HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Konstantin Lorenz

Programmierung eines embedded Devices für automatisierte WLAN Tests

2017

Fakultät **Angewandte Computer- und Biowissenschaften**

BACHELORARBEIT

Programmierung eines embedded Devices für automatisierte WLAN Tests

Autor:
Konstantin Lorenz

Studiengang:
Angewandte Informatik - IT-Sicherheit

Seminargruppe:
IF13WI-B

Erstprüfer:
Prof. Dr. rer. nat. Christian Hummert

Zweitprüfer:
Prof. Dr. rer. pol. Dirk Pawlaszczyk

Mittweida, 2017

I. Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die heutige moderne Gesellschaft nutzt das Internet als Kommunikationsmittel. Das Internet ist das wichtigste Kommunikationsmittel der Welt. 92% der deutschen Haushalte besitzen einen Internetanschluss. [Eur17] Davon gehen 39% über das WLAN ins Internet. [Ver16]

Das Internet bietet den Benutzern breite Anwendungsmöglichkeiten vom online-Banking über die Kommunikation via E-Mails, Foren, Blogs und Soziale Netzwerke. Außerdem wird es zunehmend für den Abschluss von Verträgen und Rechtsgeschäften verwendet. Die steigende Verwendung im privaten und geschäftlichen Bereich bieten aber auch Cyberkriminellen immer mehr Möglichkeiten, wodurch die Straftaten sich ins Internet verlagern. Die Bandbreite von illegalen Aktivitäten im Internet reichen von Verbreitung verbotener Inhalte über das Verkaufen von Waffen, Drogen und illegal erlangter persönlicher Daten, sowie Identitätsdiebstahl und Erpressung bis hin zu DDoS-Attacken, Hackerangriffen und Wirtschaftsspionage. [Bun17]

Die Sicherheitsbehörden werden bei wachsender Internetkriminalität/Cyberkriminalität mit wachsender Anzahl an Tatverdächtigen konfrontiert. Um die Aufgaben der Beweismittelsicherung im Bereich des WLANs der Tatverdächtigen effizienter zu gestalten und für die Sicherheitsbehörden zu vereinfachen sind alternative Konzepte gegenüber der klassischen Methode mittels Laptop und Richtantenne im WLAN notwendig.

Auch im Bereich der Netzwerkadministration und Fehlersuche im WLAN ist es sinnvoll, eine Alternative zur klassischen Methode der Netzwerkanalyse mittels Laptop mit Richtantenne zu verwenden. Es ist nicht immer möglich, einen Rechner in unzugänglichen Bereichen über einen längeren Zeitraum zu positionieren und Personal für diese Aufgabe zu Verfügung zu stellen, um dort Netzwerkanalysen vorzunehmen.

1.2 Zielsetzung

Im Rahmen dieser Bachelorarbeit soll ein Prototyp eines eingebettetes Systems zum Mitschneiden von WLAN-Daten eines bestimmten Kanals entwickelt werden. Dabei soll ein besonderer Augenmerk auf die Eigenschaften wie Energieeffizienz, möglichst geringer Größe, geringe Kosten, sowie Automatisierung des Mitschneideprozesses gelegt werden, d.h. das Gerät soll autark arbeiten und die Daten auf ein wechselbares Speichermedium ablegen können.

1.3 Grundlagen des WLAN

Um ein allgemeines Verständnis für die Kommunikation im Wireless Local Area Network (WLAN), womit eigentlich der IEEE-802.11 gemeint ist, zu bekommen, werden hier kurz die Grundlagen erklärt.

Der IEEE-802.11 gehört zu der Familie des IEEE-802 und spezifiziert die Local Area Network (LAN) Technologien. Die Abbildung 1.1 zeigt den Platz der IEEE-802 Komponenten im OSI Schichtensystem. Der IEEE-802.11 umfasst die zwei untersten Schichten des OSI-Models. Es werden sowohl Komponenten der Bitübertragungsschicht (PHY) als auch der Sicherungsschicht sowie deren Unterschichten, die Media Access Control (MAC) Schicht und die Logical Link Control (LLC) Schicht von 802.11 verwendet. Die MAC Schicht besteht aus Regeln wie man Daten senden kann, und wie der Zugriff ins WLAN erfolgt. Die LLC ist im IEEE-802.2 beschrieben und realisiert den Datenaustausch zwischen den verschiedenen Medien. Die PHY ist für Details der Übertragung und den Empfang zuständig. Am Anfang gab es im Standard nur zwei physikalische Übertragungsarten die FHSS und den DSSS, die aber durch 802.11a, b, und g erweitert wurden. Der ESP8266 unterstützt alle diese physischen Übertragungsarten auch im promiscuous Mode. [Gas13, S. 14]

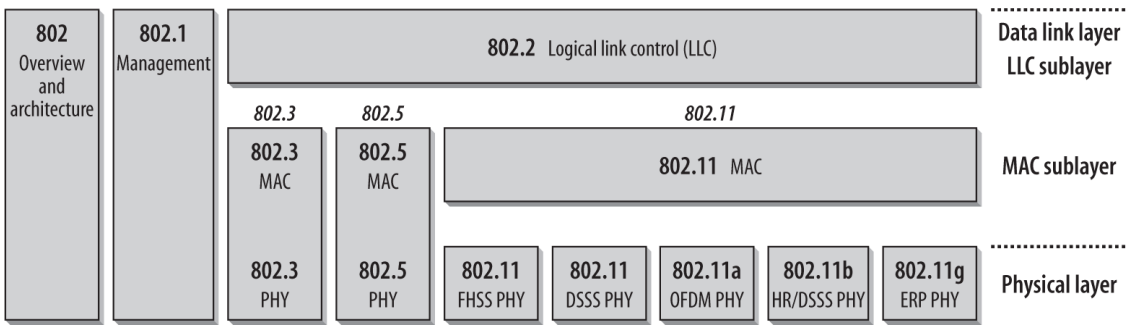


Abbildung 1.1: OSI Modell der IEEE-802 Familie [Gas13, S. 14]

Im IEEE-802.11 werden auch verschiedene Topologien beschrieben. Diese werden allgemein als Basic Service Set (BSS) bezeichnet. Es handelt sich dabei um eine Menge von Netzwerkteilnehmern, die untereinander über eine bestimmte Methode kommunizieren wollen. Die häufigste Methode, die verwendet wird, ist die Infrastruktur BSS. Dazu wird ein Access Point benötigt. Werden jetzt Daten von einer Station zu einer anderen übertragen, müssen diese Daten erst von Access Point entgegengenommen werden. Erst von dort werden die Daten zum Empfänger weitergesendet. Das hat den Vorteil, dass man den Netzwerkadapter im promiscuous Mode nur in die Reichweite vom Access Point platzieren muss, um den Netzwerkverkehr zwischen den Stationen abzufangen. Wird der Access Point, wie in den meisten Fällen, als Bridge für die Umsetzung vom WLAN ins LAN benutzt, muss eine Protokollübersetzung stattfinden.

1.4 Die UART Schnittstelle

Der ESP-01 kommuniziert über die UART Schnittstelle nach dem RS-232 Standard auf der Bitübertragungsschicht. In diesem Standard ist festgelegt, wie einen Verbindung zwischen zwei Geräten erfolgen soll. Die Verbindungen werden asynchron aufgebaut, das bedeutet, dass Sender und Empfänger nicht aushandeln wann die Daten übertragen werden. Die Datenübertragung ist nicht an ein definierte Taktsignal gebunden. Es handelt sich beim ESP8266 ebenfalls um eine serielle Datenübertragung, d.h. es werden die Daten Bit für Bit übertragen, im Gegensatz zur parallelen Datenübertragung. Die Übertragung an sich ist binär kodiert. Ein Zeichen wird in 7 Bit kodiert. Am Anfang des Zeichens steht noch ein Startbit und am Ende ein Stopbit. Man kommt somit bei der Übertragung eines Zeichens auf 9 Bit, die übertragen werden müssen. [MBW10, S. 45]

Die Pegel befinden sich laut dem Standard zwischen -3V und -25V und 3V und 25V. Der ESP8266 unterstützt allerdings nur Pegel bis 3,6V. Deshalb muss für die Kommunikation mit dem Rechner ein so genannter USB-to-TTL Konverter verwendet werden, der die Spannungen wandelt. [MBW10, S. 45]

Um eine Datenübertragung zu realisieren, werden für die asynchrone, serielle Variante die Datenleitungen RX und TX sowie die Spannung und die Masse benötigt. Die TX-Datenleitung des Senders wird mit der RX-Datenleitung des Empfängers verbunden. Ebenfalls sollten die jeweiligen Spannungen und Massen miteinander verbunden werden, um Spannungsunterschiede auszugleichen. [MBW10, S. 48]

1.5 Die SPI-Schnittstelle

Die Serial Peripheral Interface (SPI) Schnittstelle wird für die Ansteuerung des Speicherkartenmodul benötigt. Die Grundlagen werden hier erklärt.

SPI beruht auf dem Master-Slave-Prinzip. Das Gerät, welches den Übertragungstakt und den Moment der Übertragung festlegt, wird als Master bezeichnet. Der Master legt auch die Übertragungsgeschwindigkeit fest. Als Slaves werden die passiven Geräte bezeichnet, die auf die Anforderungen des Masters reagieren. SPI ist ein Bussystem. Somit können weitere Geräte als Slaves eingebunden werden. Um zu unterscheiden, welcher der Slaves senden darf, wird die CS-Leitung des aktiven Slaves auf $High$ gesetzt. [MS05, S. 49]

In Abbildung 1.2 sind vier Schieberegister dargestellt. Alle arbeiten mit dem selben Schiebetakt, der vom Master vorgegeben wird. Der Sende- und Empfangsprozess erfolgt gleichzeitig. Dafür gibt es zwei Datenleitungen [MS05, S. 49]:

- MISO - (Master In Slave Out)
- MOSI - (Master Out Slave In)

Als erstes schreibt der Master ein Byte in das TX-Master-Register. Es werden acht Takte auf die $SCLK$-Leitung gelegt und die bi-direktionale Übertragung beginnt. Es wird ein Byte vom TX-Register des aktiven Slaves gelesen, egal welche Daten dieses enthält und in das RX-Register geschrieben. [MS05, S. 50]

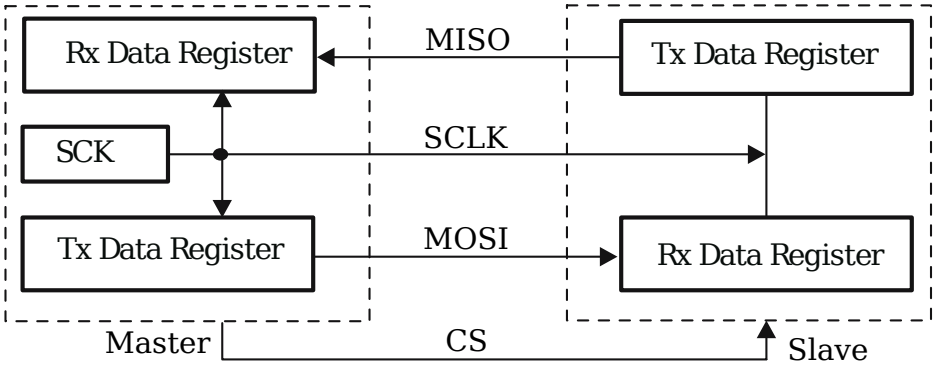
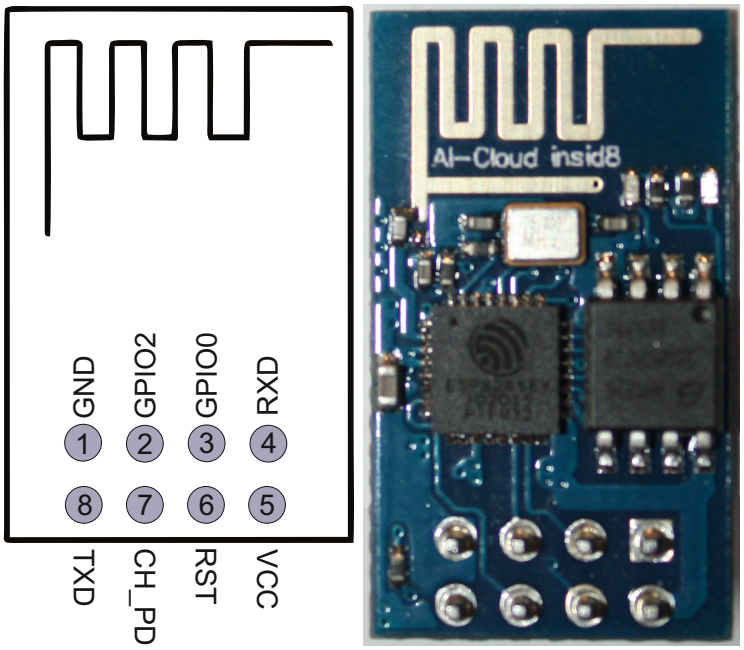


Abbildung 1.2: Block Diagramm SPI [MS05, S. 49]

1.6 ESP-01

Um die WLAN-Daten zu empfangen, wird ein leistungsfähiges WLAN-Modul benötigt, welches einen geringen Stromverbrauch aufweist und besonders klein und kostengünstig ist, und sich in einen Modus für das Mitschneiden von WLAN-Daten versetzen lässt. Dazu ist der ESP-O1 von Ai-Thinker gut geeignet. Der ESP-01 ist eine Platine, die von Ai-Thinker entwickelt wurde. Verbaut ist ein ESP8266 System on Chip (SoC) von Expressif. Die Benutzer-Firmware befindet sich auf einem 8Mbit Flash, der über SPI an den ESP8266 angebunden ist. [AI 15, S. 3] Es ist durchaus möglich, auch größere Flash-Speicher bis zu 16MB zu verbauen. [AI 15, S. 12] Damit bietet der ESP-01 genügend Platz für Programmcode als auch eventuelle Benutzerparametern. Der SPI-Flash eignet sich jedoch nicht zum durchgängigen Speichern von WLAN-Daten, da der Flash nur begrenzte Schreibzyklen zulässt. Schreibt man zu oft in den SPI-Flash, kommt es irgendwann zu Schreibfehlern. Auch wären 16MB für WLAN-Daten einfach zu wenig Speicherkapazität. Die Antenne ist eine 3dbi Printed Circuits Board (PCB) Antenne und in der Abbildung 1.3 zu sehen. [AI 15, S. 10] Die Antenne bietet eine geringe Reichweite, aber für des Mitschneiden von Daten ist ebenso die Empfindlichkeit interessant, die bei bis zu -98dbm [AI 15, S. 9] liegt und damit für ein eingebettetes System sehr hoch ist. Nach außen ist nur die Universal Asynchronous Receiver Transmitter (UART) Schnittstelle geleitet. Diese dient dazu Befehle und Daten zu übertragen. Die General Purpose Input/Output (GPIO) Pins dienen der Steuerung des Modus, um z.B. ein Flashen zu ermöglichen.



(a) Skizze (eigene Graphik) (b) Bild (eigene Graphik)

Abbildung 1.3: Pin-Belegung ESP-01 nach [AI 15, S. 7]

Nummer	Pin-Name	Funktion
1	GND	Masse
2	GPIO2	GPIO, interner Pull-up
3	GPIO0	GPIO, interner Pull-up
4	RXD	UART0, Datenpin zum senden
5	VCC	3,3V Spannungsversorgung
6	RST	externer Reset Pin
7	CH_PD	Pin zum aktivieren des Chip
8	TXD	UART0, Datenpin zum empfangen

Tabelle 1.1: Pin-Beschreibung [AI 15, S.8]

1.6.1 ESP8266

Der ESP8266 ist ein SoC bei dessen Herstellung viel Wert auf Energieeffizienz und hohe Platzersparnis gelegt wird. Es kann sowohl in vorhandene eingebetteten Lösungen integriert werden als auch als allein operierende Anwendung fungieren z.B. als Slave einer Mikro Controller Unit (MCU) [Esp17a, S.1]. [AI 15, S.3]

Die integrierte CPU ist ein Tensilica L106 32-bit MCU mit einen 16-Bit Reduced Instruction Set Computer (RISC). Die CPU Geschwindigkeit liegt bei 80 bis maximal 160 Mhz. [Esp17a, S.7]

1.6.2 Promiscuous Mode

Bei dem promiscuous Mode wird der Netzwerkadapter in einen Modus geschalten, der es erlaubt, dass nicht nur die an den Netzwerkadapter adressierten Pakete entgegengenommen werden, sondern alle Pakete, die diesen Netzwerkadapter erreichen. Im Standardmodus werden alle Pakete, die nicht an die eigene MAC adressiert sind, verworfen. Im Gegensatz zum Monitor Mode empfängt ein Gerät im promiscuous Mode keine Pakete der MAC-Schicht, wie z.B. ACK, RTS und CTS Nachrichten. Der Monitor Mode empfängt ebenfalls Pakete die Prüfsummenfehler enthalten oder auf der Bitübertragungsschicht übertragen wurde und normalerweise verworfen wurden. [Chr05, S. 550]

1.6.3 Non-OS SDK

Das SDK gibt es in verschiedenen Versionen. In dieser Arbeit wird die Non-OS Version 2.0.0 verwendet. Dieses Software Development Kit basiert auf der Sprache C und kann von der Espressif-Seite [1] heruntergeladen und in die virtuelle Maschine kopiert werden.

1.6.3.1 Der Aufbau im Allgemeinen

Wie in Abbildung 1.4 zu sehen ist, wird Beispielcode von Espressif im Ordner *examples* bereitgestellt. In dem Ordner *app* befinden sich das Arbeitsverzeichnis, welches den Quellcode und die Header-Dateien für die Compilierung beinhalteten. Die Binärdateien befinden sich im Ordner *bin*. Weitere Informationen bezüglich des SDKs findet man im Ordner *documents*. Die Treiber für die Schnittstellen sind in dem Ordner *driver_lib* untergebracht. Im Ordner *includes* befinden sich weiter Header-Dateien und relevante API-Funktionen, die vom Nutzer nicht modifiziert werden sollten. Link-Dateien und Scripte findet man im *ld* Ordner, diese sollten auch nicht modifiziert werden. Die weiteren Ordner *lib* und *tools* werden ebenfalls vom Makefile benutzt, sind aber für das Programmieren einer Anwendung nicht weiter interessant. [Esp17c, S. 9]

1.6.3.2 Aufbau der Benutzeranwendung

Der Ordner *App* wird ebenfalls nochmals untergliedert, wie in Abbildung 1.5 dargestellt. Die Hauptordner die bei jeder Anwendung existieren müssen, sind *include* und *user*. In dem *include* Ordner befinden sich alle Header-Dateien des Projektes. Die C-Quelldateien existieren hingegen alle im *user* Ordner. Alle Header der C-Quelldateien müssen in die *usermain.c* eingetragen werden. Diese Datei beinhaltet auch die Main-

[1] `https://espressif.com/en/support/download/sdks-demos`

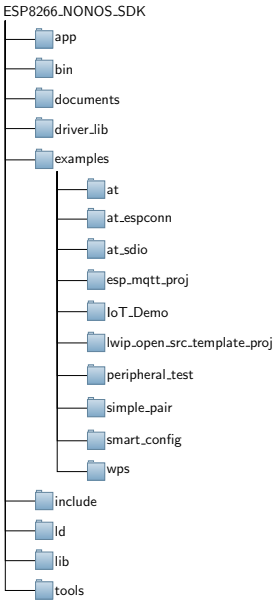


Abbildung 1.4: Non-OS SDK Struktur

Methode. Das Script *gen_misc.sh* ist für die korrekte Parameterwahl der Compilierung bestimmt und erstellt zusammen mit den Makefile die Binärdateien.

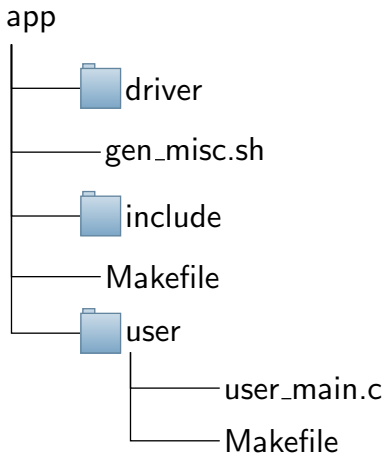


Abbildung 1.5: Struktur der Anwendung (eigene Grafik)

1.7 Nucleo-STM32F411RE

Das STM32 Nucleo-64 Board von ST ist für das Ausprobieren und Entwickeln neuer Konzepte und Prototypen auf Basis der STM32 Mikrocontroller hergestellt worden. In unserem Fall dient er als Schnittstelle zwischen dem ESP-01 und einer Platine für das Speichern der WLAN-Daten. Es ist ein ST-LINK-V2-1 Debugger und Programmer

über USB integriert, der das Programmieren mit der umfassenden Hardware Abstraction Layer (HAL) Bibliothek vereinfacht. Durch diesen HAL ist es möglich, relativ unabhängig von der Hardware zu entwickeln. Da man im Fall eines Hardwaretausches nur darauf achten muss, dass die Funktionen für die Hardware existieren.

Das Board ist mit Arduino Uno V3 und der ST Morpho Schnittstellen und Header ausgestattet. Die ST Morpho Schnittstellen sind die hauseigenen Schnittstellen von ST und ermöglichen einen Standardanschluss für Erweiterungsplatinen. Durch die Integration von Arduino Schnittstellen ist eine Entwicklung mit Erweiterungsboards für Arduino möglich. Es gibt sehr viele Arduino Erweiterungen auf dem Markt und in den meisten Fällen lassen sich die dazugehörigen Softwarebibliotheken einfach importieren. Bei der Online-Integrated Development Environment (IDE) findet man einen eigenen Import-Wizard, was die Programmierung erheblich vereinfacht. Für die offline-IDE Eclipse ist ein wenig mehr Aufwand erforderlich. Es ist notwendig, die Softwarebibliotheken herunterzuladen und ebenfalls über die Importfunktion zu integrieren. In dieser Arbeit wird Eclipse mit dem GNU Compiler Collection (GCC) und dem dazugehörigen Plug-In verwendet. [STM16a, S. 1]

Das STM32 Nucleo Board ist um den STM32 Mikrocontroller herum aufgebaut. In Abbildung 1.6 ist zu sehen, dass das Board aus einem ST-Link Teil besteht, der über USB an den Rechner angeschlossen werden kann, um die Debugger-Funktionalität zu gewährleisten. Wird der Nucleo über USB angeschlossen, erscheint ebenfalls ein Wechseldatenträger. Über diesen erfolgt das Flashen mit Binärdateien. Dazu muss einfach die Binär-Datei auf den Wechseldatenträger kopiert werden.

Sollte der ST-Link Teil nicht benötigt werden, kann er abgeschnitten werden, um die Boardgröße zu verringern. Der obere Teil ist über Serial Wire Debug (SWD) mit dem unteren MCU Teil verbunden. SWD ist der Nachfolger von JTAG als Debug-Schnittstelle und bietet viele Vorteile z.B. schnellere Übertragungsraten und Echtzeitzugriff auf den Systemspeicher. Im 2. Teil ist die MCU mit den Anschlüssen für die Arduino- und Morphoerweiterungen sowie den User-Button und den Reset-Button und den Light-Emitting Diode (LED). [STM16a, S.16]

Der Nucleo besitzt verschiedene Konnektoren, wie in Abbildung 1.7 zu sehen ist. Die Konnektoren CN1, CN2 und CN4 befinden sich auf dem STLink Teil siehe Abschnitt 1.7.0.6. Zusätzlich ist noch eine LED (LD1) verbaut, um die Verbindung zum COM-Port anzuzeigen.

Auf dem MCU Teil befinden sich die Konnektoren CN7 und CN10 für die Morpho-Schnittstelle und die Konnektoren CN5, CN6, CN8 und CN9 für die Arduino-Schnittstelle. Beide Schnittstellen werden für den Anschluss von zusätzlichen Komponenten, wie das W5500 für das Speicherkartenmodul benötigt. Des Weiteren sind verschiedene LEDs (LD2-LD3) sowie Buttons (B1-B2) verbaut, die zu Benutzersteuerung dienen sollen. Es

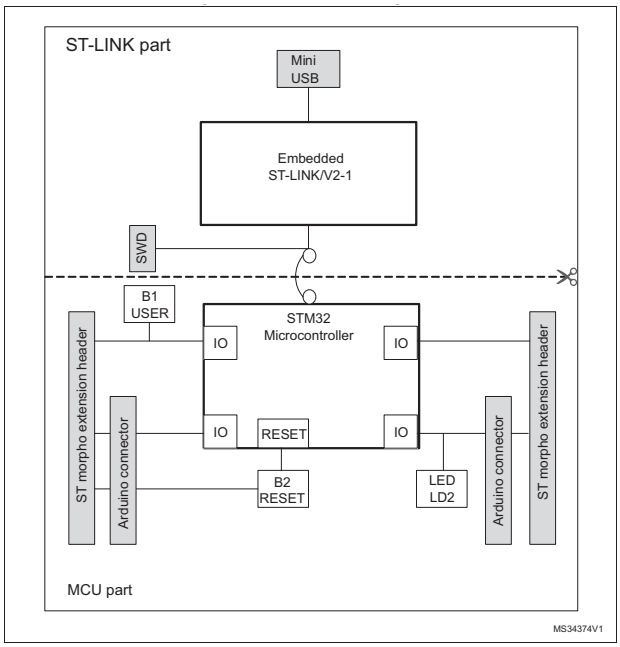


Abbildung 1.6: Hardware Block Diagramm von [STM16a, S.12]

sind LEDs für die USB Kommunikation (LD1), sowie für den Benutzer (LD2) und die Stromversorgung (LD3) integriert, wie in Abbildung 1.7 zu sehen.

1.7.0.1 Stromversorgung

Alle Komponenten, die angeschlossen werden, sind normalerweise über den Nucleo mit Strom versorgt. In der Testphase werden aber ebenfalls USB-to-TTL Konverter benutzt, die für eine ausreichende Stromversorgung garantieren. Sollte der Nucleo aber als alleiniger Stromversorger dienen, ist eine Betrachtung der möglichen Belastungen sinnvoll.

Die Spannungsversorgung kann über die USB-Schnittstelle (U5V) sichergestellt werden. Dabei kann das Board aber maximal 300mA verbrauchen. Wenn der Nucleo über die USB-Schnittstelle versorgt werden soll, ist es notwendig ein Jumper zwischen Pin 1 und Pin 2 von JP5 zu setzen, siehe Tabelle 1.3.

Ist die Belastung zu hoch z.B. durch eine Arduino- oder Morphoerweiterung, wird die MCU und die LED LD3 nicht mehr mit ausreichend Strom versorgt und geht aus. Falls zu wenig Strom geliefert wird oder wenn nur der MCU Teil betrieben werden soll, ist es sinnvoll eine externen Spannungsversorgung zu nutzen. Die externen Anschlüsse werden als VIN und E5V bezeichnet. Wie in Tabelle 1.2 dargestellt, ist mit einer externen Spannungsversorgung mit bis zu 800mA eine ausreichende Stromversorgung gewährleistet. [STM16a, S. 20]

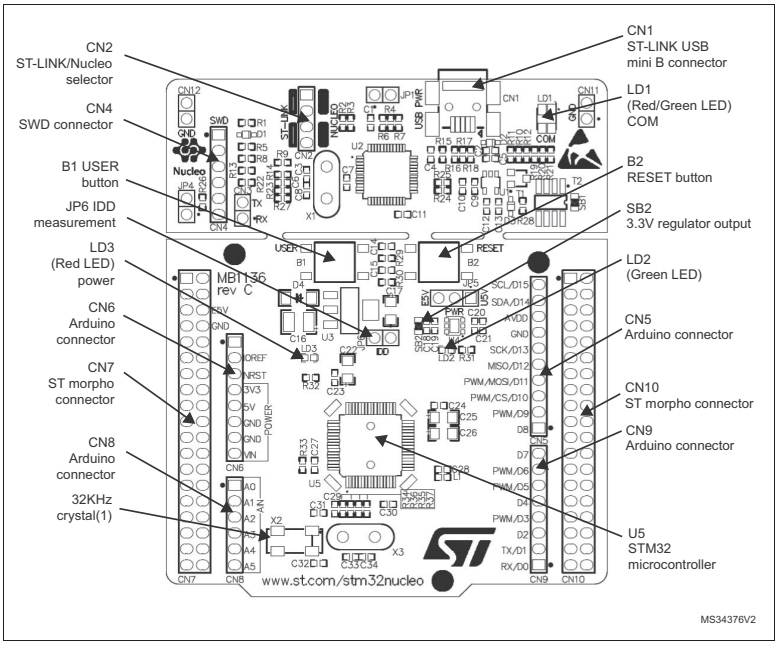


Abbildung 1.7: Layout mit Konnektoren von [STM16a, S.13]

Eingangsstrom	Pin Konnektoren	Spannung	maximaler Strom
VIN	CN6 Pin 8	7V bis 12V	800mA
E5V	CN7 Pin 6	4,75 bis 5,25V	500mA

Tabelle 1.2: Externe Spannungsquelle nach [STM16a, S. 21]

Jumper	Beschreibung
JP5	U5V ist als Stromversorgung gesetzt, wenn Pin 1 und Pin 2 verbunden sind. VIN oder E5V sind als Stromversorgung ausgewählt, wenn Pin 3 und Pin 2 verbunden sind.

Tabelle 1.3: Wahl der Stromversorgung nach [STM16a, S. 21]

1.7.0.2 LEDs

Die LED LD1 liefert Informationen über den Kommunikationsstatus des ST-Link Teil. Folgende Zustände sind möglich:

- langsam blinkend, rot: beim einschalten, vor USB Initialisierung
- schnell blinkend, rot: nach der ersten erfolgreichen Kommunikation zwischen PC und ST-Link/V2-1
- dauerhaft rot: Kommunikation zwischen PC und ST-Link/V2-1 ist abgeschlossen
- dauerhaft grün: Kommunikation erfolgreich und beendet
- blinkend, rot/grün: in der Kommunikation mit dem Zielgerät

- dauerhaft orange: Kommunikationsfehler

Eine weitere LED die vom Benutzer angesteuert werden kann, ist die LD2. Die LED ist eingeschalten, wenn PA5 (Pin 21) auf den logischen Pegel 1 (High) gesetzt wird. Die LED ist aus, wenn der logische Pegel 0 (Low) ist.

Die LD3 ist die LED, die den Status der Stromversorgung anzeigt. Man sollte immer darauf achten, dass diese LED rot leuchtet, dann wird der MCU Teil mit ausreichend Strom versorgt und es liegen +5V an. Ansonsten muss man wie in Abschnitt 1.7.0.1 beschrieben, den Strom erhöhen. [STM16a, S. 23]

1.7.0.3 Taster

Für die Kommunikation mit dem Benutzer, werden Taster verwendet. So kann z.B. die Aufnahme von WLAN-Daten gestoppt werden. Somit ist sichergestellt, das keine weiteren Daten geschrieben werden, wenn z.B. die SD-Karte entnommen werden soll. Der Taster für den Benutzer (B1) ist verbunden mit PC13 der MCU. Ein weiterer Taster ist der B2, dieser dient zum Resetten der MCU, um einen Neustart der Anwendung zu veranlassen. Der Taster ist verbunden mit NRST Pin. Man könnte also auch, falls der Taster B2 durch Erweiterungsmodule verdeckt ist, den NRST Pin auf logischen Pegel 1 setzen also auf High und somit einen Neustart erzwingen. In Tabelle 1.4 sind die Taster eingezeichnet.

1.7.0.4 Arduino Konnektoren

Die Arduino Konnektoren werden für das W5500 Erweiterungsmodul benötigt, welches als Speicherkartenmodul eingesetzt wird. Die Konnektoren CN5, CN6, CN8 und CN9 sind Buchsenleisten, die für den Arduino Standard entwickelt wurden. In Abbildung 1.8 sind diese markiert. Fast alle Shields, die für den Arduino erstellt wurden, können von den Nucleo Boards verwendet werden. Standardmäßig wird der Arduino Uno V3 von den Nucleo Board unterstützt. In Tabelle 1.4 sind die einzelnen Pins aufgeführt und mit deren Funktion beschrieben. [STM16a, S. 37]

1.7.0.5 USART

Die Universal Synchronous/Asynchronous Receiver Transmitter (USART) Schnittstelle ist an den STM32 Pins PA2 und PA3 über die Morpho- und die Arduinokonnektoren verfügbar. Die Auswahl kann über die entsprechenden Lötbrücken getroffen werden. Im Auslieferungszustand ist die Kommunikation zwischen der ST-Link MCU und dem STM32 Mikrocontroller aktiviert, um den virtuellen COM-Port nutzen zu können. Standardmäßig sind die dazugehörigen Lötbrücken wie folgt gesetzt: SB13 und SB14 verbunden und SB62 und SB63 getrennt. Für den Fall, dass PA2 und PA3 von einer Platine

Konnektor	Pin	Pin Name	STM32 Pin	Funktion
CN6	1	NC	-	-
Stromversorgung	2	IOREF	-	3,3V Ref
	3	RESET	NRST	RESET
	4	+3,3V	-	3,3 Input/Output
	5	+5V	-	5V Output
	6	GND	-	Masse
	7	GND	-	Masse
	8	VIN	-	Strom Eingang
CN8 analog	1	A0	PA0	ADC1_0
	2	A1	PA1	ADC1_1
	3	A2	PA4	ADC1_4
	4	A3	PB0	ADC1_8
	5	A4	PC1	ADC_IN11 (PC1)
			oder PB9	oder I2C1_SDA(PB9)
	6	A5	PC0	ADC_IN10 (PC0)
			oder PB8	oder I2C1_SCL (PB8)
CN5 digital	10	D15	PB8	I2C1_SCL
	10	D15	PB8	I2C1_SCL
	9	D14	PB9	I2C1_SDA
	8	AREF	-	AVDD
	7	GND	-	Masse
	6	D13	PA5	SPI_SCK
	5	D12	PA6	SPI_MISO
	4	D11	PA7	TIM1_CH1N oder SPI1_MOSI
	3	D10	PB6	TIM4_CH1 oder SPI_CS
	2	D9	PC7	TIM3_CH2
	1	D8	PA9	-
CN9 digital	8	D7	PA8	-
	7	D6	PB10	TIM2_CH3
	6	D5	PB4	TIM3_CH1
	5	D4	PB5	-
	4	D3	PB3	TIM2_CH2
	3	D2	PA10	-
	2	D1	PA2	USART2_TX
	1	D0	PA3	USART2_RX

Tabelle 1.4: Arduino Konnektoren nach [STM16a, S. 43]

benutzt werden und somit nicht für den virtuellen COM-Port eingesetzt werden können, müssen die Lötbrücken SB62 und SB63 verbunden und SB13 und SB14 getrennt werden. Die ST-Link MCU kann auch über eine andere USART-Schnittstelle verbunden werden z.B. über extra Drähte. Die USART2 (PA2, PA3) ist in den Standardeinstellung an den Debugger gebunden und kann nicht für andere Aufgaben verwendet werden. [STM16a, S. 25]

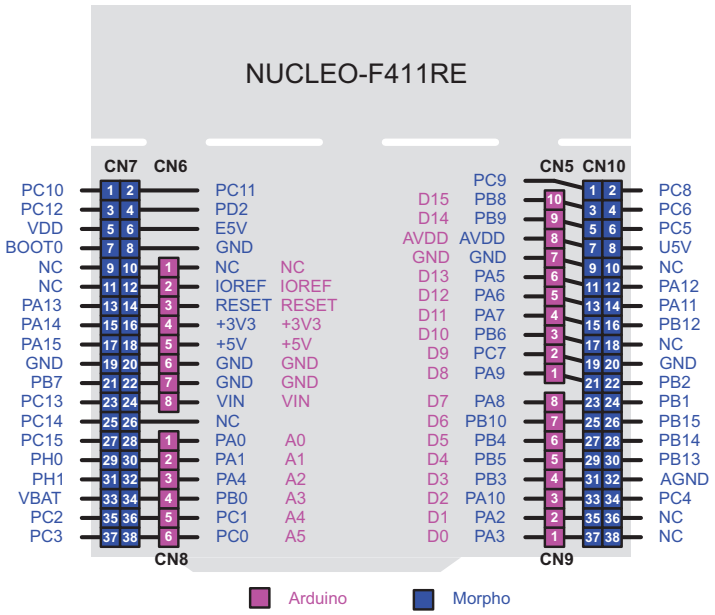


Abbildung 1.8: Pin-Belegung für Arduino und Morpho von [STM16a, S.32]

Zusammengefasst stehen folgende USART-Verbindungen zur Verfügung:

- D2 (USART1 RX)
- D8, D10 (USART1 TX)
- PA11 (USART6 TX)
- PA12 (USART6 RX)

Die Pin-Belegung ist in Abbildung 1.8 dargestellt.

1.7.0.6 ST-LINK/V2-1 Debugger

Das ST-LINK/V2-1 Programmier- und Debugging-Tool ist in das STM32 Nucleo-Board integriert. Es stellt, sobald es am Rechner über USB (CN1) angeschlossen wird, einen virtuellen COM Port zu Verfügung, sowie ein Massenspeichergerät. Will man das STM32 Nucleo-Board flashen, kopiert man nur die kompilierte Binärdatei auf den Massenspeicher. Es gibt zwei unterschiedliche Wege, um den integrierten STM-LINK/V2-1 zu benutzen. Entweder man programmiert bzw. debuggt den auf dem Board vorhandenen STM32 oder man benutzt eine externe MCU und verbindet diese mit dem SWD Connector CN4, wie in Tabelle 1.5 beschrieben. Die effektivste Methode ist, wenn man die bereits verfügbaren Funktionen des Boards verwendet. [STM16a, S. 16]

Jumper Status	Beschreibung
Beide CN2 Jumper verbunden	ST-LINK/V2-1 Funktionen für das direkte Programmieren auf dem Board ist aktiviert
Beide CN2 Jumpers getrennt	ST-LINK/V2-1 Funktionen für den externen CN4 aktiviert

Tabelle 1.5: Pin-Mode [STM16b, S. 16]

1.7.0.7 STM32F411RE

In dem Entwicklungsboard von STMicroelectronics ist ein Advanced RISC Machines (ARM) 32-Bit Cortex M4 CPU mit Floating Point (FPU) und einer Maximalfrequenz von 100 MHz verbaut. Dies ist die schnellste ARM CPU, die es in einem Nucleo gibt. Es stehen 512 KiB Flash Speicher und 128 KiB SRAM zur Verfügung, die für die Verwendung als Steuermodul für den ESP-01 ausreichend sind.

1.7.0.8 mBed als SDK

Um eine schnelle und effektive Programmierung zu ermöglichen, wird das mBed SDK verwendet. Es bietet für alle Funktionen eines eingebetteten Systems, wie in dem Fall des Nucleos, vereinfachte Methoden und Klassen.

Somit ist es nicht notwendig, jeden Teil der Hardware im Detail zu kennen. Die Methoden, die von $mBed$ bereitgestellt werden, erinnern an Methoden der Systemprogrammierung und sind somit verständlicher als hardwareabhängige Methoden des Herstellers. Um das $mBedSDK$ in die offline IDE wie Eclipse zu integrieren, besteht die Möglichkeit, ein vorhandenes online mBed-Projekt zu exportieren. Die Exportierung kann direkt als ein Projekt für $GNUARMEclipse$ ausgeführt werden. Allerdings kommt es immer wieder zu Fehlern beim Exportieren. Es besteht zusätzlich die Möglichkeit, das Projekt als GCC (ARM Embedded) zu exportieren. Danach muss das Projekt dann umständlich in Eclipse importiert werden.

1.8 W5500

Zur Speicherung der WLAN Daten wird ein Speichermodul benötigt, um die Datenpakete ablegen zu können. Um das Auslesen der WLAN Daten zu vereinfachen sollte ein mobiles Speichermodul mit einem üblichen Dateisystem verwendet werden. Somit kann eine einfache Auswertung mit üblichen Rechnersystemen erfolgen.

Das Arduino Shield W5500 von WIZnet bietet eine SPI-Schnittstelle, die mit einer microSD-Karten-Slot verbunden ist. Über die Software-Bibliotheken des Nucleos ist die Speiche-

rung der WLAN Daten in einem FAT16-Dateisystem möglich.

In der Abbildung 1.9 ist das Arduino Shield abgebildet. Zur Ansteuerung der SPI-Schnittstelle, um das SD-Karten-Modul zu nutzen, muss die Arduino Erweiterungsplatine auf die Konnektoren des Nucleos aufgesteckt werden. Für die Verwendung des SD-Karten-Moduls werden nur die PINs D13, D12, D11 und D4, sowie die Stromversorgung im 3,3V Bereich benötigt.

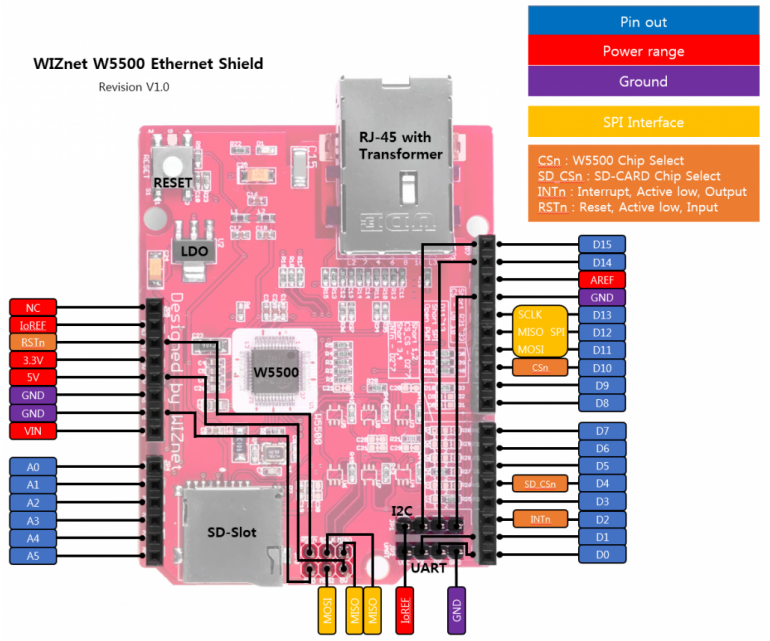


Abbildung 1.9: WIZnet W5500 Ethernet Shield von [WIZ16]

1.9 IDE

Eine IDE ist eine Sammlung von Softwareprogrammen, die den Programmierer in der Entwicklung unterstützt. In den meisten Entwicklungsumgebungen sind Editor, Compiler und Debugger integriert. Nach einer gewissen Einarbeitungs- und Einrichtungsphase dient dieses Anwendungsprogramm dazu dem Entwickler Routineaufgaben abzunehmen. Es ergeben sich aus der Verwendung einer IDE viele Vorteile:

- schnelle und unkomplizierte Navigation im Quelltext
- integrierte Versionsverwaltung (z.B. git)
- automatische Erstellung von Codegerüsten, z.B. Interfaces
- Autovervollständigung
- automatische Codeformatierung und Dokumentation
- integrierte Flash- und Debugfunktion

Die Unterteilung der IDE erfolgt auch in offline oder online Anwendung. Für den Nucleo gibt es beispielsweise eine online-IDE namens $mBed$, die kostenlos genutzt werden kann. Allerdings ist es nicht möglich, ein Debugging durchzuführen und die Quelltexte in einem externen Versionsverwaltungssystem zu nutzen. Die Funktion für den Export der Quelltexte funktioniert auch nicht zuverlässig. Ein weiterer Nachteil ist die Verfügbarkeit. Im Test von mBed kam es öfters zur Auslastung des Servers. Es gibt allerdings auch viele Vorteile. Die Einrichtung ist sehr einfach und man kann sofort nach der Anmeldung mit dem Programmieren beginnen.

Um den Quelltext offline jeder Zeit zur Verfügung zu haben und unabhängig von der Internetverbindung zu arbeiten, empfiehlt sich eine offline-IDE wie Eclipse mit der ARM GNU Toolchain für den Nucleo. Das SDK des ESP8266 kann ebenfalls in Eclipse integriert werden. Dies wird in Abschnitt 2.1.4 erklärt.

2 Methoden

2.1 ESP-01

Der ESP-01 dient in dem Projekt als Empfängermodul für die WLAN-Daten-Pakete nach IEEE 802.11. [Esp16a, S. 106]

2.1.1 Hardware Development Kit

Um die Verwendung des ESP-01 zu erleichtern und das Flashen zu ermöglichen muss eine Schaltung wie in der Abbildung 2.1 gezeigt, aufgebaut werden. Für den Betrieb

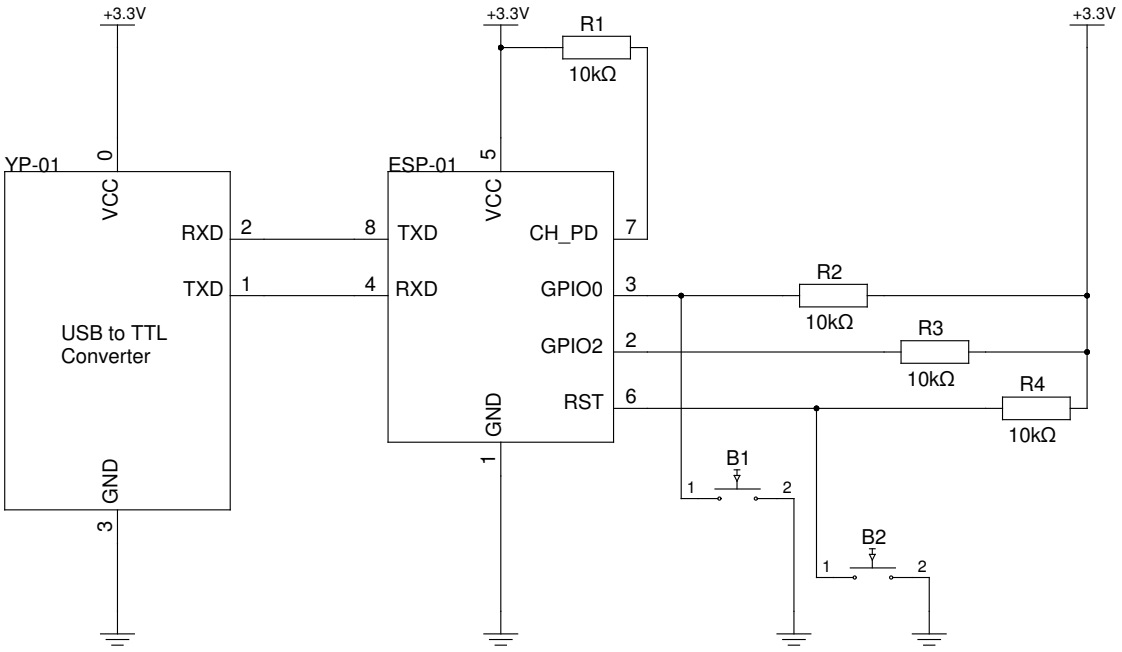


Abbildung 2.1: Schaltung zum Flashen (eigene Grafik nach [Esp17c, S. 8-9], [AI 15, S. 8])

über UART muss der CH_PD Pin über einen 10KΩ Widerstand mit dem VCC Pin verbunden werden. Das Selbe gilt für die $GPIO$ und RST Pins. Es ist also ein logischer Pegel 1 erforderlich. Des weiteren sind GND Pins mit der Masse zu verbinden. Die Spannung liefert der $YP-01$, ein USB-to-TTL Konverter.

Prinzipiell kann auch jeder andere USB zu TTL-UART Konverter benutzt werden, es sollte aber darauf geachtet werden, dass es sich um einen 3,3V Konverter handelt. Der ESP-01 ist, wie der darin verbaute ESP8266, nicht 5V tolerant.

Um den ESP-01 in den Flash-Mode zu bekommen, muss wie in Tabelle 2.1 dargestellt, der Pegel eingestellt werden. Die einfachste Umsetzung erfolgt durch eine Pull

up Schaltung. Beim Drücken des Tasters (B1) wird der Pegel von High zu Low. Durch die eingebauten Widerstände wird ein Kurzschluss verhindert. Es wird somit nur die Spannung reduziert. Um in den Flash-Mode zu gelangen, muss vor aktivieren des Reset Tasters (B2) die Pegel eingestellt sein, d.h. es muss der Taster für den GPIO0 (B1) gedrückt sein. Eine Auswertung der Pegel erfolgt vor dem Neustart.

Mode	GPIO0	GPIO2
UART	Logischer Pegel 0 (Low)	Logischer Pegel 1 (High)
Flash Boot	Logischer Pegel 1 (High)	Logischer Pegel 1 (High)

Tabelle 2.1: Pin-Mode [AI 15, S.9]

2.1.2 Toolkit

2.1.2.1 Compiler

Um das Non-OS SDK bauen zu können, das für dieses Projekt verwendet wird, werden das SDK selbst und eine virtuelle Maschine mit den vorinstallieren Compiler und den Tools zum Flashen benötigt. Die virtuelle Maschine wird von Espressif bereitgestellt.[2] [Esp17c, S.10]

Eine Alternative wäre das Installieren des $esp-open-sdk$ von $GitHub$[3]. Hier werden so weit wie möglich nur Open Source Quellen verwendet. Es ist auch bereits eine Toolchain integriert.

Da bei der virtuellen Maschine bereits alle Konfigurationen und Einstellungen vorgenommen wurden, wird sich in dieser Arbeit darauf beschränkt.

Nachdem man die virtuelle Maschine heruntergeladen und in VirtualBox integriert hat, kann man diese Starten und sich anmelden mit dem Passwort: $espressif$

Um das deutsche Tastaturlayout zu laden, gibt man folgenden Befehl ein und folgt den Anweisungen.

```
sudo dpkg-reconfigure console-setup
```

2.1.2.2 esptool.py

Das esptool.py ist ein Kommandozeilenwerkzeug zur Kommunikation mit dem ROM Bootloader im Espressif ESP8266 Mikrocontroller. Es erlaubt das Flashen von Firmware sowie das Zurückschreiben von Firmware und das Übertragen bzw. Setzen von

[2] https://drive.google.com/folderview?id=0B5bwBE9A5dBXaExvdDExVFNrUXM&usp=sharing
[3] https://github.com/pfalcon/esp-open-sdk

Chip-Parametern. Die Installation erfolgt über die Paketquellen von Ubuntu oder alternativ von Github[4]. Zur Installation über die Paketquellen benutzt man folgende Befehle:

```
sudo apt-get update
sudo apt-get install pip-python
sudo pip install esptool
```

2.1.2.3 Compilierung der Binärdateien

Am Anfang muss entschieden werden, ob die zu programmierende Anwendung als Firmware Over The Air (FOTA) oder als Non-FOTA programmiert werden soll. Bei der FOTA gibt es die Möglichkeit, die Firmware bei Änderung über die Espressif Cloud zu downloaden und direkt auf den ESP8266 zu flashen, solange der ESP8266 sich in einem Netzwerk mit Internetanschluss befindet. Es muss allerdings die Firmware auf die Espressif Cloud hochgeladen werden. Von dort kann dann der Upgrade Prozess gestartet werden. Die Reboot-Anweisung kann über einen HTTP-GET Befehl gesendet werden. Diese Methode ist sehr praktisch, wenn der ESP8266 in einer Anwendung ohne UART-Schnittstelle betrieben wird bzw. wenn der Zugang direkt zum Gerät nicht möglich ist. [Esp16b, S.] Diese Arbeit bezieht sich auf die FOTA Version, da es einen hohen praktischen Nutzen gibt, wenn die Funktionalität der drahtlosen Übertragung der Firmware auf den ESP8266 bereits vorgesehen ist.

Um die Compilierung zu beginnen, startet man das Script $gen\_misc.sh$, dabei sollte beachtet werden, dass das Script möglicherweise erst ausführbar gemacht werden muss:

```
chmod +x gen_misc.sh
```

Das Bash-Script gibt dabei die 5 Schritte zur Generierung der spezifischen Binärdateien vor. Im ersten Schritt wählt man zwischen den verschiedenen Bootversionen. Espressif empfiehlt die neueste Version des Bootloaders zu benutzen. In Abbildung 2.2 ist dargestellt, wie die Parameterauswahl für die FOTA aussehen muss. Im 1. Schritt muss ein Bootloader gewählt werden. Ohne Bootloader ist es nicht möglich die Funktionen der FOTA zu nutzen bzw. die Binärdateien zu compilieren. Der nächste Schritt betrifft die Auswahl der Binärdateien. Für FOTA ist es notwendig, die $user1.bin$ zu compilieren. Die $user2.bin$ ist nur bei aktiven Benutzung der FOTA notwendig. Weitere Binärdateien sind nur für die NON-FOTA zu verwenden. Bei den Schritten 3 und 4 sind die Standardwerte zu benutzen. Da bei dem ESP-01 ein 8Mbit Flash verbaut wurde, wählt man bei Schritt 5 als Größe die 1024KB(512KB+512KB). Die Gründe für die geteilte Angabe sind in Abschnitt 2.1.3 angegeben.

Um das Compilieren der Binärdateien zu vereinfachen bzw. eine komfortable Möglich-

[4] https://github.com/themadinventor/esptool

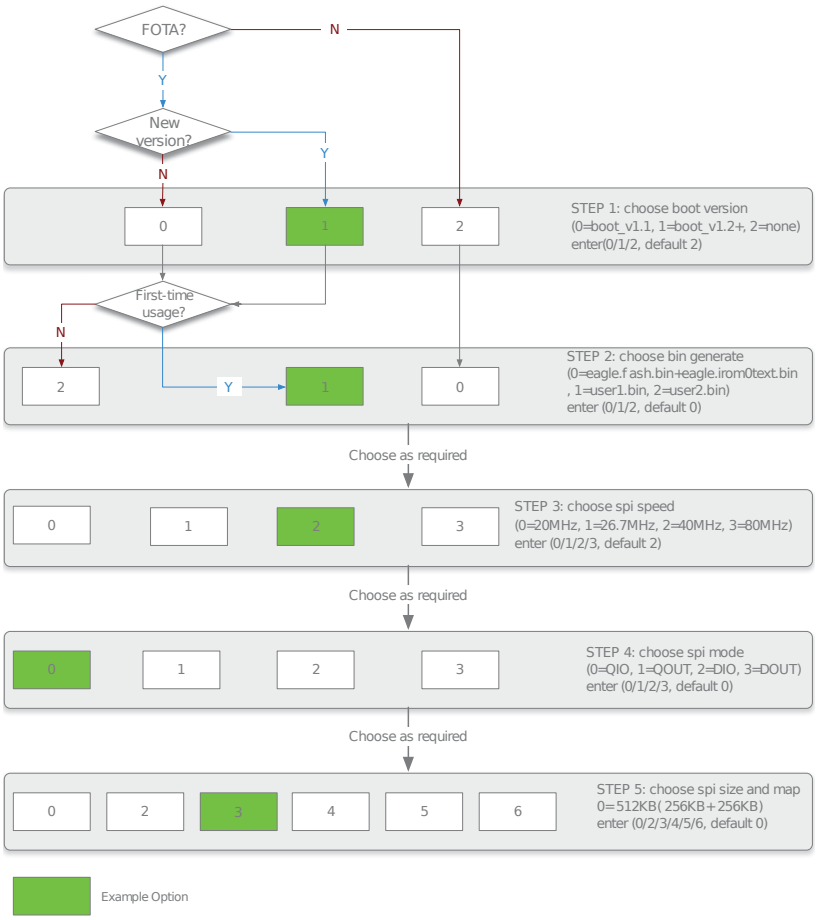


Abbildung 2.2: Compilierung SDK von [Esp17c, S. 27]

keit zu schaffen um schnell Binärdateien zu erzeugen, ohne die Parameter jedes mal erneut festzulegen, führt man den folgenden Befehl auf der Konsole im app bzw. Anwendungsordner aus.

```
make COMPILE=gcc BOOT=new APP=1 SPI_SPEED=40 SPI_MODE=QIO
    SPI_SIZE_MAP=2
```

Die Binärdatei, die den in $usermain.c$ geschriebenen Programmcode und die Includes enthält, wird in dem Ordner $/bin/upgrade$ mit dem Namen $user1.1024.new.2.bin$ abgelegt.

2.1.3 Flashen

Um den ESP8266 zu Flashen, ist notwendig nach einer $FlashMap$ vorzugehen. Die $FlashMap$ wird unterschieden zwischen Non-FOTA und FOTA. Die Flash Map für die FOTA ist in Abbildung 2.3 zu sehen.

In dem Fall des ESP-01 mit 1024KB SPI Speicher teilt sich der Speicherplatz auf in zwei mal 512KB Speicher. Der Bootloader wird auf den vorderen Speicherbereich geschrieben. Die aktuelle Bootversion des SDK ist die Version 1.6. Dann folgt der Speicherbereich, der die $user1.bin$ enthält. Bei der FOTA ist in der $user1.bin$ die $irom0text.bin$ enthalten, die die grundlegenden AT Befehle und Upgrade-Routinen der FOTA enthalten. Danach folgt der Bereich der User-Parameter. Es handelt sich hierbei um eine generierte Bin-Datei von Espressif, die für die eindeutige Identifikation des ESP-01, bei Einsatz der FOTA benötigt wird. Der Bereich des $user2.bin$ dient als Backup, falls es über den FOTA Upgrade Prozess zu einem Fehler käme. Dann kann immer noch der zweite Speicherbereich genutzt werden. [Esp16b, S.6] Die letzten 16KB sind unterteilt in die RF-Einstellungen und die Systemparameter des SDKs. In den RF-Einstellungen können Parameter wie z.B. Sendeleistung und Frequenzeinstellungen vorgenommen werden. [Esp16b, S.32]

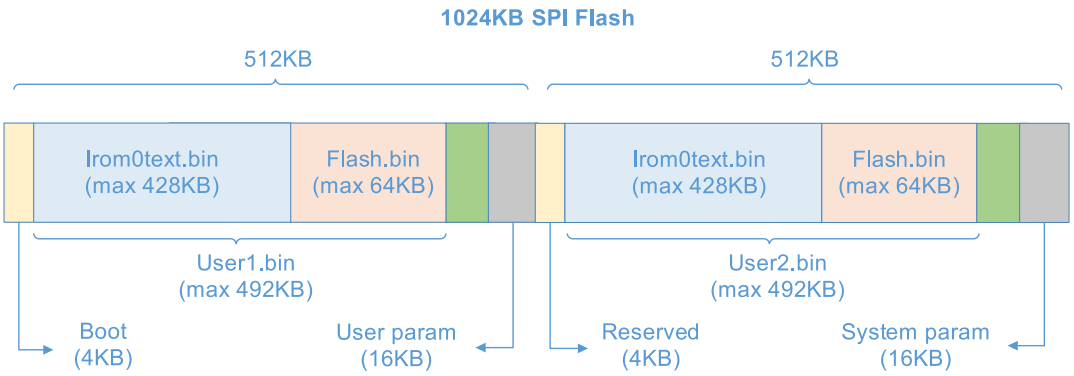


Abbildung 2.3: Flash Map von [Esp16b, S. 6]

Der Speicherbedarf der Binärdateien ist ebenfalls in der Abbildung 2.3 dargestellt. Die maximale Größe der $user1.bin$ und der $user2.bin$ beträgt 492KB. Der restliche Speicher kann mit Benutzerdaten beschrieben werden.

Die Adressen, die für die einzelnen Binärdateien genutzt werden sollen, sind in Tabelle 2.2 dargestellt. Die Adressen unterscheiden sich nach Größe des SPI Flash. In diesen Fall entsprechen die Adressen den für den ESP-01 üblichen 1MB.

Für den eigentlichen Flashvorgang benötigt man das in Abschnitt 2.1.2.2 installierte $esptool.py$ und die compilierten Binärdateien des SDKs. Um den ESP-01 flashen zu können, muss das Gerät in den Flash-Mode geschalten werden, siehe Abschnitt 2.1.1.

Binärdatei	Adresse	Description
boot.bin	0x00000 (ab 0KB)	Bootloader des SDKs
user1.bin	0x01000 (ab 4KB)	compilierte Anwendung siehe Abschnitt 2.1.2.3
master_device_key.bin	0x7E000 (ab 504KB)	User-Parameter
user2.bin	0x81000 (ab 516KB)	Backup der compilierten Anwendung
esp_init_data_default.bin	0xFC000 (ab 1008KB)	RF-Einstellungen
blank.bin	0xFE000 (ab 1016KB)	enthält Standard Systemparameter des SDK

Tabelle 2.2: Adressen der Binärdateien nach [Esp16b, S. 22]

Als erstes kann man den aktuellen Zustand des Flashs in eine Binärdatei auf die Festplatte schreiben. Dazu führt man folgenden Befehl in der Konsole aus:

```
esptool.py --port /dev/ttyUSB0 --baud 115200 read_flash 0 1048576
    backup.bin
```

In diesem Fall wird der komplette Flash in der Datei *backup.bin* abgelegt. Wenn nur bestimmte Speicherbereiche als Backup gewünscht sind, kann man nach dem Parameter *read_flash* den Speicherbereich in Byte angeben.

Als nächsten Schritt sollte man den kompletten SPI Flash löschen. Damit werden Fehler vermieden, die durch nicht überschriebene Speicherbereiche entstehen können.

```
esptool.py --port /dev/ttyUSB0 --baud 115200 erase_flash
```

Der ESP-01 wird durch diese Aktion nicht unbrauchbar gemacht. Es ist immer noch möglich den ESP-01 in den Flash-Modus zu schalten. Es befindet sich vermutlich ein ROM im inneren des ESP8266. Theoretisch ist es sogar möglich den SPI Flash zu wechseln, sollte durch zu viele Schreibprozesse der Flashspeicher unbrauchbar sein.

Um jetzt die Binärdaten auf den SPI Flash zu schreiben, wird nach den in Tabelle 2.2 gegebenen Adressen mit folgenden Befehl geflasht. Auch hierzu muss sich der ESP8266 im Flash-Mode befinden.

```
esptool.py --port /dev/ttyUSB0 write_flash 0x00000 boot_v1.6.bin 0
    x01000 upgrade/user1.1024.new.2.bin 0xfc000
    esp_init_data_default.bin 0xfe000 blank.bin
```

Nach dem Parameter *write_flash* wird als erstes die Adresse angegeben, danach der Namen der Binärdatei.

2.1.4 Integration des SDK in Eclipse

Für die Integration des SDK für den ESP8266 in eine IDE gab es bis jetzt von Espressif keine Anleitung. Da es sich bei dem SDK um eine Anwendung handelt, dessen Binärdateien per Makefile erstellt werden, ist es möglich, ein Makefile-Projekt in einer beliebigen IDE zu erzeugen. In dieser Arbeit wird Eclipse verwendet, da die Integration hier besonders leicht durchzuführen ist. Des Weiteren ist es sinnvoll, für die Entwicklung nur eine IDE zu verwenden.

Als erstes muss der komplette Ordner des SDK $ESP8266\_NONOS\_SDK$ in einen neuen Ordner verschoben werden, z.B. SDK. Das Problem ist, dass das SDK einen rekursiven Aufruf mehrerer Makefiles in verschiedenen Ordnerebenen durchführt. Um das SDK zu starten benötigt man ebenfalls ein Makefile. Um sich dem Zugriff durch den rekursiven Aufruf zu entziehen, darf der Ordner, der das SDK enthält, kein Makefile enthalten. Das Makefile wird stattdessen eine Ebene darüber platziert. Die Isolierung des $ESP8266\_NONOS\_SDK$-Ordners ist in Abbildung 2.4 noch einmal dargestellt.

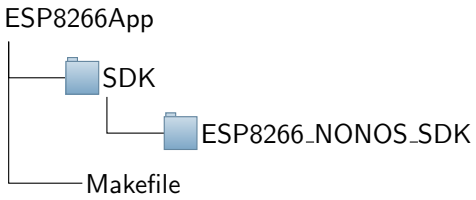


Abbildung 2.4: Ordnerstruktur in der IDE (eigene Grafik)

Das Makefile besteht unter anderem aus dem Target all, der für das Erstellen der Binärdateien verwendet wird. Als erstes muss mit dem Befehl cd an die Stelle navigiert werden, um das Makefile des SDKs aufzurufen. Es sollten absolute Pfade verwendet werden. Der Befehl $\&\&$ steht für die Verknüpfung von Befehlen. Die Funktionsweise des Makefiles des SDKs wurde bereits im Abschnitt 2.1.2.3 erläutert. Das Target $clean$ steht für das vollständige Entfernen aller erzeugten Binärdateien. Das letzte Target schreibt die Binärdateien auf den Flash des ESP-01. Der Aufbau des Befehls ist im Abschnitt 2.1.3 erklärt.

```
1 all:
2         cd ESP8266App/SDK/ESP8266_NONOS_SDK/app/ && $(MAKE) COMPILE
              =gcc BOOT=new APP=1 SPI_SPEED=40 SPI_MODE=QIO
              SPI_SIZE_MAP=2
3 clean:
4         cd ESP8266App/SDK/ESP8266_NONOS_SDK/app/ && $(MAKE) clean
5 flash:
6         esptool.py --port /dev/ttyUSB0 write_flash 0x01000
              ESP8266App/SDK/ESP8266_NONOS_SDK/bin/upgrade/user1.1024.
              new.2.bin
```

2.1.5 Promiscuous-Mode

Um den ESP-01 in den Promiscuous Mode zu schalten, ist es notwendig, den ESP-01 mit der beschrieben Methode und dem Quelltext, zu flashen. Es sind bei der vorinstallierten Firmware keine AT Befehle vorgesehen, die das Benutzen des Promiscuous Mode ermöglichen. In diesem Abschnitt werden die API-Funktionen vorgestellt, die für die Initialisierung des Promiscuous Mode notwendig sind und zusätzlich wird ein Überblick über die Hauptfunktionen des Quelltextes gegeben.

In der ersten Zeile wird der Promiscuous Mode aktiviert. Die Aktivierung kann nur erfolgen, wenn der ESP8266 sich im Station Mode befindet. Der ESP8266 kann in diesem Zustand nicht für andere Anwendungen verwendet werden. Es sollten auch keine weiteren API-Funktionen aufgerufen werden. In der zweiten Zeile steht eine API-Funktion, die eine Callback-Funktion *promisc_cb* registriert. Wird ein Paket über den Promiscuous Mode empfangen, erfolgt der Aufruf der Funktion *promisc_cb*, somit kann das empfangene Paket verarbeitet werden.

```
1 wifi_promiscuous_enable(1)
2 wifi_set_promiscuous_rx_cb(promisc_cb);
```

Die Methode, die bei Empfang eines Datenpaketes aufgerufen wird, besitzt als Parameter einen Puffer, der das Datenpaket enthält, und die Länge des Paketes in Bytes. Wird die Methode aufgerufen, wird über die *printf* Funktion die Länge des Paketes ausgegeben und danach folgt die Ausgabe des Puffers über die UART Schnittstelle. Die Daten des Puffers sind von eckigen Klammern umschlossen, um ein Einlesen zu vereinfachen. Jedes Ausgabe wird am Ende mit einem Zeilenumbruch abgeschlossen.

```
1  promisc_cb(uint8 *buf, uint16 len)
2  {
3          os_printf("(L):%d[", len);
4          int i=0;
5          while  (TRUE) {
6                   i++;
7                   if(i == len) break;
8                   os_printf("%02X",buf[i]);
9          }
10         os_printf("]\r\n");
11 }
```

2.2 W5500 als Speicherkartenmodul

Der W5500 ist, wie in der Einleitung bereits beschrieben, für das Speichern der abgegriffenen WLAN Daten des ESP-01 vorgesehen. Um eine Hardwareabstraktion zu schaffen, wurde eine Bibliothek mit dem Namen *SDFileSystem* von *mBed* benutzt. So-

mit wird die Möglichkeit geschaffen, das SPI-Karten-Modul auch auszutauschen und andere Hersteller zu verwenden.

Das Speicherkartenmodul kann über die Konnektoren auf den Nucleo gesteckt werden. In der Regel kann man das Modul so bereits verwenden. Um die Bibliothek für das SD-Karten-Modul nutzen zu können, muss diese eingebunden werden. In Eclipse gibt es eine Importfunktion, die dies auf einfache Weise möglich macht. Danach muss der Header der Bibliothek eingebunden werden. Dazu fügt man in die main.cpp folgenden Zeile hinzu:

```
import "SDFileSystem.h"
```

Zur Initialisierung der Bibliothek ist es notwendig, die PINs für die SPI-Schnittstelle anzugeben.

```
SDFileSystem sd(D11, D12, D13, D4, "sd");
sd.disk_initialize();
```

Danach erfolgt der Zugriff auf SD-Karte und das Erstellen einer Datei namens $test.txt$ mittels folgender Befehle:

```
FILE *fp = fopen("/sd/test.txt", "a");
```

Mit der Methode $fputs$ ist es möglich, die Datei mit einem char-Array zu füllen. Die Änderungen werden mittels der Methode $fclose$ auf die SD-Karte geschrieben. Beide Methoden sind in das mBed-Framework des Nucleos integriert und benötigen keine zusätzliche Bibliothek.

2.3 Integration des ESP-01 und des W5500 am Nucleo

2.3.1 Hardwareaufbau

Um den ESP-01 und den W5500 für das Aufzeichnen der WLAN-Daten am Nucleo nutzen zu können, müssen diese an den entsprechenden Schnittstellen angebracht werden. In der Abbildung 2.5 ist der Hardwareaufbau des Prototypes dargestellt.

Der ESP-01 ist über die UART-Schnittstelle mit dem Nucleo verbunden, dabei ist die TXD-Leitung des ESP-01 mit der RXD-Leitung des Nucleos verbunden, um die Übertragung der Daten vom ESP-01 zum Nucleo zu gewährleisten. Damit auch Daten vom Nucleo zum ESP-01 übertragen werden können, ist die TXD-Leitung des Nucleos mit der RXD-Leitung des ESP-01 verbunden. Diese UART-Schnittstelle wird für die Übertragung der Befehle zu Steuerung des ESP-01 verwendet. Der $GPIO2$-Pin wird als TXD-Leistung zum Übertragen der abgefangenen WLAN-Daten zum Nucleo verwendet.

Der W5500 ist über die SPI-Schnittstelle mit dem Nucleo verbunden.

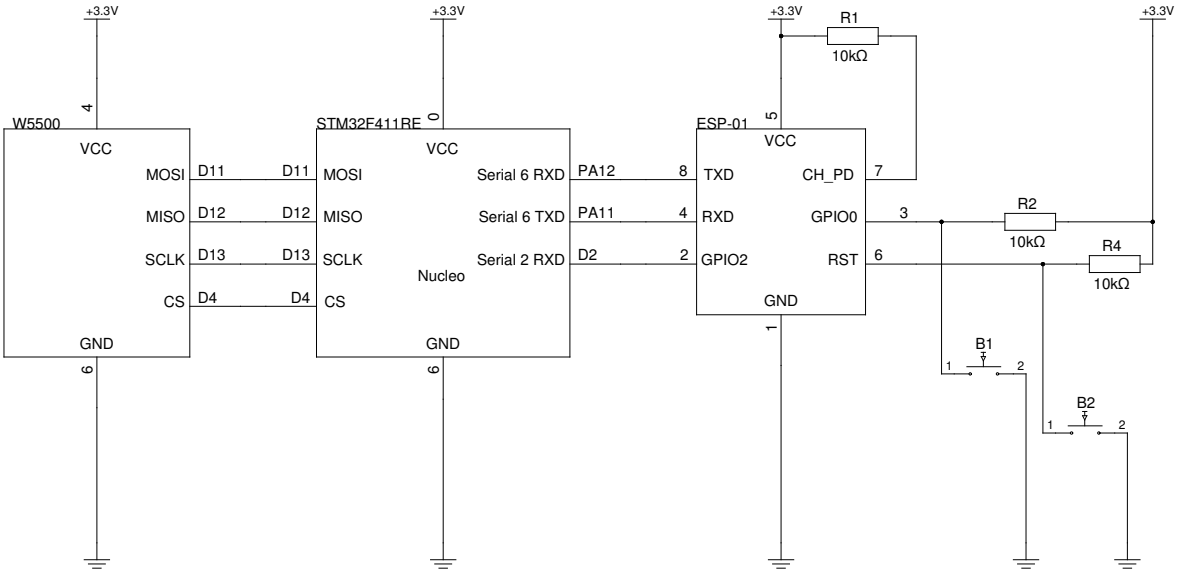


Abbildung 2.5: Anschluss des ESP-01 und des W5500 am Nucleo (eigene Grafik)

2.3.2 Programmablauf

Um eine Aufzeichnung von WLAN Daten zu beginnen, muss als erstes der *Channel* gesetzt werden. Dazu entnimmt man die SD-Karte aus dem SD-Kartenmodul und öffnet sie mit Hilfe eines Kartenlesegerätes am PC. Falls noch keine Datei mit dem Namen *channel.txt* existiert, ist diese anzulegen. Es sollte ebenfalls darauf geachtet werden, dass die SD-Karte richtig formatiert ist. Das notwendige Dateisystem ist FAT16. Sobald sich die SD-Karte wieder im SD-Kartenmodul befindet und der Nucleo und das SD-Kartenmodul sowie der ESP8266 mit 3,3V Spannung versorgt werden, beginnt der Nucleo sofort mit dem Aufzeichnen der WLAN Daten.

In der Abbildung 2.6 ist ein Flussdiagramm des Nucleos dargestellt, um den Programmablauf erläutern zu können. Das Flussdiagramm beginnt mit der Initialisierung aller Schnittstellen, wie UART und SPI. Erst danach ist es möglich, den ESP8266 anzusprechen und die Befehle zur Initialisierung zu übergeben. Auch das Speicherkartenmodul muss initialisiert werden. Dabei wird überprüft, ob die Speicherkarte beschreibbar ist. Als nächste müssen der $USERBUTTON$ und die $LED2$ für die Verwendung vorbereitet werden. Die LED signalisiert, dass die Verarbeitung beginnt. Bei dem ESP8266 ist es nicht möglich, auf allen Frequenzen gleichzeitig mitzuschneiden. Es muss ein bestimmter *Channel* gewählt werden. Sobald der *Channel* feststeht, kann der ESP8266 in den Promiscuous Mode gesetzt werden. Danach gelangt man in eine Endlosschleife wie bei eingebetteten Systemen üblich. In dieser Endlosschleife wird geprüft, ob der $USERBUTTON$ gedrückt wurde. Ist das der Fall, wird die LED ausgeschaltet. Es wird dem Benutzer signalisiert, dass kein Schreibvorgang auf die SD-Karte erfolgt und diese

entnommen werden kann. Wird der $USERBUTTON$ ein weiteres Mal gedrückt, wird die LED wieder eingeschaltet und es werden wieder Daten vom ESP8266 gelesen und auf die SD-Karte geschrieben.

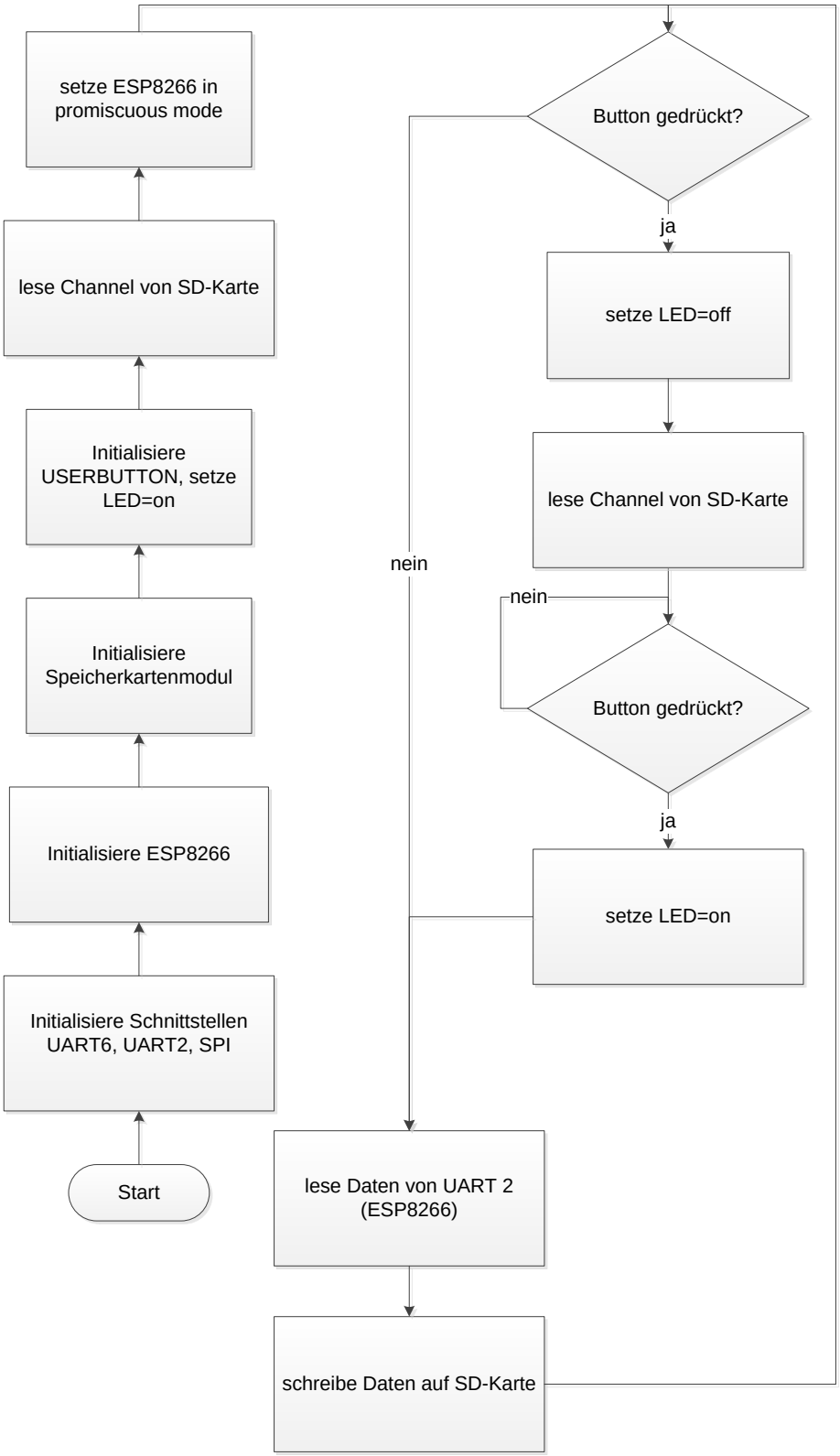


Abbildung 2.6: Flussdiagramm Nucleo STM32f411RE (eigene Grafik)

3 Ergebnis

3.1 Versuchsaufbau

Bei Versuchsaufbau handelt es sich um einen Access-Point (00:90:4C:C5:12:38) mit der SSID $APAndroid$ der auf dem Kanal 6 sendet und mit der WLAN-Karte (A0:88:B4:B5:CE:58) eines Rechners eine unverschlüsselte BSS Infrastruktur aufbaut. Es wird ebenfalls ein WLAN-Adapter in den Monitor Mode geschalten und mittels des Programms airodump-ng eine $.pcap$-Datei erzeugt, um einen Vergleich zu ermöglichen. Von dem Client wird ein $http$-Seite aufgerufen, um Datenverkehr über den Access-Point zu erzeugen.

3.2 Einlesen der WLAN Daten in Wireshark

Nachdem man die WLAN-Daten, wie in Abschnitt 2.3.2 beschrieben, aufgezeichnet hat, liegen die Pakete zeilenweise getrennt in der Datei $Dump.txt$ auf der SD-Karte.

Ein Problem ist, dass $Wireshark$ für den Import der Daten einen 802.11 Header erwartet, aber der ESP8266 liefert einen zusätzlichen Header mit dem $Wireshark$ nichts anfangen kann. Dieser Header ist Espressif spezifisch und enthält Informationen wie z.B. Pakettypen, Signalintensität und Frequenzinformationen. Genauere Informationen des so genannten RXControl befinden sich in Anhang A. Da dieser Header nur spezifische technische Informationen enthält und $Wireshark$ nicht damit umgehen kann, sollte der Header entfernt werden. Die einfachste Methode ist, die Anwendung eines regulären Ausdrucks in einem Editor wie z.B. gedit. Dazu wählt man die Option $Suchen\ und\ Ersetzen$. Der Suchausdruck lautet:

```
\r\n\w\w\w\w\w\w\w\w\w\w\w\w\w\w\w\w\w\w\w\w\w\w\w\w\w\w
```

Der zu ersetzende Ausdruck enthält:

```
\r\n
```

Die einzige Zeile, die manuell bearbeitet werden muss, ist die erste Zeile im Dokument. Dazu löscht man die ersten 22 Zeichen.

Um die Darstellung der Pakete zu vereinfachen, könnte man die Daten in $Wireshark$ importieren. Dazu ist es notwendig, die Pakete in ein für $Wireshark$ lesbare Darstellung zu bringen. Leider ist es nicht möglich, die Pakete bereits vom ESP8266 zu verarbeiten, da es bei höheren Datenaufkommen zum Verwerfen von Paketen kommt. Um die Pakete

umzuwandeln, benötigt man das Programm gen_hexdump [5].

Compilieren lässt sich das Programm mittels folgenden Befehl:

```
g++ -Wall -std=c++0x -o gen_hexdump gen_hexdump.cc
```

Um die Datei *Dump.txt* mit Hilfe des Programms umzuwandeln, benutzt man folgenden Befehl:

```
gen_hexdump -i Dump.txt -o Output.txt
```

Um die Pakete in *Wireshark* betrachten zu können, muss die Importfunktion genutzt werden. Dazu wählt man $Datei \rightarrow Aus\ Hex\ Dump\ importieren$. Bei den Importoptionen ist es erforderlich als Datenkapselungstyp das 802.11 WLAN Protokoll zu wählen.

3.3 Darstellung der Ergebnisse

3.3.1 Wireshark

Es ist festzustellen, dass der ESP8266 im promisuous Mode Frames, die nicht an seine MAC-Adresse gerichtet waren, aufgezeichnet hat.

In *Wireshark* (Abbildung 3.1) ist zu erkennen bis zu welcher Protokollebene Daten vom ESP8266 empfangen wurden. Es sind nur Paketheader vom 802.11 und dem LLC zu erkennen, obwohl es sich um eine unverschlüsselte Verbindung handelte. Es müssten aber ebenfalls TCP-Header bzw. IP-Header vorhanden sein. In dem Frame in Abbildung 3.2 sind dort aber keine identifizierbaren TCP-Header im Framebody enthalten. Ein weiterer Protokollheader, der gefunden wurde, ist das LLC Protokoll.

Es handelt sich bei den empfangenden Paketen hauptsächlich um so genannte *Beacon*-Frames. Die meisten *Beacon*-Frames besitzen eine Größe von 116 Bytes. Weiterhin sind Data-Frames vorhanden, dessen Größe 58 Bytes nicht überschreitet.

Die Protokollheader von dem WLAN Protokoll (Abbildung 3.2), dem QoS und dem LLC belegen 36 Byte. Die restlichen Bytes sind eine Struktur, die von Espressif hinzugefügt wurde und die Länge des Paketes enthält.

[5] https://github.com/bo-yang/misc/blob/master/gen_hexdump.cc

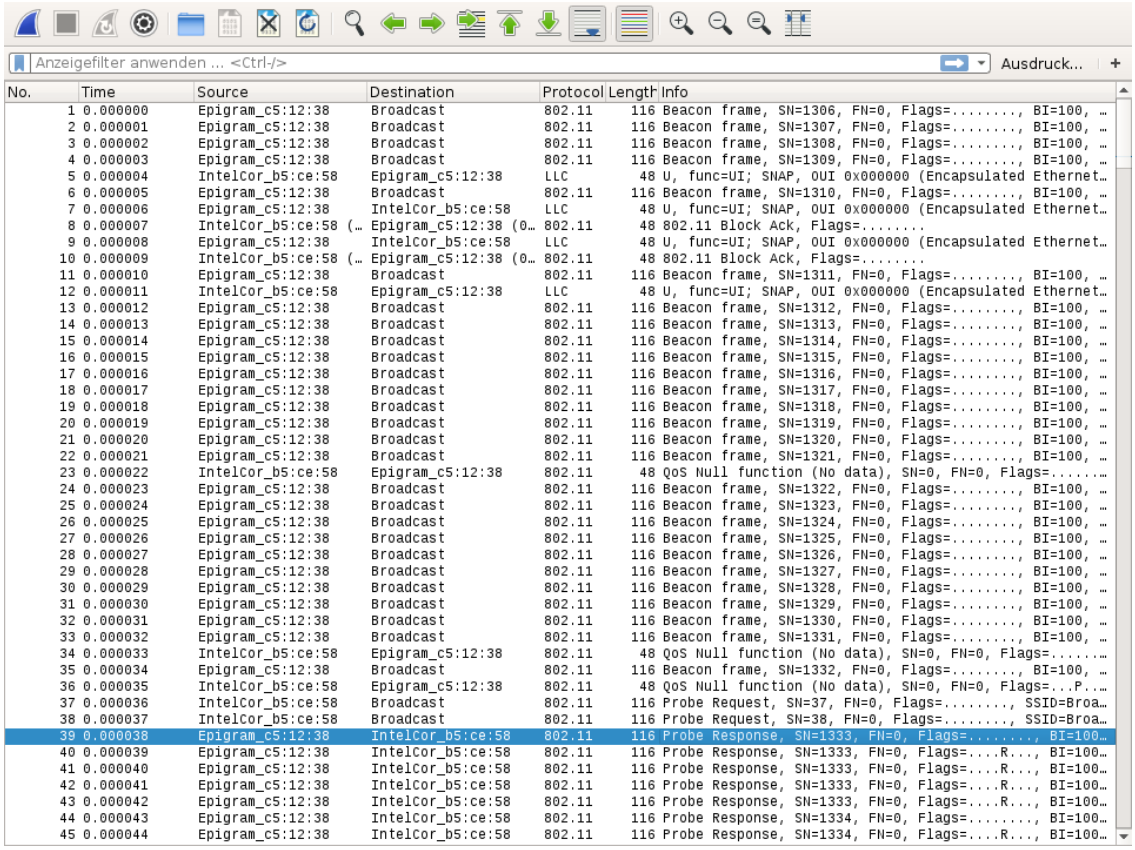


Abbildung 3.1: Überblick der aufgezeichneten Frames im Wireshark (eigene Grafik)

3.3.2 Espressif C-Struktur

Allgemein unterscheidet Espressif die Pakete in ihren C-Strukturen in $sniffer\_buf$ und $sniffer\_buf2$. Je nach Größe des Puffers werden die Pakete in einem dieser Strukturen zugeordnet.

```
struct sniffer_buf{
        struct RxControl rx_ctrl;
        u8 buf[36]; // Paketheader IEEE802.11
        u16 cnt;
        struct LenSeq lenseq[1];
};
struct sniffer_buf2{
        struct RxControl rx_ctrl;
        u8 buf[112];
        u16 cnt;
        u16 len;
};
```

[Esp16b, S. 99]

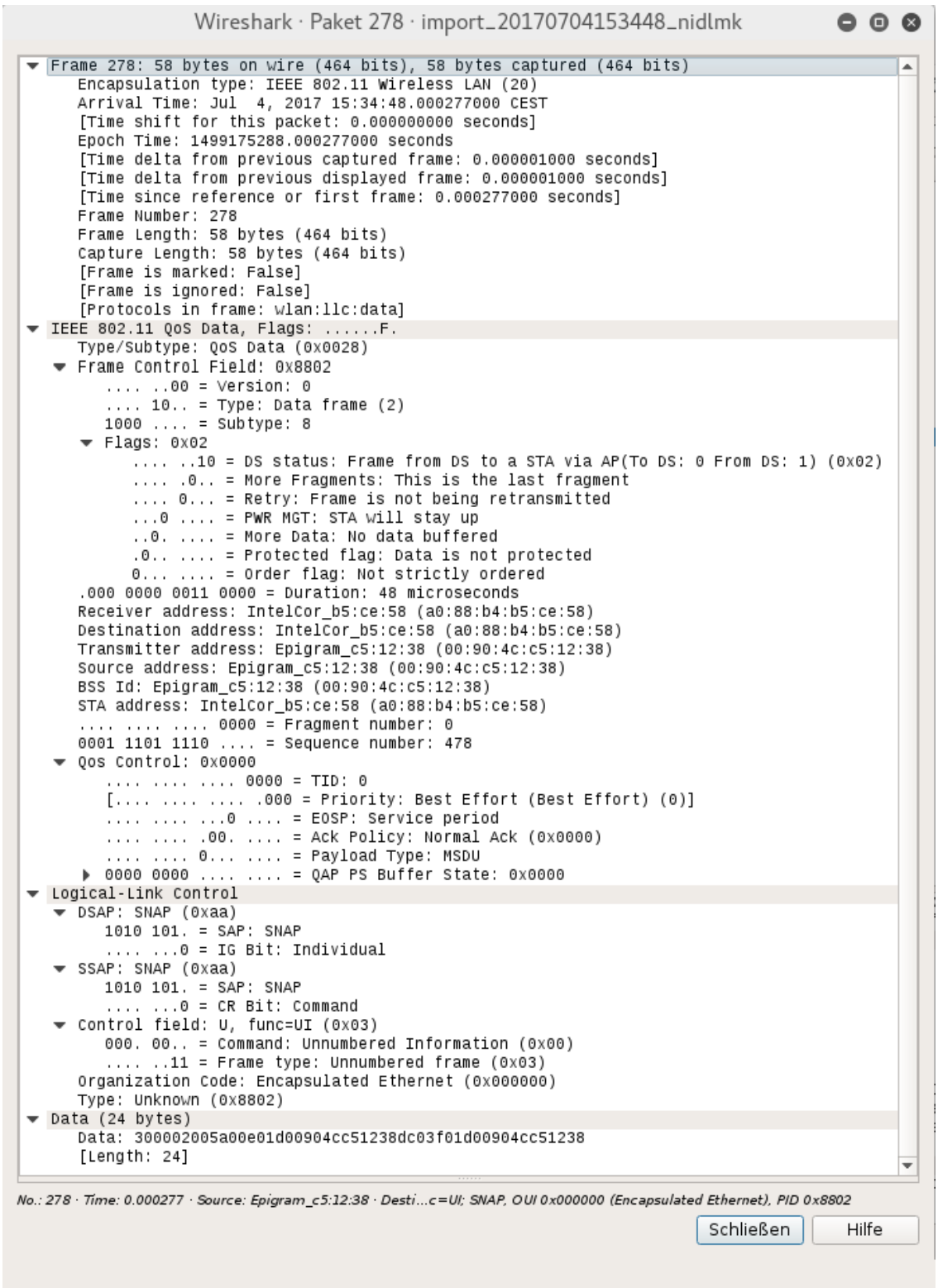


Abbildung 3.2: Frame im Wireshark vom ESP8266 aufgezeichnet (eigene Grafik)

4 Diskussion

4.1 Vor- und Nachteile von Linuxsystemen und eingebetteten Systemen

Um WLAN-Daten mitzuschneiden, gibt es ebenfalls die Möglichkeit einen Einplatinen Computer auf ARM-Basis mit einem Linuxsystem zu verwenden. Sehr beliebt sind Raspberry Pis, die mit einer WLAN-Adapter ausgerüstet werden, die den Monitor Mode unterstützen. Im weiteren Verlauf wird diese Kombination als Linuxsystem beschrieben. Diese Systeme sind ebenfalls für einen autarken Betrieb geeignet, wenn sie beispielweise mit einem Akku ausgestattet werden. [MWZ, S. 204]

Zum Vergleich wird der ESP8266 in Verbindung mit dem Nucleo STM32f411RE und dem SD-Kartenmodul herangezogen. Da es sich bei dem Nucleo um ein Entwicklungsboard handelt, wird sich bei der Größe auf den STM32f411RE bezogen, der das Herzstück des Nucleos ist.

Im Allgemeinen sind Linuxsysteme mit Linuxvorkenntnissen einfacher zu bedienen und anzupassen, da es für jeden Anwendungszweck bereits Lösungen in den Paketquellen gibt. So gibt es für die Linuxsysteme z.B. eine Aircrack-Suite, um WLAN-Daten mitzuschreiben. Ein weiterer Vorteil ist die freie Wahl von WLAN-Adaptern. Somit ist es möglich, USB-WLAN-Adapter für spezielle Anforderungen zu wählen z.B. mit Unterstützung für den Monitor Mode, der noch mehr Pakete mitschneidet als der promiscuous Mode, oder durch Verwendung eines Adapters mit einer höheren Sensitivität, wodurch sich eine erhöhte Reichweite ergibt. Somit kann der Abstand zum abzuhörenden Gerät deutlich verringert werden. Durch den 700MHz ARM Prozessor und den 512MB Arbeitsspeicher [MWZ, S. 205] ergibt sich ein leistungsstarkes System, welches mit großen Datenmengen zurecht kommt. Durch die Möglichkeit der individuelle Speicherkartenwahl mit bis zu 128GB ist ein genügend großer Speicher für WLAN-Daten möglich. Bei einer aktiven Internetverbindung des Linuxsystems gibt es die Möglichkeit der Kommunikation zum Anwender, z.B. über eine weiteres verfügbares WLAN. Durch Installation eines SSH-Dienstes ergibt sich die Möglichkeit, die aufgezeichneten WLAN-Daten direkt zum Benutzer zu senden. Im Idealfall sollte es sich nicht um den gleichen Kanal handeln, welches belauscht wird. Es könnte sonst der aufzuzeichnende Netzwerkverkehr unnötig manipuliert werden. Wenn es die Umstände zulassen, ist auch eine einfache Integration in ein LAN-Netzwerk möglich. Bei vorhandener Internetverbindung innerhalb des LANs ist ein Tunneln durch eine vorhanden Firewall über einen freien Port einfach zu realisieren. [MWZ, S. 206]

Nachteile von Linuxsystemen sind der erhöhte Stromverbrauch gegenüber eingebet-

teten Systemen. So verbraucht ein Linuxsystem in der passiven Erfassung von WLAN-Daten ca. 2000mA [MWZ, S. 207]. Das bedeutet, dass man mit einem Akku mit 5000mAh ca. 2,5h Betrieb ermöglicht wird. Für den realen Einsatz ist die kurze Betriebsdauer nicht praktikabel. Durch erhöhten Stromverbrauch steigt auch der Personaleinsatz der notwendig ist, um das System ununterbrochen zu betreiben. Ein Linuxsystem ist im Vergleich zu einem eingebetteten System relativ groß, so umfasst der Raspberry Pi $85,60mm \times 56mm \times 21mm$. Hierbei sind die Abmessungen des Akkus und eines geeigneten WLAN Adapters nicht mit kalkuliert. [MWZ, S. 204] Ein weiterer Nachteil sind die relativ hohen Kosten. Kommt es zu einem Verlust des Gerätes, sind Kosten im zwei bis dreistelligen Bereich zu erwarten. Das macht einen Einsatz in kritischen Bereichen, wie im Outdooreinsatz schwierig, da immer mit einem Verlust durch Diebstahl oder Ausfall zu rechnen ist. Hinzu kommt die Gewissheit, dass das Linuxsystem sich in fremden Händen befinden könnte und somit auch vertrauliche Informationen, die vom Betriebsystem und dessen Log-Dateien, sowie gespeicherten Passwörtern ausgehen, kompromittiert sind. Eine weiterer Nachteil ist das Verhalten des Betriebsystem bei Abstürzen z.B. bei Spannungschwankungen. Dadurch kann es zu Fehlfunktionen kommen und das Dateisystem beschädigen, wodurch ein Neustarten des Systems nicht mehr möglich ist. Hinzu kommt, dass für das automatisierte Aufzeichnen von WLAN-Daten eine zusätzliche Programmierung erforderlich ist, um beim Starten bzw. Neustart des Systems mit der Aufzeichnung zu beginnen. Wesentliche Vorteile des eingebetteten Systems sind der geringe Stromverbrauch. So verbraucht der ESP8266 mit dem Nucleo ca. 150-200mA und somit maximal 10% des Stromes eines Raspberry Pis. Bei 5000mAh ergibt sich somit eine Laufzeit von 25h. Dadurch wird er Personaleinsatz verringert um das System dauerhaft zu betreiben z.B. Akkus zu tauschen. Weitere Vorteile ergeben sich aus den geringen Kosten. Die günstigsten Preise für Mikrocontroller, wie der STM32F411RE und den ESP8266, befinden sich im geringen einstelligen Bereich. Mit den größten Kosten müssen bei den SD-Karten gerechnet werden. Es entsteht somit bei Verlust kein nennenswerter wirtschaftlicher Schaden. Weitere Vorteile sind die geringen Abmessungen und die davon ausgehende Flexibilität für die Auswahl des Standortes. Durch die geringen Kosten und die geringe Größe, kommen auch Standorte für die eingebetteten System in Frage, die sonst aus Kostengründen undenkbar gewesen wären, weil die Diebstahlgefahr oder die Ausfallwahrscheinlichkeit zu hoch sind z.B. im Outdoorbereich. Wird ein eingebettetes System von Dritten entwendet so verringert der Aufbau die Wahrscheinlichkeit, dass Daten vom System unbeabsichtigt an Dritte weitergegeben werden, da nur der Quelltext auf den integrierten SPI-Flash gespeichert werden. Dieser Quelltext könnte theoretisch durch Codeverschleierung vor Decompilierung des Codes geschützt werden.

Allerdings haben eingebettete System nicht nur Vorteile. Der größte Nachteil ist die geringe Leistung der Prozessoren. Der ESP8266 hat eine maximale Taktfrequenz von 160MHz. Dadurch ist damit zu rechnen, dass bei größeren Datenmengen im WLAN bestimmte Pakete nicht mit aufgezeichnet werden können. Durch zusätzliche Verarbeitungsschritte, wie die Formatierung des Ausgabestroms, wird der Prozessor ebenfalls

zusätzlich ausgelastet, wodurch es zum Paketverlust kommen kann. Ein weiterer Nachteil sind die eingesetzten WLAN-Module, die meistens nicht alle Paketarten verarbeiten können. Außerdem werden meisten von den Herstellern eigene Strukturen für die Verarbeitung der Pakete verwendet. Ein weiterer Nachteil ist, dass man bei der Entwicklung auf den Hersteller angewiesen ist. Sollte es sich bei dem SDK nicht um ein quelloffenes System handeln, ist man auf die Angaben des Herstellers angewiesen, die nicht immer vollständig und eindeutig sind. Ein Beispiel ist, dass bei der Entwicklung des NON-OS-SDK anscheinend nur auf die Ausgabe des 802.11 Paketheaders Wert gelegt wurde. Die Strukturen des ESP8266 geben für die eigentlichen Daten eine maximale Größe von 112 Byte vor und es waren keine TCP oder IP-Header zu finden.

4.2 Hypothesen

Hypothese 1: Der ESP8266 kann in Verbindung mit einem SD-Kartenmodul als Netzwerkscanner eingesetzt werden.

Wie in den Ergebnissen beschrieben gibt der ESP8266 mindestens 58 Bytes von mitgehörten Paketen aus. Der Header eines 802.11-Paketes besitzt eine Große von 36 Bytes. Diese enthält sowohl den Absender als auch den Empfänger des Paketes mit dessen MAC-Adressen. Auch in den Beacon-Frames sind Empfänger und Absender mit ihren MAC-Adressen enthalten. In den ersten 24 Bits lässt sich ebenfalls der Hersteller eines Gerätes ablesen, wodurch weitere nützliche Informationen gesammelt werden können. Durch die Pakete ist es möglich den Geräten eine BSSID zuzuordnen und somit zu ermitteln in welchen WLAN die Geräte angemeldet sind. Dadurch das der ESP8266 so stromsparend ist, kann dies über einen längeren Zeitraum erfolgen. Somit können Profile erstellt werden, welche Geräte sich um welche Zeit in ein bestimmtes WLAN einloggen. Dadurch können Störgeräte die z.B. die Netzwerksicherheit beeinträchtigen ermittelt werden oder einfach alle MAC-Adressen gesammelt werden, die für Netzwerkadministratoren interessant sind. Da die Geräte nur einen begrenzte Anzahl an Bytes von WLAN-Daten aufzeichnet, fällt ein geringer Speicherbedarf an. Es verringert sich der Personalaufwand, der mit dem häufigen Speicherkartenwechsel verbunden ist und zusätzlich entsteht weniger Rohmaterial, welches ausgewertet werden muss.

Hypothese 2: Der ESP8266 kann als Lösung alle bestehenden Systeme zum Aufzeichnen von WLAN-Daten ersetzen.

Dies ist eine Hypothese die eindeutig negativ beantwortet werden muss. Da der ESP8266 nur 802.11 Header in der Bitübertragungsschicht und der Sicherungsschicht aufzeichnet, ist eine Auswertung von höheren Schichten nicht möglich. Für Sicherheitsbehörden ist eine Aufgabe den Netzwerkverkehr von Verdächtigen mitzuschneiden um Beweismittel zu sichern. Da der ESP8266 nur die Header des 802.11 Paketes mitschneidet, sind keine Informationen über aufgerufenen IP-Adressen vorhanden, sowie keine Infor-

mationen über den Inhalt aufgerufener Websiten die das HTTP-Protokoll oder andere Protokolle die über der Sicherungsschicht des OSI-Modells liegen.

Hypothese 3: Durch die hohe Mobilität eines ESP8266 ergeben sich neue Anwendungen im Bereich der Netzwerkanalyse.

Normalerweise werden Computersysteme nur in Umgebungen eingesetzt, die vor Zutritt Dritter geschützt sind und wo eine permanente Stromversorgung gegeben ist. Durch die hohe Laufzeit mittels Akku ist man mit dem ESP8266 unabhängig von Stromquellen. Durch eine wasserundurchlässige Verpackung ist es sogar möglich den ESP8266 an Orten zu Lagern, die für andere Systeme nicht in Frage kommen und wo möglicherweise die Diebstahlgefahr sehr hoch ist. Da der ESP8266 neben dem 802.11 Header auch die Received Signal Strength Indication (RSSI)-Werte aufzeichnet ergeben sich weitere Verwendungsmöglichkeiten. Ein hoher RSSI Wert steht dabei für eine bessere Verbindung. Mit zunehmender Entfernung nimmt der RSSI Wert ab und man kann davon ausgehen, dass das Gerät weiter entfernt ist. Der RSSI Wert ist allerdings ebenfalls von Hindernissen zwischen den Geräten abhängig. Durch Verwendung mehrerer ESP8266 Geräte an verschiedenen Standorten könnte eine Ortung von WLAN-Geräten erfolgen.

4.3 Fazit

Mit der Verwendung eines ESP8266 und eines STM32f411RE sowie eines SD-Speicherkartenmoduls hat man ein eingebettetes System für die Aufzeichnung von WLAN-Daten im promiscuous Mode. Das System bietet für Netzwerkadministratoren und Sicherheitsbehörden eine interessante Alternative bei bestimmten Aufgaben z.B. der Aufklärung gegenüber herkömmlichen teuren und unflexiblen Systemen. Durch die geringen Kosten und der geringe Stromverbrauch hat ein eingebettetes System gegenüber den herkömmlichen Methoden wie das Mitschneiden von WLAN-Daten mittels Laptop oder eines Raspberry Pis starke Vorteile.

Der ESP8266, der für den Mitschneideprozess eingesetzt wird, hat seine Stärke im Aufzeichnen von 802.11 Headern und beschränkt sich auf die Bitübertragungsschicht und die Sicherungsschicht. Dadurch entfallen Aufgaben die auf höheren Schichten basieren wie das Auflösen von Routing-Problemen. Auch die Aufgaben der Sicherheitsbehörden die mit diesem System ausgeführt werden können, sind sich somit beschränkter.

4.4 Ausblick

Auch bei den eingebetteten Systemen gilt das Mooresche Gesetz. Es besagt, dass alle 18 Monate eine Verdopplung der Integrationsdichte in Halbleiterindustrie zu erwarten ist. [Sch97, S. 58] Es ist also zu erwarten, dass die Leistungsfähigkeit bei den eingebetteten Systemen weiter steigt und es in Zukunft eingebettete Systeme geben wird, die mit den großen Datenmengen, die in WLAN-Netzwerken anfallen umgehen können. Ein Schritt in diese Richtung ist der neue ESP32 von Espressif. Dieser verfügt über eine noch leistungsfähigere MCU und schneidet nicht nur den 802.11-Header mit. Durch höhere Leistung ist auch eine direkte Verarbeitung der WLAN-Mitschnitte möglich. Es entfällt die Verarbeitung die durch eine externe MCU notwendig war. Dadurch wird die Flexibiltät erhöht und die Kosten und der Stromverbrauch wird weiter gesenkt.

Anhang A: Aufbau RXControl

Der Aufbau des RXControl ist hier als 11Byte große Struktur dargestellt.

```
struct RxControl {
        signed rssi:8; //Signalintensitaet des Paketes
        unsigned rate:4;
        unsigned is_group:1;
        unsigned:1;
        unsigned sig_mode:2; //Falls 0, kein 11n Paket
        unsigned legacy_length:12; //Falls kein 11n Paket, laenge
           des Pakets
        unsigned damatch0:1;
        unsigned damatch1:1;
        unsigned bssidmatch0:1;
        unsigned bssidmatch1:1;
        unsigned MCS:7; //Falls 11n Paket, Anzeige der Modulation
        unsigned CWB:1; //Falls 11n Paket, Anzeige ob HT40 Paket
           oder nicht
        unsigned HT_length:16; //Falls 11n Paket, Angabe der Laenge
        unsigned Smoothing:1;
        unsigned Not_Sounding:1;
        unsigned:1;
        unsigned Aggregation:1;
        unsigned STBC:2;
        unsigned FEC_CODING:1;//Anzeige ob LDPC Paket oder nicht
        unsigned SGI:1;
        unsigned rxend_state:8;
        unsigned ampdu_cnt:8;
        unsigned channel:4; //genutzter Kanal des Pakets
        unsigned:12;
};
```

[Esp16b, S. 99-100]

Literaturverzeichnis

[AI 15] AI THINKER: *ESP-01 WiFi Module*. http://wiki.ai-thinker.com/lib/exe/fetch.php/modules/esp8266/aithinker_esp_01_datasheet_en_v1.0.pdf. Version: 1, 2015. – Abruf: 24.04.2017

[Bun17] BUNDESKRIMINALAMT: *Internetkriminalität / Cybercrime*. https://www.bka.de/DE/UnsereAufgaben/Deliktsbereiche/Internetkriminalitaet/internetkriminalitaet_node.html. Version: 2017. – Abruf: 18.05.2017

[Chr05] CHRISTIAN DOERR, MICHAEL NEUFELD, JEFF FIFIELD, TROY WEINGART, DOUGLAS C. SICKER, AND DIRK GRUNWALD: *2005 First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2005: DySPAN 2005 ; 8 - 11 Nov. 2005, [Baltimore, MD, USA ; conference record]: MultiMAC - An Adaptive MAC Framework*. Piscataway, NJ : IEEE Operations Center, 2005 http://ieeexplore.ieee.org/servlet/opac?punumber=10350. – ISBN 1–4244–0013–9

[EGOM] ELÇI, Atilla (Hrsg.) ; GAUR, Manoj S. (Hrsg.) ; ORGUN, Mehmet A. (Hrsg.) ; MAKAREVICH, Oleg B. (Hrsg.): *the 6th International Conference*

[Esp16a] ESPRESSIF: *ESP8266 FOTA Introduction*. http://www.espressif.com/sites/default/files/99c-esp8266_ota_upgrade_en_v1.6.pdf. Version: 1.6, 2016. – Abruf: 18.04.17

[Esp16b] ESPRESSIF: *ESP8266 Technical Reference*. https://espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf. Version: 1.2, 2016. – Abruf: 05.04.17

[Esp17a] ESPRESSIF: *ESP8266 SDK: Getting Started Guide*. https://espressif.com/sites/default/files/documentation/2a-esp8266-sdk_getting_started_guide_en.pdf. Version: 2.6, 2017. – Abruf: 24.03.2017

[Esp17b] ESPRESSIF: *ESP8266 System Description*. https://espressif.com/sites/default/files/documentation/0b-esp8266_system_description_en.pdf. Version: 2.0, 2017. – Abruf: 24.03.2017

[Esp17c] ESPRESSIF: *ESP8266EX Datasheet*. https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. Version: 5.3, 2017. – Abruf: 24.03.2017

[Eur17] EUROSTAT ; STATISTIA (Hrsg.): *Anteil der Haushalte in Deutschland mit Internetzugang von 2002 bis 2016*. https://de.statista.com/statistik/daten/studie/153257/umfrage/haushalte-mit-internetzugang-in-deutschland-seit-2002/. Version: 2017. – Abruf: 18.05.2017

[Gas13] GAST, Matthew S.: *802.11 wireless networks: The definitive guide ; [creating & administering wireless networks ; covers 802.11a, g, n & i]*. 2. ed. Beijing : O'Reilly, 2013. – ISBN 978–0–596–10052–0

[HH10] HOLT, Alan ; HUANG, Chi-Yu: *802.11 Wireless Networks: Security and Analysis*. London : Springer-Verlag London Limited, 2010 (Computer Communications and Networks). http://dx.doi.org/10.1007/978-1-84996-275-9. http://dx.doi.org/10.1007/978-1-84996-275-9. – ISBN 978–1–84996–274–2

[MBW10] MANDL, Peter ; BAKOMENKO, Andreas ; WEISS, Johannes: *Grundkurs Datenkommunikation: TCP/IP-basierte Kommunikation: Grundlagen, Konzepte und Standards*. 2., überarbeitete und aktualisierte Auflage. Wiesbaden : Vieweg+Teubner Verlag / GWV Fachverlage GmbH Wiesbaden, 2010. http://dx.doi.org/10.1007/978-3-8348-9699-5. http://dx.doi.org/10.1007/978-3-8348-9699-5. – ISBN 978–3–8348–0810–3

[MS05] MITESCU, Marian ; SUSNEA, Ioan: *Springer series in advanced microelectronics*. Bd. 18: *Microcontrollers in practice*. Berlin and Heidelberg : Springer, 2005 http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10143800. – ISBN 978–3–540–28308–9

[MWZ] MORTENSEN, Casey ; WINKELMAIER, Ryan ; ZHENG, Jun: Exploring attack vectors facilitated by miniaturized computers. In: ELÇI, Atilla (Hrsg.) ; GAUR, Manoj S. (Hrsg.) ; ORGUN, Mehmet A. (Hrsg.) ; MAKAREVICH, Oleg B. (Hrsg.): *the 6th International Conference*, S. 203–209

[Sch97] SCHALLER, R. R.: Moore's law: Past, present and future. In: *IEEE Spectrum* 34 (1997), Nr. 6, S. 52–59. http://dx.doi.org/10.1109/6.591665. – DOI 10.1109/6.591665. – ISSN 00189235

[STM16a] STMICROELECTRONICS: *STM32F411xC STM32F411xE*. http://www.st.com/content/ccc/resource/technical/document/datasheet/b3/a5/46/3b/b4/e5/4c/85/DM00115249.pdf/files/DM00115249.pdf/jcr:content/translations/en.DM00115249.pdf. Version: Rev 6 1/149, 2016. – 22.04.2017

[STM16b] STMICROELECTRONICS: *UM1724 User Manual: STM32 Nucleo-64 board*.

http://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf. Version: 11, 2016. – Abruf: 07.04.2017

[Ver16] VERBRAUCHS- UND MEDIENANALYSE: *Bevölkerung in Deutschland nach im Haushalt vorhandenen Internetzugangsarten in den Jahren 2013 bis 2016 (in Millionen)*. https://de.statista.com/statistik/daten/studie/171510/umfrage/im-haushalt-genutzte-internetzugangsarten/. Version: 2016. – Abruf: 18.05.2017

[WIZ16] WIZNET: *W5500 Ethernet Shield*. http://www.wiznet.io/product-item/w5500-ethernet-shield/. Version: 2016. – Abruf: 12.5.2017

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 14. August 2017