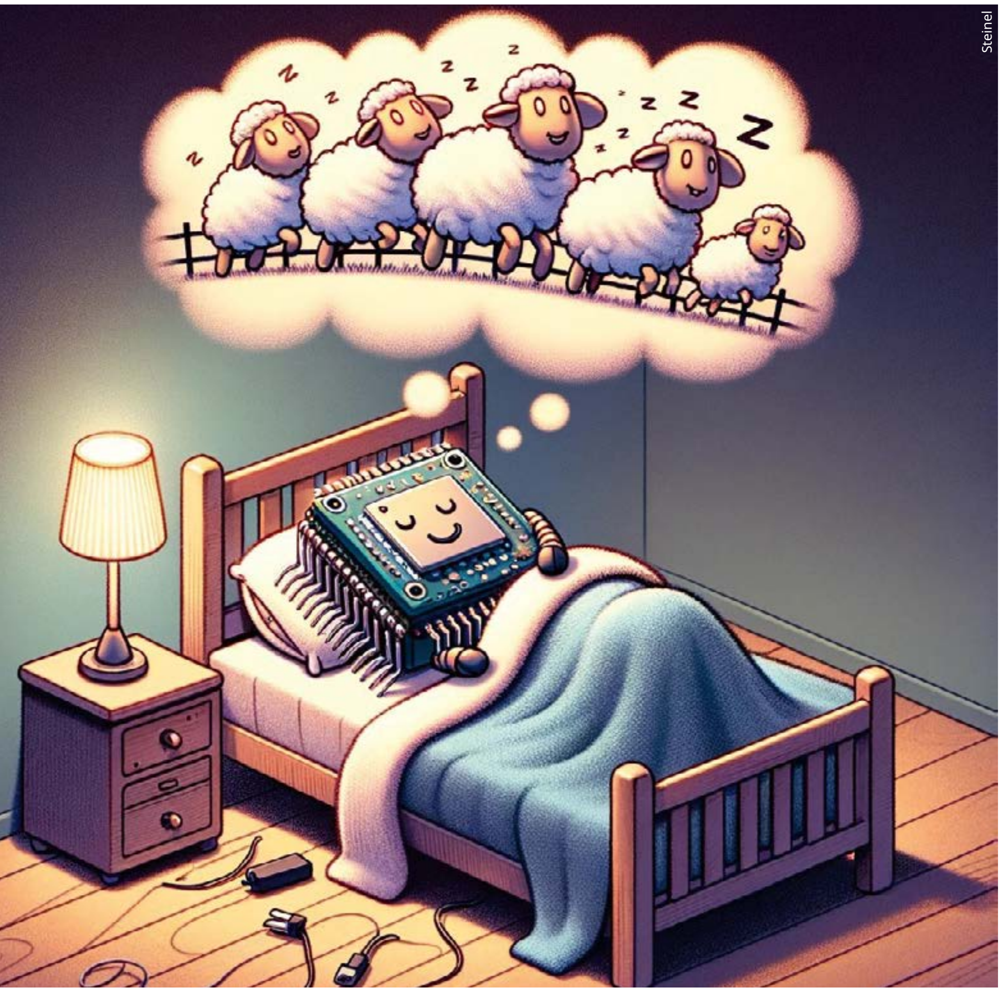
Der Microcontroller zählt Schäfchen

Microcontroller sind Weltmeister in Low-Power-Applikationen. Es ist faszinierend, was aus den kleinen Rechnern mit wenigen Mikroampere herausgepresst werden kann. Der wichtigste Low-Power-Trick ist, den Microcontroller schlafen zu lassen – und dies so lange wie möglich. Klar müssen zyklisch bestimmte Aufgaben und Operationen durchgeführt werden, aber danach «schnell zurück ins Bett».



Steinel

Je länger der Microcontroller schläft, desto weniger Strom verbraucht er.

Technisch spannend und herausfordernd sind Anwendungen mit Funkverbindungen. Nicht selten laufen diese mit einer Batterie und müssen haushälterisch mit den Ressourcen umgehen. Exemplarisch möchte ich von einem aktuellen Projekt und dessen Herausforderungen berichten.

In der hier beschriebenen Lösung werden sogenannte Smart-Buttons miteinander in einem Wireless-Mesh-Netzwerk verbunden. Die Smart-Buttons werden mit einer Batterie betrieben, haben ein e-Paper-Display, einen NFC-Scanner und drei mechanische Taster, um Eingaben durch den Benutzer zu registrieren und drahtlos an das Backend zu übertragen. Die Buttons sind an verschiedenen Orten im Gebäude verteilt. Sie formieren sich selbstständig zu einem Mesh-Netzwerk. Das Design der Elektronik ist auf minimalen Stromverbrauch ausgelegt: Die Spannungsversorgung läuft mit Batterie ohne DC/DC-Konverter, sämtliche Verbraucher können über Low-Q MOSFETs geschaltet werden und es wurden qualitativ hochwertige Komponenten verwendet. Die 800-mAh-Batterie soll eine Lebensdauer von zwei Jahren ermöglichen.

In einem Mesh-Netzwerk sind die Nodes nicht alle direkt mit dem Master wie in einem Stern-Netzwerk verbunden, sondern können Daten von Node zu Node weiterleiten, bis der gewünschte Empfänger erreicht wurde. Der Vorteil ist, dass der Master nicht von jedem Node direkt erreicht werden muss. Im Beispiel mit den Smart-Buttons können die Meldungen so von Gerät zu Gerät weitergegeben werden, bis die Meldung schlussendlich das Gateway erreicht.

Der Nachteil besteht darin, dass jeder Node in einem Wireless-Mesh normalerweise permanent auf Empfang sein muss, denn sie soll Daten uneingeschränkt empfangen und weiterleiten können. Technologien wie Thread, Zigbee oder BLE-Mesh konsumieren bei einem Nordic-nRF52-Device dauerhaft 7 mA. Dies ist im Vergleich zu anderen Herstellern bereits ein sehr tiefer Wert, würde aber die Batterie für den Smart-Button innerhalb von fünf Tagen leer saugen.

Ausgeklügelter Low-Power-Trick

Auf der Suche nach einer Alternative stiess ich auf das Open Source Contiki-NG-Framework. Contiki-NG setzt die IEEE-802.15.4-2015-Spezifikation um und versendet die Daten in einem Mesh in kleinen Timeslots, verwendet also einen ausgeklügelten Low-Power-Trick. Die Nodes werden in deren Zeitbasis exakt synchronisiert und sind für maximal 10 ms gleichzeitig auf Empfang. Danach deaktivieren sie den Empfang und gehen für die folgenden 990 ms in den Low-Power-Mode. Die Zeitsynchronisierung ist nicht sehr kompliziert. Die Nodes müssen jedoch laufend synchronisiert sein und eine zeitliche Abweichung (Drift) zum Koordinator berech-

Steinel

Messung des Stromverbrauchs vor und nach dem Redesign von Contiki-NG.

nen, damit die Signallaufzeit aufgrund der örtlichen Distanz zum Koordinator kompensiert werden kann. Diese Lösung ist nicht für Anforderungen mit tiefer Latenz geeignet, was jedoch bei IoT-Anwendungen in der Regel nicht relevant ist.

Die Idee war gut, aber in der Entwicklung läuft es bei komplexen Projekten selten nach Plan. Bei den ersten Tests mit Contiki-NG zeigten die Messungen einen Stromverbrauch des nRF52 von 480 µA (Lebensdauer rechnerisch nur 70 Tage). Um die definierten Anforderungen an die Batterielebensdauer von zwei Jahren zu erreichen, dürfte der Stromverbrauch mit den benötigten Peripherien bei ca. 50 µA liegen.

Tauchen wir deshalb noch etwas tiefer in die Technik ein: Frameworks wie Zephyr, FreeRTOS und auch Contiki-NG verwenden einen periodischen SysTick, meistens im Bereich von 1000 Hz, um zwischen Programm-Tasks zeitgenau umzuschalten und ein Multi-Threading zu simulieren. Bei Contiki-NG muss dieser SysTick sehr genau sein, ansonsten würden die Nodes nicht zu der exakt gleichen Zeit auf den aktiven Empfang wechseln.

Durch tiefe Nachforschungen im Github-Code von Contiki-NG konnte ich den Grund für den hohen Stromverbrauch ausfindig machen. Die Implementierung für den Nordic-Microcontroller verwendet die TIMER-Peripherie als Basis für den SysTick, der selbst wiederum den High Frequency Quarz (HFXO) mit 32 MHz aktiviert. Dies allein verbraucht in etwa 500 µA. Der Nachteil ist, dass der SysTick während des Low-Power-Modes nicht deaktiviert werden kann, denn ansonsten wären zwischen Einschlafen und Aufwachen nur ein Tick vergangen und die Zeitbasis auf dem Node wäre falsch.

Hier stellte sich mir die Frage: Muss man also mit dem hohen Stromverbrauch leben, oder gibt es eine Alternative zum TIMER und dem HFXO? Tatsächlich bietet der Nordic MCU die Möglichkeit, einen zusätzlichen Low Frequency Quarz (LFXO) mit 32.768 kHz zu verwenden. Sobald dieser bestückt wird, kann der RTC (Real Time Clock für einen ausreichend genauen SysTick verwendet werden, und wir können die Aktivierung des HFXOs und TIMERs auf den kurzen Timeslot im aktiven Mode reduzieren.

Microcontroller schläft fast immer

Das Resultat machte Freude. Der Microcontroller schläft zu 99,79 Prozent und benötigt im Durchschnitt bescheidene 55 µA. Trotzdem ist jeder Smart-Button ein vollwertiges Mitglied des Wireless-Mesh. In Thread und Zigbee würden wir von einem Full Function Device (FFD) Node sprechen.

Eine Low-Power-Lösung ist in der Entwicklung grundsätzlich immer aufwendig. Viele spannende und herausfordernde Effekte passieren erst in diesem Modus. Zum Beispiel braucht der HFXO eine gewisse Zeit, bis er stabil läuft und er muss ein paar Mikrosekunden vor dem geplanten Einsatz gestartet werden. Unser Redesign der Contiki-NG-Nordic-Implementierung war mit ca. 1000 Code-Zeilen umfangreicher als anfänglich gedacht, und ist nun im Github in der aktuellen Version vollständig integriert. ■

Marcel Graber
Electronics & Firmware bei der Steinel Solutions AG

► **Contiki-NG: https://docs.contiki-ng.org/**