Titel: ESP32 für den Einsatz in Geräten der Industrie 4.0
Autor: Originalartikel von Dror Gluska, übersetzt und bearbeitet von Dipl.Ing. Stefan Tauschek, Macnica Europe GmbH
Datum: 21.3.2018

ESP32 für den Einsatz in Geräten der Industrie 4.0

Applikationsschrift

Industrie 4.0[1] als industrielle Variante des Internet-of-Things meint die technologische Migration bestehender Produktionssysteme in vielseitig vernetzte, automatisierte und durch smarte Algorithmen gesteuerte Anlagen. Der Markt für Komponenten und Dienstleistungen dafür wird sich bis zum Jahr 2022 nach einer Prognose von Markets & Markets[2] auf einen Umfang von 152 Milliarden US-Dollar entwickeln.

Mikrocontroller auf allen Ebenen und in nahezu allen Geräten werden dabei eine wichtige Rolle spielen, doch werden sich Programmierparadigmen ganz erheblich ändern und weitaus komplexere Protokolle, Schnittstellen und Szenarien antizipieren als dies gegenwärtig der Fall ist. Profitieren davon werden zweifelsfrei Embedded-Software-Entwickler, die diese Technologien beherrschen und ein erheblicher Mangel dieser Spezialisten am Arbeitsmarkt ist nicht schwer zu prognostizieren.

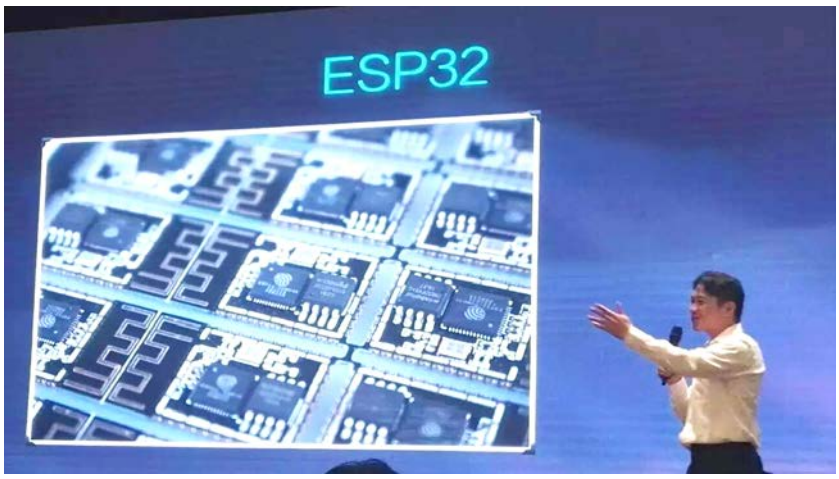

Abbildung 1: Ankündigung des ESP32 als „Cloud on Chip“ Prozessors im September 2016 (Quelle: Espressif News)

In diesem Artikel geht es nicht um die Suche nach dem ultimativen Killerprozessor für das IoT, sondern um einen Weg zur möglichst schnellen Entwicklung einschlägiger Anwendungen mit maximaler Skalierbarkeit.

Industrie 4.0 als Oberbegriff spielt eine wichtige Rolle bei der industriellen Optimierung, indem Systeme überwacht und Sensordaten statistisch oder durch Lernalgorithmen analysiert werden und sich damit Fehlerprognosen, Lastanalysen und Optimierungen durchführen lassen.

---

1 Siehe: Guide To Industry 4.0, https://www.i-scoop.eu/industry-4-0/
2 Siehe: http://www.rnrmarketresearch.com/industry-40-market-by-technology-industrial-robotics-cyber-security-internet-of-things-3d-printing-advanced-human-machine-interface-big-data-augmented-reality-virtual-reality-artificial-intelligence-ver-st-to-2022-market-report.html

Lohn für diese Investitionen sind Produktionskostensenkung und Produktivitätswachstum, zwei essentielle Ziele eines jeden am globalen Markt operierenden Unternehmens.

Während einige der großen Marktteilnehmer bereits fertige Lösungen anbieten, sind alle technologischen Komponenten längst in der Open-Source Community verfügbar und das Internet-der-Dinge ist keine Spielwiese mehr von Geeks & Nerds, sondern bereit für die Migration in industrielle Prozesse und Produktionseinrichtungen.

Tags: *espressif, esp32, esp32-idf, rtos, iot, internet of things, industry 4.0,*

Titel: ESP32 für den Einsatz in Geräten der Industrie 4.0
Autor: Originalartikel von Dror Gluska, übersetzt und bearbeitet von Dipl.Ing. Stefan Tauschek, Macnica Europe GmbH
Datum: 21.3.2018

Einsatz in der Industrie

Obwohl der ESP32 kein Ersatz für kommerziell erhältliche PLC [3] /SCADA [4] Geräte ist, kann er diese Zielsetzungen doch recht elegant erfüllen, als Sensor-, Relais- oder SSR-Modul[5] beispielsweise. Connectivity ist eine recht einfache Übung für das SoC und falls wirklich eine paar I/Os fehlen sollten, so kann ein simpler externer Multiplexer helfen, solange das notwendige Timing dabei nicht verletzt wird.

Wie aber steht es um die Software?

Derzeit gilt Node-Red[6] als populärste IoT Entwicklungs- und Steuerungssoftware. Sie ist einfach zu bedienen und es existieren viele Tutorials dazu sowohl als Fachartikel als auch als Demonstrationsvideos.

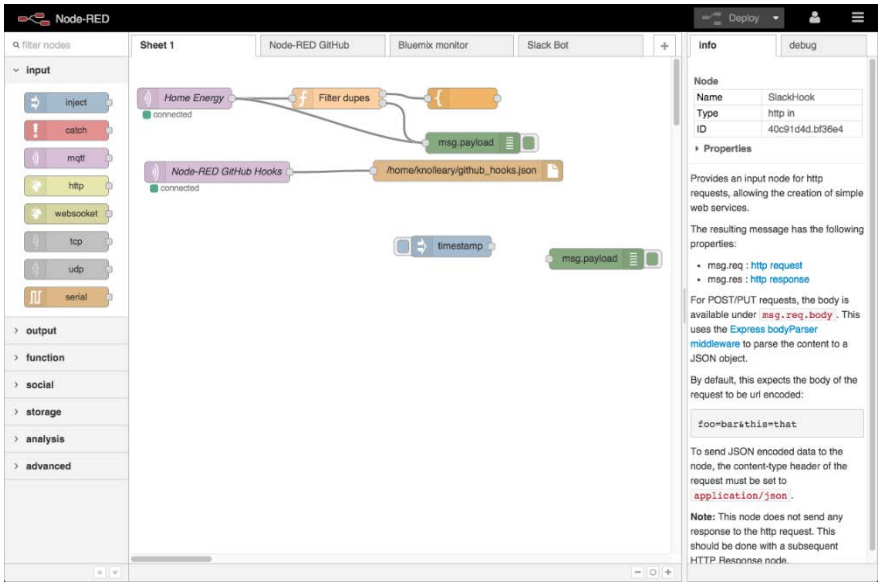

Abbildung 2: Screenshot Node-Red

Weitere Optionen sind Eclipse Kura[7] und Project Flogo[8].

Geht es hingegen mehr um industrielle Anwendungen, werden andere Frameworks oder Entwicklungsumgebungen bevorzugt, hier eine kleine Auswahl aus der Open-Source-Gemeinde:

- RapidSCADA[9] - Rapid SCADA ist eine freie Open-Source SCADA-Software mit umfangreicher Ausstattung.
- Eclipse NeoSCADA™[10] ist flexibel. Keine Out-of-the-Box-Lösung, sondern eine Sammlung von Tools, die auf viele verschiedene Arten kombiniert werden können. Es bietet Entwicklungsbibliotheken, Schnittstellenanwendungen, Massenkonfigurierungstools, Front-End- und Back-End-Anwendungen.
- IndigoSCADA – Ein SCADA-System mit kleinem Speicherbedarf, vollständig in C und C++ entwickelt, unterstützt zahlreiche Betriebssysteme und Frontend-Protokolltreiber.
- ScadaBR is freie Open-Source-Software für die Entwicklung von Automatisierungs-, Datenerfassungs- und Überwachungsanwendungen.

---

3 PLC = Programmabloe Logic Controller = Speicherprgrammierbare Steuerung (SPS)
4 SCADA = Supervisory Control and Data Akquisition, siehe: https://de.wikipedia.org/wiki/Supervisory_Control_and_Data_Acquisition
5 SSR = Solid State Relay
6 Node-Red, siehe: https://nodered.org/
7 Eclipse-Kura, siehe: https://www.eclipse.org/kura/
8 Project Flogo, siehe: http://www.flogo.io/
9 RapidScada, siehe: http://rapidscada.org/
10 Eclipse neoSCADA, siehe: http://www.eclipse.org/eclipsescada/

- SZARP ist ein voll ausgestattetes SCADA-Systen zur Überwachung sich nur langsam verändernder industrieller Prozesse beispielsweise bei städtischen Heizwerken. SZARP ist freie Software, veröffentlicht unter GNU General Public License 2.0.

Warum ESP32?

Während es auf dem Markt zahlreiche Schlüsselbauelemente gibt, die sich zum Einsatz in IoT-Geräte eignen, soll es im vorliegenden Artikel speziell um den Espressif ESP32 gehen, der sich nach Einschätzung des Verfassers für Anwendungen im Industrie 4.0-Umfeld besonders eignet.

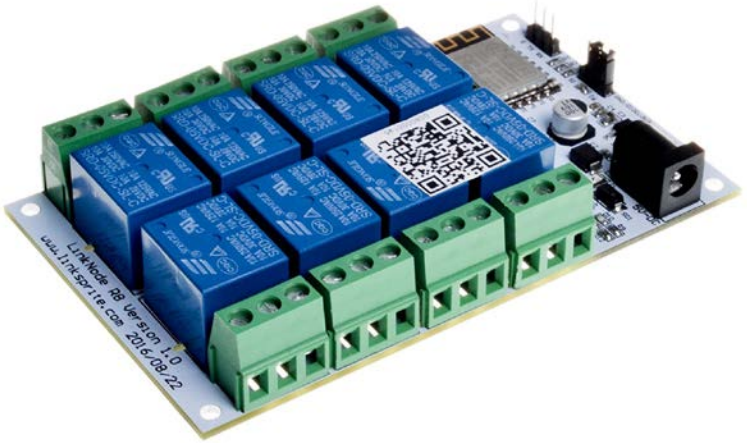

Abbildung 3: Relaisboard von LinkSprite mit einem ESP8266

Espressif hat bereits einen guten Namen in der Hobby- und Makerszene, ist aber im professionellen und industriellen Sektor noch nicht ganz angekommen.

Mit dem ESP32 erhält der Entwickler eine faire Chance im Wettbewerb gegenüber anderen Mikrocontrollern - Maxim, Texas Instruments, Microchip, Nordic Semiconductor, NXP, Silicon Labs, ST und Atmel bieten alle wettbewerbsfähige Bausteine, einige sind bekannter als andere, manche bieten eine kostenlose IDE andere wiederum sehr teure SDKs und die Mehrzahl verfügt über eine Art RTOS, also über ein Echtzeit-Betriebssystem.

Die Entscheidung von Espressif, ihr SDK und gesamte Entwicklungsumgebung öffentlich verfügbar zu machen, war sicher richtig in diesem Kontext und erlaubt anderen Marktteilnehmern wie etwa PlatformIO[11], ihre Framework-Entwicklungen ebenfalls im Open-Source Sektor zu platzieren.

Abbildung 4: Espressif Hackathon am 27.4.2017 in Shanghai

Espressifs Wahl von FreeRTOS als Echtzeit-Betriebssystem erlaubte dem Unternehmen zudem die Veröffentlichung ihres gesamten Frameworks und mit dieser Hilfe kann jeder Interessierte das Entwickeln eingebetteter Systeme mit den Espressxif SoCs lernen, da er alle notwendigen Informationen besitzt, die nur über den Source-Code vermittelt werden können.

[11] PlatformIO siehe: https://platformio.org/

Titel: ESP32 für den Einsatz in Geräten der Industrie 4.0
Autor: Originalartikel von Dror Gluska, übersetzt und bearbeitet von Dipl.Ing. Stefan Tauschek, Macnica Europe GmbH
Datum: 21.3.2018

Neben der leichten Verfügbarkeit von Hard- und Software kann eine Community gebildet werden und damit genießen Hard- und Software, Treiber und Komponenten einen Support, der mit anderen Modellen nur schwer realisierbar ist einfach dadurch, dass der Community alle notwendigen Dokumente und Spezifikationen zur Verfügung stehen. Diese Offenheit zahlt sich aus, weil Freiwillige Zeit und Arbeit investieren, während Espressif sich entspannt zurücklehnen kann und Muße hat, die nächsten Killerchips zu planen.

Um hier ein Beispiel zu geben, wie man es nicht machen sollte, sei auf Intels Curie, Galileo, Joule und Edison verwiesen. Obwohl Intel große Anstrengungen unternahm, diese Chips in die Makerszene zu drücken, beispielsweise indem es viele davon großzügig als Geschenke verteilte oder als Preise bei Wettbewerben und Hackathons auslobte, waren viele davon zu teuer und litten an schlechter Performance. Intels größtes Problem war aber der Mangel an verfügbarer Dokumentation und Unterstützung seitens der Community. Unternehmen, die mit diesen Makerchips ein Geschäftsmodell aufbauen wollten, hängen nun in der Luft und müssen ihre Produktentwicklung neu aufsetzen.

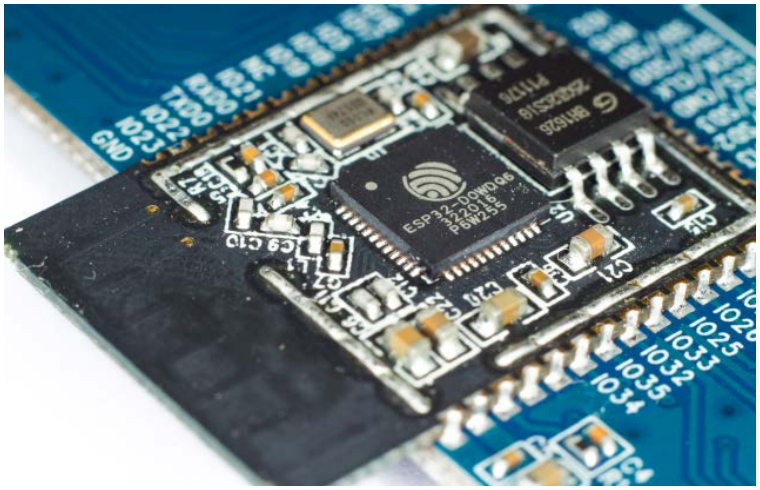

Abbildung 5: Der ESP32, montiert in einem Modul

Das ESP32-Ökosystem

Hier nun mehr Details zum angesprochenen ESP32 und seiner Entwicklungs-/Softwareumgebung:

ESP32 Mikroprocessor – Dual-Core Tensilica Mikroprozessor mit 240Mhz, 520 kB Ram, RTC, ULP, 34 GPIOs, Netzwerk-Connectivity und zahlreiche Schnittstellen wie UARTs, I2Cs, I2Ss, SPIs, CanBus, ADCs und DACs sowie externem SPI EEPROM mit Kapazitäten von 1-16 MByte.

ESP-IDF – Espressif hat hier große Anstrengungen unternommen, ein Software Development Kit (SDK) auf die Beine zu stellen, das die meisten, wenn nicht alle Fähigkeiten der Hardware umsetzt.

Netzwerkstandards - WiFi/BT/Ethernet-Connectivity, lwIP TCP/IP-Stack.

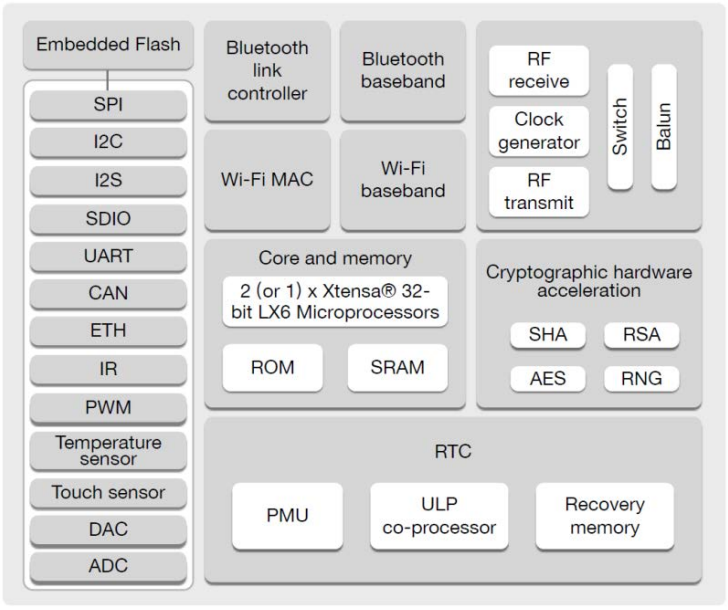

Abbildung 6: Blockdiagramm des ESP32 von Espressif

Sicherheit & Kryptographie – Befehle zur Verschlüsselung (AES-128, 192, 256 und SHA 1,256,384,512), Netzwerkdatenverschlüsselung, FLASH-

Verschlüsselung, Secure Boot und JTAG-Sperrung. Des Weiteren existiert eine mbedtls[12] Bibliothek (bekannt als PolarSSL), eine OpenSSL-API und libsodium[13].

Cloud Ready – getestete Komponenten sowohl für AWS-IoT und Azure-IoT.

FreeRTOS – Vollständige Implementierung einschließlich Tasks und Echtzeit Task Scheduler, Timer, Queues, Interrupt Handling, Critical Sections und Events.

Moderner C++ 11/14 GCC Compiler – C++ 11 und höher bietet viele Vorteile gegenüber C++ 98, u.a. verbesserte Produktivität, Geschwindigkeit, Stabilität, sicherer Code und Portierbarkeit sind hier die Stichworte. Smart Pointer, verbesserte Collections, Lambdas und Threads sind die wesentlichen Vorteile, die jeder nutzen sollte.

Professionelle Qualität, Industriequalität

Durch Kombination all dieser Komponenten hat Espressif sichergestellt, dass der ESP32 nahtlos in viele Anwendungen passt einschließlich solchen mit dem Anspruch des industriellen und professionellen Sektors.

Wie aber unterscheidet sich die Softwareentwicklung im Hobby-/Communitybereich von der im professionellen Umfeld? Hier der Versuch einer Kategorisierung:

Industrieentwickung	**Hobbyentwicklung**
Finanziert durch Unternehmen	Finanziert durch private Konsumenten aus der OpenSource Community
Mehrere Entwickler oder Teams arbeiten an einem gemeinsamen Ziel	Ein oder mehrere Entwickler arbeiten in ihrer Freizeit oder mit sehr geringem Budget
Unternehmensrelevante Systeme, die entscheidend sind für das Wachstum des Geschäftsbereiches	Gelegentlicher Ausfall wird erwartet aber ohne gravierende Auswirkung
Zuverlässige und robuste Applikation als Beweis für den kommerziellen Wert.	Zuverlässigkeit und Robustheit kann nicht garantiert werden
Fehlertolerant, hoch verfügbar und selbsterholend.	Reparatur und Support, wenn überhaupt nur zeitweise verfügbar
Vorhersagbare und wiederholbare Performance und Timing	Performance ist nicht definiert und hat keine negativen Auswirkungen
Abarbeitung in Warteschlangen oder durch Interrupt Handling	Langsame Reaktionszeiten und inkonsistente Ergebnisse sind ggf. akzeptabel
Connectivity wesentlich	Connectivity optional

12 mbedTLS, ehemals PolarSSL, eine TLS/SSL Implementierung, siehe: https://en.wikipedia.org/wiki/Mbed_TLS
13 Siehe: https://www.gitbook.com/book/jedisct1/libsodium/details

Titel: ESP32 für den Einsatz in Geräten der Industrie 4.0
Autor: Originalartikel von Dror Gluska, übersetzt und bearbeitet von Dipl.Ing. Stefan Tauschek, Macnica Europe GmbH
Datum: 21.3.2018

Höhere Kosten	Mittlere Kosten

Es bedeutet also nicht automatisch die höhere Codequalität, wenn Anwendungen im Industrie- oder Unternehmensbereich entwickelt wurden und nicht zwingend geringere Qualität, wenn sie im Hobbybereich entstanden sind - beide Bereiche haben einfach unterschiedliche Anforderungen.

So könnte ein Hobbyentwickler mehr Zeit mit der Gestaltung der Benutzerschnittstelle verbringen während ein kommerzieller Entwickler wahrscheinlich mehr Zeit auf Stabilität und Belastungstests verwendet um sicherstellen, dass die Software auch dann nicht versagt, wenn eine Bohrmaschine mit 30.000 U/min und sehr schnellem Vorschub auf einen Aluminiumblock trifft. Beide Bereiche haben einfach unterschiedliche Anforderungen, das ist alles.

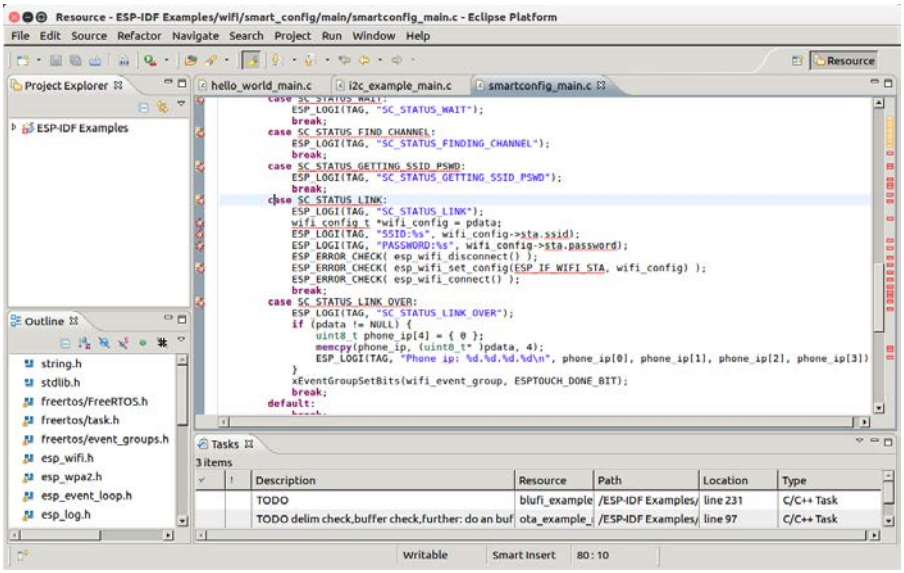

Abbildung 7: Eclipse Editor mit Beispielprogramm aus dem ESP-IDF

Ein Programm, das niemals ausfällt

Man stelle sich ein Programm vor, dass nahezu niemals ausfällt, immer durchgehend 24/7 läuft und jederzeit überwacht werden kann, was es gerade tut und wie der Systemzustand ist und wenn es doch ausfällt, warum es ausfällt. Zudem ist es zu jeder Zeit möglich, Korrekturupdates einzuspielen, wenn ein Problem erkannt wird oder eine bestimmte Funktion verbessert wurde. Das alles bei maximaler Sicherheit.

Ein solches Programm gibt es in Wirklichkeit nicht, aber es kann einiges getan werden, um zumindest die üblichsten Fehler zu vermeiden und die Stabilität zu verbessern:

- **Einsatz moderner Compiler und Programmierparadigmen**, objektorientierte Entwicklung, RAII[14], Smart Pointer, Sammlungen, Lambdas und alles, was noch weiter die Sicherheit erhöht. Standard-Pointer (auch Raw Pointer genannt) gibt es solange wie „C“ aber Smart Pointer sind sicherer in der Anwendung. Vorzugsweise sind dynamische und statische Typumwandlungen (static and dynamic casts) gegenüber uminterpretierten Casts und „C“-Casts zu verwenden.
- **Tests** sind ganz wesentlich für eine stabile Anwendung. Während es recht kompliziert werden kann, Tests für einen Xtensa-Prozessor in QEMU auf einem PC durchzuführen, so kann doch mit

14 RAII = Resource Acquisition Is Initialization, siehe https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization

Hilfe von Abstraktionsebenen und Mocks der größte Teil von entwickeltem Code auf dem Betriebssystem der Entwicklungsmaschine getestet werden. ESP-IDF und PlatformIO bieten dazu Möglichkeiten, um die Sache etwas zu erleichtern. Da der Sourcecode Standard „C"-Code ist, kann zudem cppcheck zur statischen Code-Analyse verwendet werden.

- Stets sollte eine **Versionskontrolle** für den Source-Code gepflegt werden, auch wenn es sich nur um „Proof-of-Concept" (POCs) Anwendungen handelt.
- **Crash Dump Analysis / Stack Overflow Detection** sind Teil der ESP-IDF und die Anwendung wird dringend empfohlen.
- **Logs** sind wichtig, um Fehlerursachen oder Fehlverhalten aufzudecken und die ESP-IDF bietet Logging-Mechanismen zum Tracen des Programmablaufs. Es ist auch sehr hilfreich, diese Meldungen als Dateien speichern zu können, um sie „post mortem" zu analysieren oder einen Netzwerk-Syslog aufsetzen zu können.
- Verwendung von **OTA-Updates**, Versionierung, signierte Zertifikate und eine „Inszenierungs-umgebung" (Staging Environment), die jene Hardware vollständig abbildet, auf der die Firmware betrieben werden soll.

Was können wir also tun, um Software zuverlässiger zu machen, robust und sicher?

- **Validierung der Eingabedaten** – Eingabedaten sind stets zu verifizieren, der „Buffer Overrun"[15] ist immer noch eine häufig verwendete Hackermethode und auch SQL-Injection ist noch lange nicht vergessen.
- **Tests** – sind möglichst unter erschwerten Bedingungen durchzuführen, mit ungültigen Eingabe-daten, langen WiFi SSIDs, externen Hardware-Glitches und wiederholtem Unterbrechen der Stromversorgung. „Worst Case" Fehler können zum Software-Crash führen, dürfen aber nicht die Sicherheit beeinträchtigen.
- Verwendung moderner **Programmierparadigmen** wie Warteschlangen, Ereignisse, Callbacks, Ausnahmen, Mikrokomponenten, Layer, Steuerungsumkehr[16]
- **Logs** sollten alle Informationen beinhalten die notwendig sind, um ein Problem durch Fernzugriff zu diagnostizieren.

Welche Maßnahmen sind möglich, um Software fehlertolerant, hoch verfügbar und selbsterholend zu gestalten?

- **Fail Fast** – möglichst rasches Auftreten von Fehlern ist zu provozieren und bietet den Vorteil, das sie während der Entwicklungsphase bereits gelöst werden können. Fehler dürfen keinesfalls einfach hingenommen werden.
- **Exception Handling** – eine Exception tritt in einem Ausnahmezustand auf, der nicht eintreten sollte. Wenn doch, ist die Ursache zwingend herauszufinden. Grundsätzlich sollten Exceptions nicht für eine Standard-Fehlerbehandlung verwendet werden.

---

15 Buffer Overrun exploit, siehe: https://en.wikipedia.org/wiki/Buffer_overflow
16 Steuerungsumkehr = Inversion of Control, siehe: https://en.wikipedia.org/wiki/Inversion_of_control

- **Reboot bei Fehler** – die ESP-IDF ist konfiguriert zum „Reboot on Panic“, d.h. ungültige Pointerzugriffe genau wie Stack-Overflows führen zu einem Reboot.
- **Watchdog** – manchmal endet ein Fehler nicht in einem Crash sondern in einer extrem verzögerten Reaktionszeit. Manchmal gerät eine Task auch in eine Endlosschleife und verbraucht die gesamten Prozessorressourcen, um gar nichts zu tun. Watchdogs und Heartbeats wurden zur Erkennung genau solcher Situationen geschaffen und sollten daher auch verwendet werden.
- **Erkennung hoher CPU-Last** – ist zwar nicht ursächlich dem FreeRTOS zuzuordnen sondern schlecht performender Software, sollte aber überwacht werden.
- **Brownout-Erkennung** – Brownout ist ein Zustand mangelnder elektrischer Versorgung (Versorgungsspannung zu gering), um die Hardware spezifikationsgemäß zu betreiben. Dieser Zustand sollte zu einem Reboot der Hardware führen.

Was lässt sich tun, um Software vorhersagbarer, performanter und im Zeitablauf präziser zu gestalten?

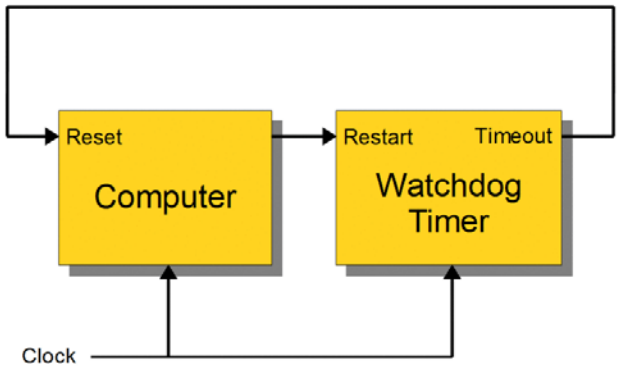

Abbildung 8: Watchdogs überwachen Mikroprozessoren

- **RTOS** – Echzeit-Betriebssysteme bieten sichere Multi-Tasking-Eigenschaften, Warteschlangen und Interrupt-Bearbeitung.
- **Multicore und schnellerer Mikrocontroller** – wenn der Prozessorkern die meiste Zeit beschäftigt ist, kann er nicht rechtzeitig auf Stimuli reagieren, was im besten Fall zu einer reduzierten Leistung führt, im schlimmsten Fall aber eine angeschlossene Hardware zerstören kann.
- **Verwendung von Interrupts und Timern**, die alte Geschichte von Pushing anstelle von Polling.

Eine technische Analyse

Das ESP-IDF ist ein sehr interessantes SDK und Espressif unternahm große Anstrengungen, um die Anwendung so einfach wie möglich zu machen. Innerhalb des SDKs gibt es APIs zur Steuerung der Hardware, Driver, pthread[17]-kompatible APIs und C++ stdlib, sogar glob[18], regex, tar und miniz werden unterstützt. All dies macht die Programmierung eines ESP32-Systems relativ einfach und auch die Portierung existierender Software wird damit erheblich beschleunigt.

17 Pthreads = Posix Threads, siehe: https://en.wikipedia.org/wiki/POSIX_Threads
18 Siehe https://en.wikipedia.org/wiki/Glob_(programming)

Das ESP-IDF hat auch ein paar hilfreiche Komponenten, die bei der Entwicklung robuster Anwendungen helfen, darunter ein SPI-Dateisystem mit Unterstützung sowohl von FAT [19] als auch SPIFFS[20]. Mit dem Dateisystem lassen sich Log- und Datenfiles speichern wobei das SPI-EEPROM einen Wear-Leveling-Layer beinhaltet, der dabei hilft, die Anzahl möglicher Schreiboperationen zu vergrößern. Parameter lassen sich dabei separat im NVS-Speicher sichern, wodurch sie durch Formatierung des Dateisystems nicht beeinträchtigt werden. Und sollte die Größe des SPI FLASH Speichers nicht reichen, kann auch die SD-Card Bibliothek zur Einbindung externen Speichers genutzt werden.

Abbildung 9: Espressif SoCs werden von gut dokumentierten SDKs unterstützt

Cloud Connectivity ist wesentlicher Bestandteil beim Monitoring und der Steuerung großer Anlagen. ESP-IDF beinhaltet die AWS IoT [21] sowie Microsoft Azure IoT [22]-Connectivity, eine TelegramBot-API und die essentiellen IoT-Protokolle CoAP und MQTT.

lwIP ist ein IPv4/IPv6 TCP/IP-Protokollstapel mit einigen interessanten Features wie etwa DNS/mDNS Namensauflösung, SNMP, DHCP, AUTOIP, PPP und L2TP, die Basis für Netzwerktechnik quasi. Damit lassen sich Komponenten entwickeln wie etwa DHCP-Server, SNTP, PING, HTTP(s)-Client/Server, CoAp-Client/Server und OpenSSL-Client/Server.

nghttp2 ist ein HTTP/2 Client/Server wie er beispielsweise bei Alexa Voice Services (AVS) eingesetzt wird.

mbedTLS/libsodium sind beides ähnliche kryptographische Bibliotheken und haben ihre jeweiligen Vor- und Nachteile.

FreeRTOS - APIs beinhalten das Task Management, einschließlich Scheduler und Monitoring sowie Datenstrukturen wie Queues/Stacks, Ringbuffer, Locking-Mechanismen wie Semaphoren und Mutexe.

Das FreeRTOS ist kostenlos, kann aber nicht alle Wünsche erfüllen. Für manche Projekte sind Funktions- oder Wartungsgarantien erforderlich und technischer Support. Dafür gibt es die kommerzielle Alternative OpenRTOS[23], das exakt gleiche RTOS aber mit einem anderen Lizenzmodell. Eine weitere Alternative ist SafeRTOS[24] für sicherheitskritische Anwendungen.

Power Management ist eine wichtige Steuerungsmöglichkeit für besonders sparsame Anwendungen. Der ESP32 hat mehrere Stomsparmodi und einen Ultra-Low-Power (ULP) Prozessor zur weiteren Reduzierung des Stromverbrauchs.

---

19 FAT = File Allocation Table, siehe: https://en.wikipedia.org/wiki/File_Allocation_Table
20 SPIFFS = Serial Peripheral Interface Flash File System
21 AWS IoT = Amazon Web Services IoT, siehe: https://aws.amazon.com/de/iot-core/
22 Azure IoT = Microsoft Azure IoT Suite, siehe: https://azure.microsoft.com/de-de/suites/iot-suite/
23 OpenRTOS siehe: https://www.highintegritysystems.com/openrtos/
24 SafeRTOS, siehe: https://www.highintegritysystems.com/safertos/

Debugging – Espressif hat zum Debugging [25] OpenOCD [26] (Open On-Chip-Debugger) zusammen mit Unterstützung von JTAG implementiert und das ESP-IDF bietet eine Unit-Testapplikation basierend auf Unity. Auch Nutzer von PlatformIO können diese Debugging-Verfahren verwenden.

RTOS

Auch andere Echtzeit-Betriebssysteme (RTOS) sind sicher nicht schlecht, man sollte aber bedenken, dass bei der Einarbeitung in ein SDK mit all seinen Spezialitäten und Limitierungen die Lernkurve sehr steil ist und es viele Stunden dauern kann, bis man ein erstes, einfaches Programmbeispiel am Laufen hat. Der Wechsel zu einem anderen RTOS kann daher großen Aufwand bedeuten nur um später festzustellen, dass dort die gleichen Strukturen und APIs verwendet werden wie im ESP-IDF.

Abbildung 10: FreeRTOS Betriebssystem

Aber warum sollte man ein populäres RTOS einsetzen?

Zum einen beschleunigt dies ganz erheblich die Einarbeitung in ein komplexes SoC mit seiner technischen Dokumentation von mehreren Hundert Seiten. Auf der anderen Seite beschleunigt es aber auch die Entwicklung eigener Anwendungen und erhöht die Designsicherheit, weil vielfach ausgetestete Bibliotheken zum Einsatz kommen. Dies eben ist einer der großen Vorteile als Mitglied der inzwischen doch recht großen Espressif Entwicklergemeinde: Viele Entwickler haben sich schon in die technische Dokumentation eingelesen, Treiber und Beispielcode entwickelt, Probleme identifiziert, gelöst und in der Community veröffentlicht. Wenn auch nicht alle Beiträge von bester Qualität sind so gibt es doch meist zu jeder Aufgabenstellung wertvolle Hinweise und Lösungsansätze.

Der ESP32 kann zusätzlichen mit 2 alternativen Echtzeit-Betriebssystemen genutzt werden:

Simba http://simba-os.readthedocs.io und

NuttX http://www.nuttx.org/

Lizensierungsmodelle

Das ESP32-SDK ist unter Apache[27] lizensiert und gibt von daher Entwarnung, allerdings ist bei Entwicklung kommerziell verwendeter Implementierungen bei jeder Komponente das Lizensierungsmodell unbedingt zu prüfen. FreeRTOS hat ein anderes Lizensierungsmodell, einige LCD-Treiber haben wiederum andere etc. Der beste Rat an dieser Stelle ist es, einen spezialisierten Rechtsanwalt zu konsultieren, bevor eine kommerzielle Anwendung auf den Markt gebracht wird.

25 Siehe: https://dl.espressif.com/doc/esp-idf/latest/api-guides/jtag-debugging/index.html
26 OpenOCD, siehe: http://www.openocd.net/
27 https://en.wikipedia.org/wiki/Apache_License

Titel: ESP32 für den Einsatz in Geräten der Industrie 4.0
Autor: Originalartikel von Dror Gluska, übersetzt und bearbeitet von Dipl.Ing. Stefan Tauschek, Macnica Europe GmbH
Datum: 21.3.2018

Kurzer Weg zum Prototyp oder Rapid Prototyping

SDKs bieten eine hervorragende Möglichkeit, die Eigenschaften eines Prozessors zu verstehen und obwohl C++ eine einfach erlernbare Sprache ist gibt es mitunter vor allem bei POC Entwicklern mit unterschiedlichem Erfahrungshintergrund wie Physik oder Biologie die Notwendigkeit, die Möglichkeiten des ESP32 in einer anderen Sprache zu beschreiben. Hier die Alternativen:

- **LUA** – Das Lua[28] RTOS ist ein Echtzeitbetriebssystem speziell für eingebettete Systeme mit minimalen Anforderungen an FLASH- und RAM-Speicher.
- **MicroPython[29]** - ist eine Implementierung von Python 3.x auf Mikrocontroller und kleine, eingebettete Systeme.
- **Basic** – Ein Basic Interpreter ist fest (aber undokumentiert) im ESP Silizium integriert.

Einige interessante Projekte

Web-Radio - MP3 Webradio Project: https://github.com/MrBuddyCasino/ESP32_MP3_Decoder

Kamera-Demo – Mit OV7725 Kameramodul: https://github.com/igrr/esp32-cam-demo

Websocket Beispielprojekt: - https://github.com/ThomasBarth/WebSockets-on-the-ESP32

WiFi-Manager - WiFi Connection-Manager mit Fallback-Konfigurationsportal als Webserver - https://github.com/zhouhan0126/WIFIMANAGER-ESP32

CAN-Bus Driver https://github.com/ThomasBarth/ESP32-CAN-Driver

ESP32 QEMU - https://github.com/Ebiroll/qemu_esp32/

Ethernet Connection mit dem LAN8720 - https://sautter.com/blog/ethernet-on-esp32-using-lan8720/

st

28 Siehe: https://github.com/whitecatboard/Lua-RTOS-ESP32
29 Siehe: https://github.com/micropython/micropython-esp32

Titel: ESP32 für den Einsatz in Geräten der Industrie 4.0
Autor: Originalartikel von Dror Gluska, übersetzt und bearbeitet von Dipl.Ing. Stefan Tauschek, Macnica Europe GmbH
Datum: 21.3.2018

Über die Autoren

Dror Gluska ist ein erfahrener Softwareentwickler mit praktischen Fähigkeiten als Softwarearchitekt und ein „Maker“, lange bevor der Begriff „Maker“ überhaupt populär wurde.

Er ist stets auf der Suche nach Dingen, die es zu entwickeln und verbessern gibt, sieht sich aber auch als Mentor und Unterstützer für junge Softwareentwickler. Sein Leitsatz ist: „Wenn es nicht kaputt ist, mach‘ es kaputt und finde so heraus, wie es funktioniert".

Blog: http://drorgluska.com

Stefan Tauschek studierte Nachrichtentechnik an der Fachhochschule München und arbeitete nach seinem Abschluss über mehrere Jahre in der Entwicklung von Multimediakomponenten, Videoverarbeitung und Streaming-Media-Verfahren.

Heute ist er Applikationsingenieur und Technologieberater bei der Macnica GmbH und unterstützt Industriekunden bei der Realisierung von Projekten aus dem Bereich Bildverarbeitung, Netzwerke und Industrieautomatisierung.

E-Mail: stefan.tauschek@macnica.com

Titel: ESP32 für den Einsatz in Geräten der Industrie 4.0
Autor: Originalartikel von Dror Gluska, übersetzt und bearbeitet von Dipl.Ing. Stefan Tauschek, Macnica Europe GmbH
Datum: 21.3.2018

Inhaltsverzeichnis

Abbildungsverzeichnis