IoT-Anbindung mit schmalen Ressourcen? Kein Problem!

Erfahrungen mit Embedded-Software für kleine IoT-Knoten

Andreas Foltinek, IMACS GmbH

1. Vorspann

Eine Datenfernkommunikation zwischen technischen Systemen ist so alt wie die Systeme selbst. Durch die immer weitere und schnellere Web-Infrastruktur liegt es nahe, diese für technische Systeme zu nutzen, wofür sich die Begriffe IoT bzw. IIoT etabliert haben.

Auf Basis von Linux/Windows o.ä. ist mittlerweile die Realisierung der IoT-Fähigkeit softwareseitig überschaubar. Leider eignen sich jedoch Linux und Co. basierte Systeme aus vielen Gründen (Bauteil-/Herstellungskosten, Bootvorgang, Komplexität, Stabilität, Energiebedarf) für viele Anwendungen nicht, insbesondere wenn höhere Stückzahlen und damit Effektivität erforderlich sind.
Die Realisierung auf "kleinen" Embedded-Targets stellt sich hingegen als größere Software-Herausforderung dar.

Der folgende Praxisbericht zeigt, dass auch mit den Ressourcen von kleineren 32-Bit Singlechip-Controllern die IoT-typischen Features, wie TCP-IP-Protokolle, Web-Client/-Server, Push-Mechanismen und Firmware-Updates via Cloud, problemlos realisierbar sind – auch unter Erfüllung der aktuellen Security-Standards. Hierzu werden im Folgenden die Erfahrungen aus mehreren derartigen Projekten bezüglich der Hardware, Toolchain, RTOS, Stacks, Treiber und deren Besonderheiten bei der Integration, Inbetriebnahme und täglichen Anwendung erläutert.

2. IoT-Konnektivität - Klassifizierung des Umfelds

Für die Realisierung eines IoT-fähigen Gerätes bzw. für die Integration einer IoT-Schnittstelle in ein Gerät sollten zunächst die relevanten Einsatzfälle und die sich daraus ergebenden Anforderungen klar definiert sein. Hierbei sind folgende Aspekte zu berücksichtigen:

2.1 Use Case

- Stückzahl und damit Kostenrelevanz sowie mögliche Hardwareausstattung des Systems
- Typischer Einsatz: Wie oft muss mit der Cloud kommuniziert werden? Wie oft/schnell wird über die Cloud das Gerät "bedient"
- Erforderliches Datenaufkommen (punktuell) und Datenvolumen (längerfristig im Mittel)
- Security-Anforderungen z.B. durch das Betriebsumfeld. Jedoch wird nicht jedes Gerät, das über Web-Technologie kommuniziert, am Internet betrieben. Viele firmenlokale Intranet-Anwendungen sind nicht minder interessant.
- Weiterer Geräteaufgaben neben der IoT-Konnektivität (Steuerungen, Bus-Systeme, HMI, …) und der damit zusätzliche Ressourcenbedarf.

- Verfügbarkeit bzw. erforderlicher Offlinebetrieb und Datenpufferung bei Internet-Ausfall
- Anwenderumfeld: Installation durch Fachkräfte, bis hin zu technischen Laien
- Konfigurierbarkeit: Wie und mit welchen Hilfsmitteln wird das Gerät zu Beginn konfiguriert, welche Handlungen/Angaben sind für den Anwender zumutbar?
- Cloud-Typ: Im klassischen Ansatz stellt die Cloud einen Web-Server zur Verfügung. Neuere Konzepte arbeiten hingegen auf Basis von MQTT oder OPC-UA. Projektabhängig bietet beides Vor- und Nachteile, die es abzuwägen gilt.
- Cloud-Flexibilität: Handelt es sich um ein Bestandsystem "as it is" (z.B. das der "großen Anbieter") mit fixen Schnittstellen/Funktionen oder ist eine individuelle Cloud-Lösung mit beeinflussbarer Verarbeitung möglich/geplant?
- Updatefähigkeit des Gerätes per Cloud
- Reaktionsschnelligkeit (Zeiten für Ereignis am Gerät nach Cloud bzw. Cloud nach Gerät)

2.2 Einsatzarten und gerätelokale Funktionen/Features

Welches ist die primäre Funktion des Gerätes?

- Gateway, d.h. die Hauptaufgabe ist die Datenvermittlung, -sammlung und -aufbereitung
- oder Gerät mit eigenen, anspruchsvollen Hauptfunktion, z.B. Safety, Echtzeit, HMI, Kommunikationsschnittstellen zu anderen Systemen, Massenspeicher wie USB-Sticks/SD-Karten, analoge/digitale EAs

2.3 Physikalische Schnittstellen zur Cloud

Welche Verbindungstechnologien sollen zum Einsatz kommen? Bzgl. dieser stehen prinzipiell folgende zur Verfügung und sind etabliert:

- Primär TCP/IP-basiert: Ethernet / WLAN / Powerline-LAN / Mobile (GPRS, LTE, …)
- Sekundär TCP/IP basiert: LPWAN (Sigfox, LoRa)

Letztere bilden eine eigene Klasse von Systemen und benötigen technologiebedingt deutlich weniger Ressourcen, sind jedoch daher auch nur für einen bestimmten Anwendungsbereich relevant. Diese sollen im Folgenden nicht weiter betrachtet werden.

2.4 Security

Die zunächst naheliegende Verwendung von sicheren Tunneln (z.B. VPN) scheidet leider in den meisten Fällen aufgrund der aufwendigen Installation/Nichtakzeptanz aus. Für die/den unkomplizierte Installation/Betrieb eines IoT-Gerätes muss vielmehr ein freier Internetanschluss ausreichend sein.
Angriffe sind damit jedoch in verschiedenen Formen (aushorchen, manipulieren) an allen Stellen der Verbindung möglich. Durch die Verwendung etablierter, teilweise sogar direkt lesbaren Formate/Protokolle (XML, JSON, HTTP, …) wird diese Thematik noch verschärft.

Es sind daher folgende Schutzmechanismen (einzeln oder in Kombination) einzuplanen:

- Der Aufbau einer Verbindung darf nur vom IoT-Gerät zur Cloud erfolgen, d.h. das Gerät stellt einen Web-Client dar (Ausnahme: Installationstätigkeiten, falls das Gerät über kein eigenes HMI oder andere Mechanismen verfügt)
- Die ausgetauschten Daten sollten mit einem zertifikatsbasierten Mechanismus (aktuell TLS1.2) verschlüsselt werden.
- Nötige Zugangskanäle von außen (z.B. Webserver zur Konfiguration) sollten im Umfang extrem funktionsreduziert sein.

Beim Zertifikatseinsatz bestehen ferner folgende Variationsmöglichkeiten:

- Verwendung von eigengenerierten Zertifikaten oder die eines bekannten Zertifikatsanbieters
- zeitlich begrenzte oder unbegrenzte Zertifikatsgültigkeit
- Verwendung eines Zertifikats für alle Exemplare eines Gerätes oder exemplarspezifische Zertifikate (besser, jedoch aufwändiges Handling).

Hinzukommend möchten Kunden immer häufiger die vorhandene IoT-Schnittstelle nutzen, um sich direkt lokal (via mobilem Gerät) oder auch mit "fremden", nicht zertifikatsunterstützenden Systemen (z.B. SmartHome, Gebäudetechnik, …) zu koppeln. Je nach Art der Kommunikation (Client/Server) ist wie oben genannt zu verfahren.

3. Hardwareplattformen

Was ist "schmal"? Um die Kosten möglichst gering zu halten, wurden Targets der Klasse Cortex M3/M4 bzw. PIC32 MX/MZ mit prozessorinternem Speicher im Bereich ab 256kB-Flash, 128kB-RAM, 80MHz bis 2048kB-Flash, 512kB-RAM, 200MHz eingesetzt.

Auch bei der kleinsten Hardware-Konfiguration wurden trotz LAN-und WLAN-Unterstützung weniger als 50% der Ressourcen für die IoT-Funktionalität benötigt. Als Schnittstelle wurde stets ein 100 MBit-Ethernet mit externem PHY und der controllereigenen MAC-Unit realisiert sowie zusätzlich in verschiedenen Kombinationen WLAN und Powerline-LAN über SPI hinzugefügt.

Zur Realisierung der primären IoT-Fähigkeit war weiter lediglich ein serieller EEPOM erforderlich (Speicherung von Zugangsdaten). Um applikationsspezifische Anforderungen (Offline-Buffering, Update, Protokollierung) zu erfüllen, wurde diese teilweise mit einem Data-Flash, USB-Master-Schnittstelle (für Memory-Stick) und einer SD-Karte ergänzt. Das PCB wurde in den meisten Fällen zweilagig ausgeführt und erfüllte trotzdem die EMV-Anforderungen vollständig und auf Anhieb.

4. Softwarekomponenten und -umfang

Für die Erstellung der Basis-Software (untere Schichten) wurden die herstelleroffenierten Tool-Chains und Libraries in Kombination mit FreeRTOS verwendet. Als TLS-Stack kam, falls erforderlich, Wolf-SSL zum Einsatz.
Neben den verschiedenen HAL-Teilen beinhalteten die Systeme bis zu 2 Web-Clients (Zugang zur Cloud via LAN und WLAN bzw. Powerline-LAN) und einem

Web-Server (Zugang für lokale App) ausgestattet (damit auch 3 Instanzen des TLS-Stacks).
Hinzukam die eigentliche kundenspezifische Anwendungsschicht (Nutzdatenkommunikation, Auswerten und Aufbereiten der Daten etc.), die modellbasiert entwickelt und via Code-Generator erzeugt wurde. Ständig wachsende Modell-Bibliotheken erleichterten hierbei die Entwicklung der späteren Projekte erheblich und ermöglichten sogar die Vorabausführung/-test der Applikation auf einer PC-Plattform bei voller IoT-Funktionalität.

5. Integration
Die Verwendung des Herstellercodes bedeutete trotz funktionsfähiger Referenzdesigns eine naturgemäß zeitintensive Einarbeitung und die Aufdeckung von Bugs, die jedoch kurzfristig seitens des Herstellers behoben wurde. Ebenfalls bedurften neue Herstellerversionen auch verschiedener Anpassungen des bis dahin bei uns entstandenen Applikationscodes. Dies zahlte sich jedoch bei der Realisierung der Folgeprojekte auf dieser Plattform aus.
Das modellbasierte Entwicklungssystem bot ferner die Möglichkeit einer PC-Visualisierung, die via Schnittstelle an das Target gekoppelt ist und alle Variablen sowie Ereignisse online darstellt und in Logs aufzeichnet. Dies erleichterte das Debuggen von Fehlern, die hier oft erst nach Tagen des Dauerbetriebes auftraten, erheblich.

6. Erfahrung und Erkenntnisse
Bei den bisher durchgeführten Projekten dieser Art wurden folgende Aspekte als wichtig und bemerkenswert angesehen:

- Naturgemäß werden bei derartigen Systemen viele Daten über Buffer ausgetauscht, an deren Größe durch die allgemeine Ressourcen-Beschränktheit gespart wird. Hier sollte durch Prüfmechanismen erkannt werden, wenn diese zu klein dimensioniert wurden.
- Da Probleme oft erst im Zusammenspiel mit der Cloud auftreten/erkannt werden, ist eine korrelierte Protokollierung der Daten intern mit denen zur/von der Cloud zur Analyse unabdingbar.
- Durch viele parallele und häufige Verarbeitungen sollte das Scheduling der Tasks und die Aufteilung der Rechenzeit gemessen und bewertet werden. Verschiedene Prozesse wurden hier nicht zeitoptimal geschrieben, was sich oft erst längere Zeit nach der Implementierung auswirkt und dann nativ nicht erkannt wird.
- Frühzeitig Debug-Möglichkeiten festlegen und bereitstellen (WireShark, Cloud-Protokolle, …)
- Die Sicht des Kunden und der Produktion ist zyklisch zu prüfen. Das Handling mit IT-/Web-Technik benötigt viele Schritte, die für einen Entwickler normal, für einen Kunden jedoch teilweise nicht zumutbar sind und entsprechend anders ausgelegt werden müssen.

Autor
Andreas Foltinek studierte Elektrotechnik an der Universität Stuttgart und kann auf über 30 Jahre Embedded Hard- und Softwarepraxis zurückblicken.

1994 gründete er IMACS und ist seither dort als Geschäftsführer für Forschung und Entwicklung verantwortlich. Als Initiator eines UML basierten CASE-Tool-Systems sowie eines modularen Open-Source Hardware-Systems gilt seine Leidenschaft der Vereinfachung und Redundanzvermeidung bei der Entwicklung von Embedded Hard- und Software durch Modularität, Wiederverwendung und insbesondere modellbasierte Methoden und Generierung.

Kontakt
Email: andreas.foltinek@imacs-gmbh.de