Diplomarbeit

Höhere Technische Bundeslehranstalt Leonding
Abteilung für Informatik

Mesh Netzwerk

Eingereicht von: **Tim Untersberger, 5BHIF**
Stefan Waldl, 5BHIF
Datum: **4. April 2020**
Betreuer: **Thomas Stuetz**
Projektpartner: **HTBLA Leonding**

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorgelegte Diplomarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Gedanken, die aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Leonding, am 4. April 2020

Tim Untersberger, Stefan Waldl

Declaration of Academic Honesty

Hereby, I declare that I have composed the presented paper independently on my own and without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such.

This paper has neither been previously submitted to another authority nor has it been published yet.

Leonding, April 4, 2020

Tim Untersberger, Stefan Waldl

Zusammenfassung

Zusammenfassung in Deutsch + Bild am Ende

Abstract

Zusammenfassung in Englisch + Bild am Ende

Autoren der Diplomarbeit

Danksagung

If you feel like saying thanks to your grandma and/or other relatives.

Inhaltsverzeichnis

Kapitel 1

IstZustand

1.1 hello

Kapitel 2

SollZustand

Kapitel 3

Verwendete Technologien

3.1 Over The Air Update (OTA)

Problemstellung

Der Workflow mit Mikrocontroller in Produktivsystemen ist aufwändig, da sie sich meist an ungünstigen Orten befinden. Bis jetzt musste man seine Datenquelle physisch mit dem Chip verbinden um neue Firmware auf den Mikrocontroller zu spielen. OTA ermöglicht es den Mikrocontroller mit neuer Firmware zu versorgen, dabei muss der ausgewählte Microcomputer lediglich mit einem Wlan-Router verbunden sein.

Wie OTA funktioniert

Am wichtigsten ist es, dass man die Partitionstabelle an das ausgewählte Programm anpasst. Die Standard-Partitionstabelle ist je nach Hersteller anders. Um OTA zu ermöglichen ist es notwendig im Partitionstabelle mindestens eine, ausreichend große, Partition zu vergeben. Dabei ist es wichtig, dass die OTA-Partition ausreichend Speicher für die gewünschte Firmware hat.

Damit der OTA-Vorgang funktionieren kann, muss zuerst ein lauffähiges Programm auf dem Mikrocontroller gestartet sein. Danach muss z.B. der ESP32 die neue Firmware über die bestehende Internet oder Intranet-Verbindung in die zuvor definierten Partition herunterladen. Anschließend muss nur noch der ESP32 mit der aktualisierten Partition neu gestartet werden.

Implementation

Die ausgewählte Struktur für die Diplomarbeit kann man im folgenden Bild sehr gut sehen.

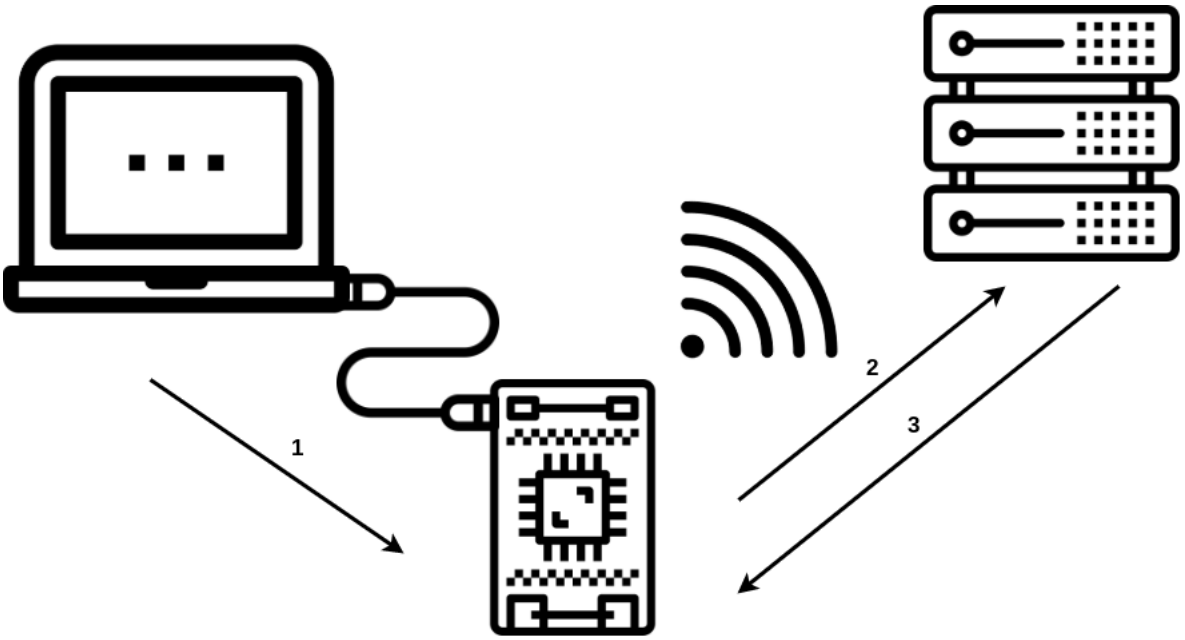

Abbildung 3.1: OTA Erklärung (Quelle: eigene Darstellung)

3.2 Mesh Netzwerk

3.3 Nodejs

Nodejs ist eine Laufzeit die auf der Javascript Engine von Chrome aufbaut. Nodejs wurde für das Backend des OTA Servers verwendet.

3.3.1 Warum Nodejs?

Java EE wurde hier nur als ein Beispiel gewählt. Folgendes gilt für jedes Framework das HTTP requests wie JavaEE behandelt. Bei der Überlegung welche Sprache bzw. welches Framework für das backend gewählt wird, war die Entscheidung sehr leicht. Die Anforderungen des OTA Servers sind sehr einfach. Der Server dient nur zur Bereitstellung der Firmwares und beschreibende Informationen über die Firmwares selber. Da dies nicht CPU intensiv ist sondern I/O intensiv, wurde in diesem Fall Nodejs gewählt.

3.3.2 Event Loop

Was ist der Event Loop?

Nodejs selber ist single-threaded deswegen ist der Event Loop einer der wichtigsten Bestandteile von Nodejs. Der Event Loop erlaubt es nicht-blockenede I/O Operationen

durchzuführen. Dies passiert durch die Abladung auf den Kernel von so vielen Operationen wie möglich.

Die meisten modernen Kernels sind multi-threaded. Das bedeutet, dass sie mehrere Operationen im Hintergrund unterstützen.

Wie funktioniert der Event Loop?

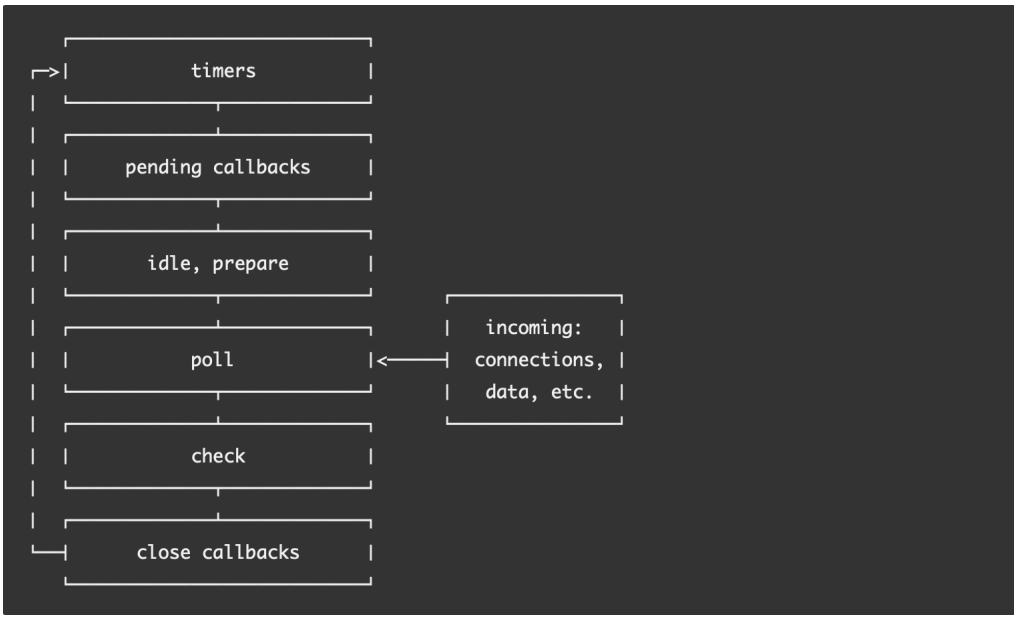

Abbildung 3.2: Nodejs Event Loop [1]

Jeder Block im obigen Bild wird als Phase bezeichnet. Jede Phase verfügt über eine First-In-First-Out-Warteschlange (FIFO-Warteschlange) mit auszuführenden Rückrufen. Während jede Phase auf ihre Weise etwas Besonderes ist, führt sie im Allgemeinen, wenn die Ereignisschleife in eine bestimmte Phase eintritt, alle für diese Phase spezifischen Operationen aus und führt dann Rückrufe in der Warteschlange dieser Phase aus, bis die Warteschlange erschöpft ist oder die maximale Anzahl von Rückrufen ausgeführt hat. Wenn die Warteschlange erschöpft ist oder das Rückruflimit erreicht ist, wechselt die Ereignisschleife zur nächsten Phase und so weiter.

Da für jeden dieser Vorgänge möglicherweise mehr Vorgänge geplant werden und neue Ereignisse, die in der Abfragephase verarbeitet wurden, vom Kernel in die Warteschlange gestellt werden, können Abrufereignisse in die Warteschlange gestellt werden, während Abrufereignisse verarbeitet werden. Infolgedessen können lange laufende Rückrufe dazu führen, dass die Abfragephase viel länger als der Schwellenwert eines Timers läuft.

Phasen

- **timers:** In dieser Phase werden von setTimeout () und setInterval () geplante Rückrufe ausgeführt.
- **pending callbacks:** führt I/O Callbacks aus, die auf die nächste Iteration verschoben werden
- **idle, prepare:** wird nur intern verwendet
- **poll:** neue I/O Events abrufen; I/O bezogene Callbacks ausführen (fast alle mit Ausnahme von schließ Callbacks, die von Timers geplant werden, und setImmediate ()); Der Knoten wird hier gegebenenfalls blockiert
- **check:** setImmediate() Callbacks werden hier ausgeführt
- **close callbacks:** schließ Callbacks werden ausgeführt

Zwischen jedem Durchlauf des Event Loops prüft Nodejs, ob es auf asynchrone I/O oder Timer wartet, und fährt sauber herunter, wenn keine vorhanden sind.

[2, Zitiert von der offizielen Nodejs Website]

3.4 Platform IO

3.5 Docker

3.6 Docker Compose

3.7 ESP IDF Utility lib

3.8 React

3.9 Yarn

3.10 Webpack

Kapitel 4

Ausgewählte Aspekte

4.1 NGINX sichern mit letsencrypt

4.2 EspWifiManager Implementation

4.3 Die Wichtigkeit von Erase Flash

4.4 ELF vs Bin

Kapitel 5

Summary

Here you give a summary of your results and experiences. You can add also some design alternatives you considered, but kicked out later. Furthermore you might have some ideas how to drive the work you accomplished in further directions.

Literaturverzeichnis

[1] Nodejs Event Loop.
`https://nodejs.org/en/`.
Abgerufen am 7.2.2020.

[2] Nodejs Event Loop Erklärung.
`https://nodejs.org/uk/docs/guides/event-loop-timers-and-nexttick/#event-loop-explained`.
Abgerufen am 7.2.2020.

Abbildungsverzeichnis

Tabellenverzeichnis

Project Log Book

Date	Participants	Todos	Due

Anhang A

Additional Information

If needed the appendix is the place where additional information concerning your thesis goes. Examples could be:

- Source Code
- Test Protocols
- Project Proposal
- Project Plan
- Individual Goals
- ...

Again this has to be aligned with the supervisor.

Anhang B

Individual Goals

This is just another example to show what content could go into the appendix.