Der BierAlyzer

Die Dokumentation

v1.0

Autor:

Referent für Technik des AktionsBündnisses Durstiger Donnerstag

Dokument Version:

v1.0

Version	Datum	Beschreibung
v1.0	März 2015	Seit Jahren in der Pipeline und endlich fertig gestellte Version. Behandelte Versionen: • Hardware: v1.0 • Embedded Software: v1.1

Tabelle 0.1.: Dokument Version Historie

Inhaltsverzeichnis

1. Einführung

Das vorliegende Dokument ist die Dokumentation des BierAlyzers!

Der Entwurf, der Bau der Hardware und die Programmierung der Embedded-Software erfolgte durch den Technik-Referenten des AktionsBündnisses Durstiger Donnerstag. Er ist Inhaber von CC-Software und als solcher auch Ersteller des vorliegenden Dokuments.

Diese Dokumentation soll einerseits den Aufbau und die Funktionsweise des BierAlyzers offen legen, andererseits interessierten Freunden der Hardwareentwicklung und Studenten der Informations- und Elektrotechnik einen Anreiz bieten, im Studium Erlerntes sinnvoll anzuwenden. Die Zeit in der das Fachliche nur für das Bestehen von Laboren und Klausuren diente, sollte für den engagierten und begeisterten Studenten vorbei sein. Das Potential des technischen Studiums in Verbindung mit begeistertem Alkoholgenuss in vertrauter Runde ist enorm, dafür ist der BierAlyzer der beste Beweis.

Diese Dokumentation ist hingegen nicht gedacht für Personen und Studierende, deren Studium bzw. Bildung eine unangemessene finanzielle Bereicherung der eigenen Person auf Kosten der Gesellschaft zum Ziel hat. Der Autor ist sich jedoch sicher, dass dieser Personenkreis ohnedies nur minderes Interesse an technischen Realisierungen zur Steigerung der Gesellschaftsfähigkeit von gemeinsamen Tätigkeiten rein zum Wohl des Geistes hat, dies bleibt den begeisterten Enthusiasten vorbehalten.

1.1. Vorgeschichte

Als das AktionsBündnis Durstiger Donnerstag zu Beginn des Wintersemesters 2010/11 gegründet wurde, stand schnell fest, dass dieser Anfangs kleinen Gruppe vorbehalten war, zukünftig Maßstäbe technischer und gesellschaftlicher Art im Kreise der Studierenden an der Fachhochschule Kaiserslautern zu setzen. Es fehlte unter den Mitgliedern weder Begeisterung noch Bereitschaft, stets über die eigenen Fähigkeiten und Kompetenzen hinauszuwachsen um in gemeinschaftlichem Engagement die Zeit des Studiums als eine besondere Zeit des Lebens zu gestalten.
Von diesem Grundgedanken ausgehend entstand auch der BierAlyzer. Das Gründungsmitglied und zudem Referent für Technik und Suggestionsdesign des AktionsBündnisses startete zu Beginn des Sommersemesters 2011 das BierAlyzer-Projekt.

Die Zeit für die Entwicklung schien Anfangs nur schwer aufzubringen, da zeitgleich das Semester an der Hochschule bewältigt werden musste. Als Zeitquelle diente das Resultat einer

Beschneidung der eigenen Freizeit und somit besonders die Beschneidung der gemeinsamen Zeit mit dem weiblichen Lebenspartner des Autors zu dieser Zeit.

Relativ früh entstanden konkrete Pläne zu gewünschten Funktionen des BierAlyzers. Er sollte der statistischen Auswertung der von den Mitgliedern des AktionsBündnisses konsumierten Biermengen abhängig von definierten Events dienen. Dies sollte zusätzlich unbedingt unter Verwendung moderner Technologien erfolgen, um der technischen Professionalität des AktionsBündnisses besonderen Ausdruck zu verleihen.

1.2. Funktion des BierAlyzers

Der BierAlyzer sollte folgenden Punkten genügen:

- Handlichkeit und einfache Handhabung:
  - Die Schaltung muss in einem handlichen Gehäuse untergebracht werden
  - Die Stromversorgung erfolgt mit Batterien
  - Die Funktionsweise muss einfach und auch für Betrunkene leicht erkennbar sein
  - Die Identifikation des jeweiligen Mitglieds muss unkompliziert ablaufen
- Aufnahme, Sicherung und Abrufen der Daten (genossene Biermenge)
  - Intelligente und flexible interne Datenstruktur
  - Speicherung der Daten im nicht flüchtigen EEPROM
  - Auslesen der Daten per PC
- Steuerung und Konfiguration per PC-Software
  - Entwicklung der PC-Software AlcoLyzer
  - Konfiguration des BierAlyzers über den AlcoLyzer (Anlegen eines neuen Benutzers, Einstellen eines bestimmten Events)
  - Auswertung der Daten und Erstellen von Datenbanken durch den AlcoLyzer
- Nutzen aktueller und moderner Techniken
  - Verbindung mit dem PC via USB
  - Identifizieren des aktuellen Benutzers per RFID-Technologie (RFID-Transponder als Mitgliedsausweis)
  - Einsatz eines LC-Displays zur visuellen Kommunikation mit dem Benutzer

Bald danach entstand der Schaltplan und das Platinenlayout. Nach Fertigstellen der Hardware wurde mit der Entwicklung der Embedded-Software begonnen. Gegen Ende des Sommersemesters 2011 war die erste Betaversion der Firmware v1.0 fertig, sodass beim legendären 1.Flunkiball-Turnier des AktionsBündnisses am 12.Mai 2011 der Jungferneinsatz stattfinden konnte.

2. Hardware des BierAlyzers

2.1. Anforderungen an die Hardware

Bevor der Schaltplan zum BierAlyzer entstand, wurden freilich Überlegungen abstrakter Natur zusammengefasst. Diese konnten bezüglich der gewünschten Funktionalitäten recht einfach wie folgt zusammengefasst werden:

- Generelle Bedieneingabe des Benutzers bzw. Trinkers, sprich Tasten zum Drücken
- Generelle Kommunikation seitens des BierAlyzers zum Trinker, also ein Display
- Batteriebetrieben, da Keiner Bock hat beim Trinken nach Saft aus der Dose Ausschau zu halten
- Ein Schalter, um den BierAlyzer komplett abschalten zu können
- Ein USB-Anschluss, zur Konfiguration per PC wäre etwas Tolles, da wichtige Teile der Funktionalität, wie beispielsweise das Verwalten der Trinker aus der Embedded Software an eine PC Software ausgelagert werden kann. Solche Sachen machen in C keinen Spaß , da gibts das angenehmere Objektorientiertdingsbums, das mittels geeigneter GUI auch von technisch nicht versierten bedienbar ist.
- Ganz klar wird natürlich eine RFID-Funktionalität benötigt, um den jeweiligen Trinker auch individuell anhand seines Mitgliedsausweises identifizieren zu können.

Nach diesen nicht ins Detail gehenden Anforderungen ergeben sich freilich einige, deren technischer Tiefgang unausweichlich beim Auflisten ist.

Dazu zählt die Wahl des Mikrocontrollers, die Wahl der entsprechend integrierten Peripheriebausteine und eventuell vorzusehende Hardwarefunktionalitäten, die zukünftigen Firmwareversionen breitere Möglichkeiten bieten.

2.2. Wahl konkreter Bauteile

Da der Entwickler des BierAlyzers frei entscheiden konnte, was er wofür nimmt, ohne soziale Bedürfnisse technisch weniger versierter Teampartner bedienen zu müssen, mag dem Leser dieses Dokuments die ein oder andere Wahl des Bausteines willkürlich erscheinen.

Dass jedoch die entwickelte Hardware auf Anhieb ohne nachträgliche Änderungen funktionierte, räumt Zweifel bezüglich Richtigkeit der Bauteilewahl aus dem Weg.

Der Mikrocontroller

Das Herz der Schaltung bildet natürlich ein Mikrocontroller. Beliebte Hersteller der relativ simplen 8-Bitter Familie sind u.a. Microchips PICs und Atmels AVRs. Der Autor dieses Dokuments bekennt starke Zuneigung zu den Atmel AVR-RISC-Jungs, sodass die PICs nicht mal einer Überlegung gewürdigt wurden.

Jetzt fragt sich nur noch, welcher AVR das Rennen macht. Die große Anzahl verfügbarer Typen wurde durch die erwünschte USB-Funktionalität drastisch eingeschränkt.

Kurz und gut, die Wahl fiel auf den AT90USB162, ein Mikrocontroller mit Hardware-USB, 16 kByte Programm-Flash, 512 Byte EEPROM und diversen anderen Funktionalitäten (mehrere externe Interrupts, 2 Timer inkl. PWM, jeder Menge GPIOs und hastenichgesehn).

Der Mikrocontroller wird natürlich quarztechnisch getaktet, um der USB-PLL die erforderliche Genauigkeit zu bieten.

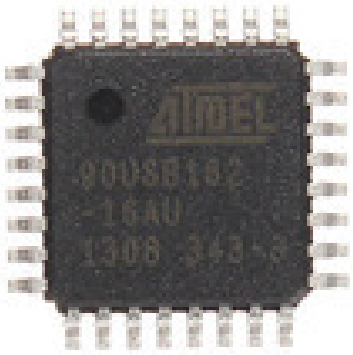

Abbildung 2.1.: Atmels Mikrocontroller AT90USB162 als Herz des BierAlyzers

Chip für RFID

Bevor die Suche nach einem geeigneten IC beginnen kann, muss man sich für eine der vielen RFID-Techniken entscheiden. In diesem Fall wurde sich auf die einfachste Variante beschränkt, da schließlich nur ein Benutzer bzw. Trinker anhand eines eindeutigen Musters erkennbar sein muss. Für diesen Fall bieten sich ReadOnly RFID-Chipkarten an, deren Inhalt neben mehreren Kontrollbits lediglich die fortlaufende Seriennummer ist.

Die Wahl fiel auf den EM4102 Standard. Per 125kHz RFID-Technik wird die 20 Bit Seriennummer im Mikrocontroller genutzt, den entsprechend vorher registrierten Trinker identifizieren zu können.

Als Baustein für die ganze Geschichte wurde der U2270B gewählt, der von mehreren Herstellern verfügbar ist. Die Sende/Empfangs-Spule wird über ein kleines Anpassungsnetzwerk dirket mit dem IC verbunden. Einzigstes hässliches Merkmal ist die Tatsache, dass die Daten manchester-like codiert an den Mikrocontroller übertragen werden. Das macht die Embedded-Software deutlich aufwendiger.

Es wäre scheinbar zu viel verlangt gewesen, eine bequeme synchrone Schnittstelle auf dem Chip zu implementieren.

Abbildung 2.2.: Der RFID-Chip U2270B

Display

Es wurde sich für ein alphanumerisches LC-Display entschieden, ausgestattet mit 4 Zeilen mit jeweils 20 Zeichen. Ausreichend um einem Trinker einige Informationen übermitteln zu können.

Das gewählte Display ist ein China-Teil vom Typ Y2004 mit HD44780 kompatiblem Controller. Um ein paar GPIO-Pins einzusparen, wird das Display im 4-Bit Modus betrieben.

Eine LED Hintergrundbeleuchtung ist integriert. Diese wurde mittels Emitterschaltung an einen PWM-Kanal angeschlossen, um die Helligkeit auch aus Energiespargründen einstellen zu können.

Abbildung 2.3.: Das alphanumerische LC-Display

Bauteile für Future-Use

Bisher nicht in der Firmware geplantes lohnt sich dennoch einzubauen, um einer höheren Firmwareversion weitere Hardwaremöglichkiten zu bieten.

Erster Punkt ist ein externer EEPROM, da der mikrocontrollerinterne mit 512 Byte offensichtlich etwas klein ist, um viele Benutzer inklusive ihrer Daten und Trinkleistungen an verschiedenen Events verwalten zu können. Es wurde aus diesem Grund ein serieller 8kByte EEPROM mittels I^2C angeschlossen.

Weiterhin erschien dem Autor eine zuverlässige Zeitbasis als hilfreich, weshalb noch eine RTC (Real Time Clock) des Typs DS1307 inklusive GoldCap als Puffer und entsprechender Ladeschaltung vorgesehen wurde. Auch dieser Baustein ist mittels I^2C angeschlossen worden.

Von beidem wurde in der Firmware v1.0 kein Gebrauch gemacht, wohl aber in v1.1.

Leider verfügt der Mikrocontroller nicht über ein Hardware-TWI-Interface, sodass das I²C-Interface in Software realisiert werden muss, sofern die Komponenten benutzt werden sollen.

2.3. Schaltplan

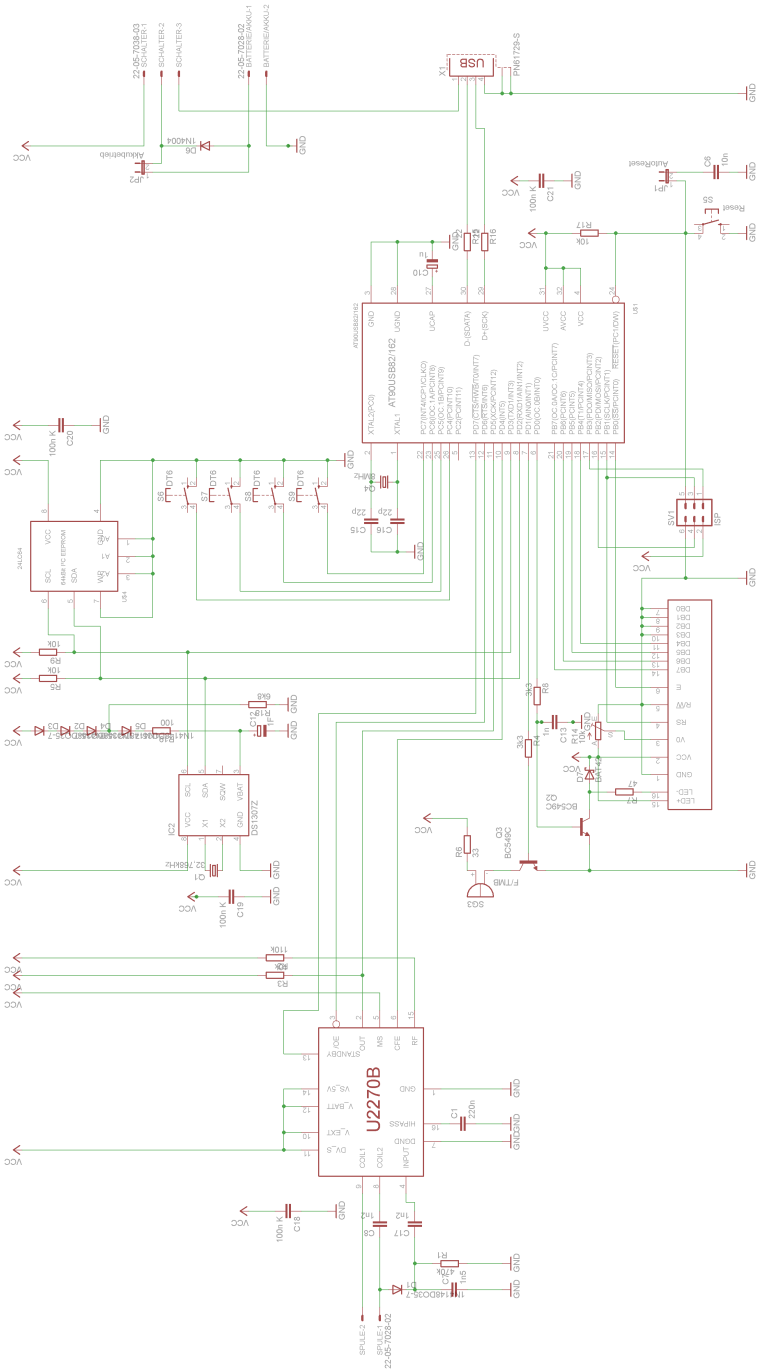

Abbildung 2.4.: Der Schaltplan des BierAlyzers

2.4. PCB-Layout

Nach umfangreicher Prüfung des Schaltplans wurde die Platinengröße festgelegt, die selbstverständlich dem verwenteten Gehäuse angepasst werden musste.

Nach Festlegung der Anordnung der Drucktaster, des Displays und der USB-Buchse konnte in EAGLE geroutet werden. Die Drucktaster wurden direkt unter dem Display angebracht. Dies spart die Tastenbeschriftung, da diese Information durch das Display angezeigt kann und somit auch flexibel ist, je nach Art der vom Benutzer bzw. Trinker gewünschten Interaktion.

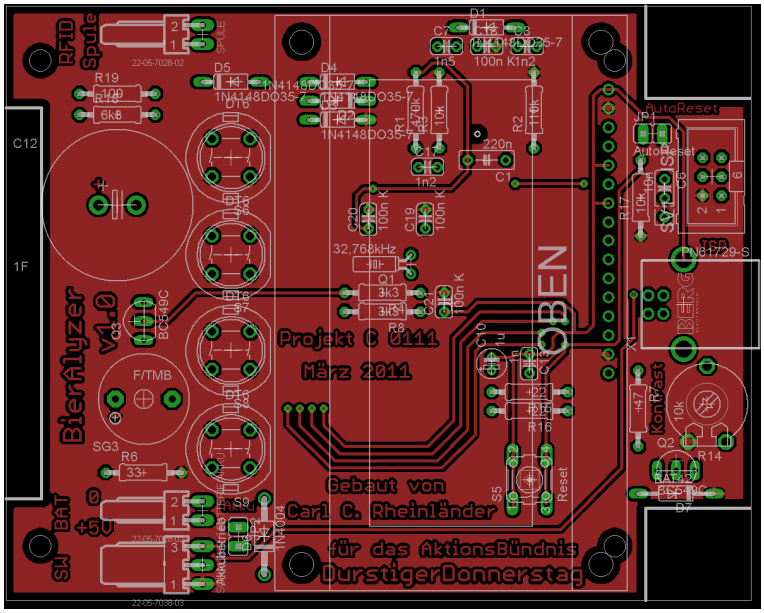

Abbildung 2.5.: Vorderseite des Platinenlayouts

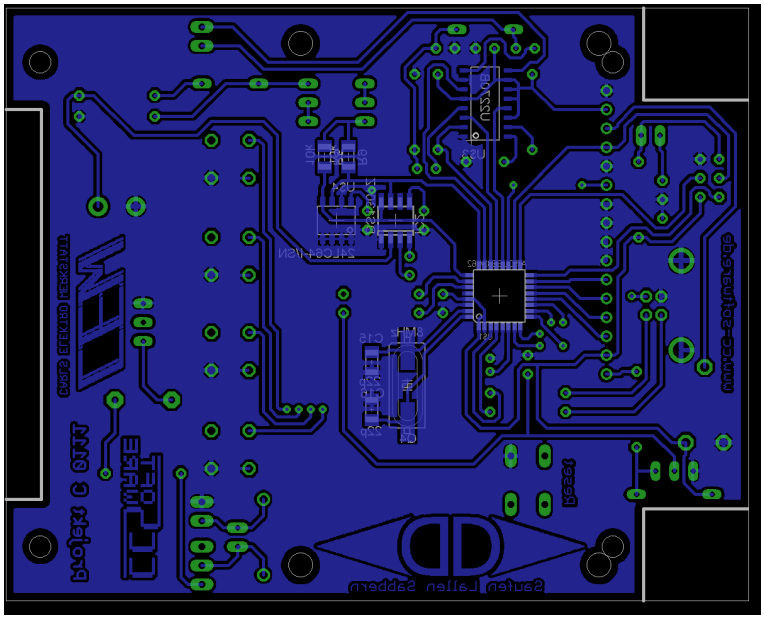

Abbildung 2.6.: Rückseite des Platinenlayouts

3. Embedded-Software des BierAlyzers

Um der an sich toten Hardware Leben einzuhauchen muss der Flash des Mikrocontrollers mit Nullen und Einsen gefüllt werden.

Dazu gibt es mehrere Möglichkeiten der Vorgehensweise:

1. Maschinencode direkt schreiben, sprich in Assembler programmieren
2. Weicheier-Programmiersprachen wie bspw. Basic verwenden
3. Grafische „Programmierung“ für Dilettanten wie bspw. Flowcode
4. Objektorientiert programmieren
5. Software in C schreiben

Da die Software irgendwann fertig werden sollte, wurde Ersteres nicht gemacht.

Verwenden einer Weicheier-Programmiersprache ist grundsätzlich unter dem Niveau eines Elektrotechnikers, sodass diese Möglichkeit aus Selbstwertgefühl nicht in Erwägung gezogen wurde. Außerdem entbehren sämtliche nicht-objektorientierte Programmiersprachen außer C jeglicher Ästhetik (Assembler sei hier ausgenommen, da dessen Ästhetik unangreifbar ist).

Dies bedenkend müssen hier keine weiteren Worte bzgl. Punkt drei verschwendet werden.

Objektorientierte Programmierung auf Mikrocontrollerebene ohne Betriebssystem macht aus Sicht des Autors keinen Sinn.

So bleibt das gute alte C übrig, der treue Begleiter des technischen Informatikers seit vielen Jahrzehnten!

Da es sich um einen Mikrocontroller von Atmel handelt, wurde auch das AVR Studio als Entwicklungsumgebung benutzt. So entstand kein Ärger mit Makefiles oder lästiger Konfiguration anderer IDEs wie das Schreckensbeispiel Eclipse.

Zusammengefasst:

- **Entwicklungsumgebung:** Atmel AVR Studio 6 [7]
- **Programmiersprache:** C
- **Compiler:** WinAVR 20100110 [8]

Die in diesem Dokument beschriebene Embedded-Software ist die BierAlyzer Firmware v1.1.

Version v1.0 wird nicht beschrieben. Es handelt sich um eine stellenweise dirty-gehackte Software. Trotz Komplexität lief diese Software bugfrei und erfreute viele Jahre das Herz der Trinkergesellen. Ehre sei ihr zuteil!

Software aus externen Quellen

Das Modul *USB Raw-HID Driver* stammt komplett aus einer externen Quelle. Der Quelltext ist Bestandteil des Teensy RawHID Projektes[3].
Der Code darf frei verwendet werden, sofern der originale Kommentartext beibehalten wird. Änderungen am Code war die Anpassung an den 8MHz Quarz, der eine andere Konfiguration der USB-PLL erfordert sowie Änderung des Produktnamens, als der das USB-Gerät am PC erscheint.

In der Firmware v1.0 wurde zur Auswertung des Datensignals des RFID-Chip auf einen Quelltext von Stefan Seegel[4] zurückgegriffen, der auf mikrocontroller.net[5] veröffentlicht ist.
In Firmware v1.1 wurde daraus nur noch der Teil zur Berechnung der Paritätswerte der Seriennummer benutzt, die Manchesterdekodierung wurde hier vom Autor neu und interruptbasiert implementiert.

3.1. Anforderungen an die Software

Bei Betrachten der gewünschten Funktionalitäten des BierAlyzers, lassen sich die folgenden Teilbereiche als Anforderungen auflisten:

- Kommunikation mit einem PC über USB:

  Zum Einstellen von Events

  Zum Registrieren eines Trinkers

  Zum Auslesen der Biermengen

- Kommunikation mit RFID-Mitgliedskarten (Erkennen eines Trinkers)
- Steuerung der abstrakten Funktionalität; nach Auflegen der Mitgliedskarte über die Taster die gewünschte Biermenge buchen können
- Powermanagement: Energie sparen, wenn vergessen wird, den BierAlyzer auszuschalten. Die Wahrscheinlichkeit dieses Vergessens steigt nachweislich mit steigendem Alkoholpegel!
- Management des EEPROMs, zur Verwaltung der Events und Trinker, der Seriennummern derer RFID-Karten und der Möglichkeit die Biermengen getrennt für mehrere Events speichern zu können.
- Ansteuerung des LC-Displays zur Visualisierung der internen Vorgänge

3.2. Aufbau

Da sich die Anforderungen ganz leicht in logische Bereiche untergliedern lassen, währe es milde gesagt unvernünftig, im Vorfeld keine Überlegungen zu einem modularen Aufbau zu machen.

Folgende Softwaremodule sind in der Embedded-Software enthalten, wobei jedes Modul praktisch aus eine Quell- und einer Headerdatei besteht:

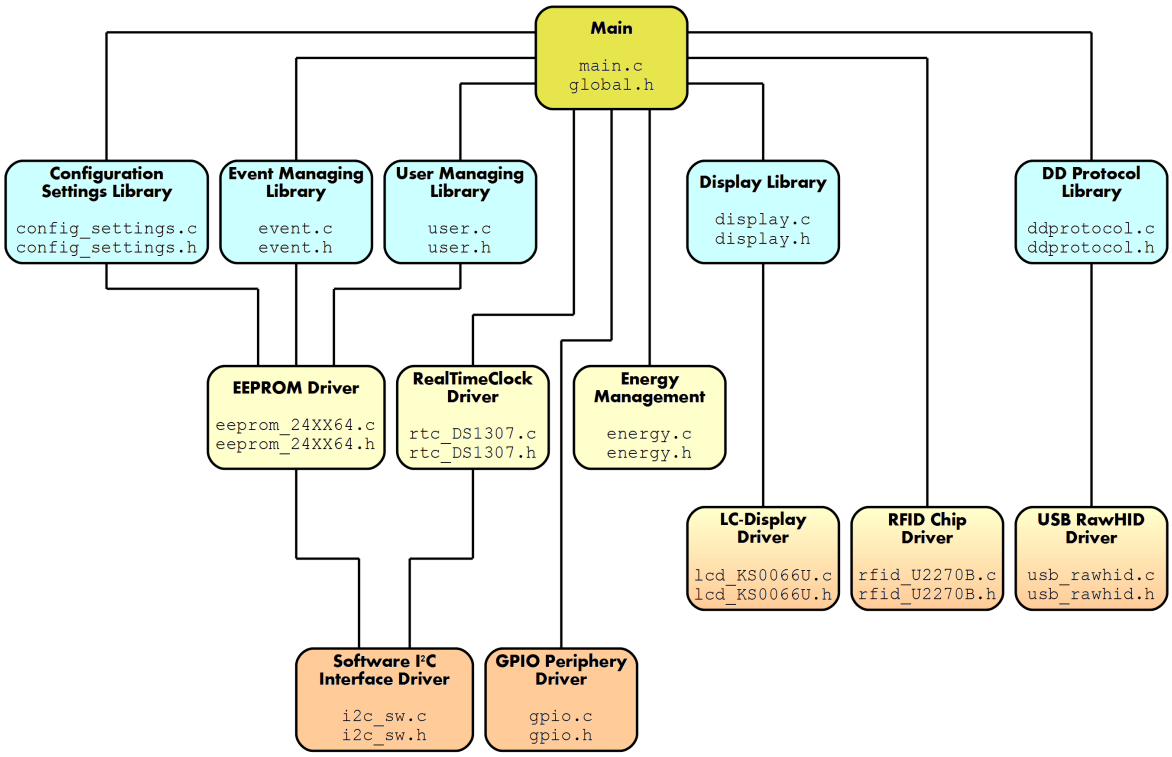

Abbildung 3.1.: Die Module der Embedded-Software

Library

Eine Library/Bibliothek definiert und fasst Funktionalitäten zusammen. Sie interagiert nicht direkt mit der Hardware bzw. Register des Mikrocontrollers. Entsprechende Treiber werden von ihr dirigiert, um den Job der Hardwarekommunikation zu erledigen. Der Bibliothek ist es grundsätzlich scheißegal, wer die Hardwarekommunikation macht.

Nicht verstanden? Ok, kleines Beispiel:
Die Displaybibliothek stellt z.B. die Funktionalität bereit, das DD Logo auf einem LC-Display darzustellen. Ihr ist es dabei aber völlig gleich, ob das ein HD44780er LCD oder ein LED-DotMatrix Display tut.

Driver

Dem Driver/Treiber hingegen ist das, was der Bibliothek egal ist, nicht egal. Er muss speziell für das entsprechende Display geschrieben werden, da die Ansteuerung unterschiedlicher Displaytypen in der Regel auch unterschiedlich ist.

Der Treiber wird daher auf die Hardware zugeschnitten, bspw. auf den RealTimeClock Chip (RTC). Letzterer ist ein gutes Beispiel für einen Geräte/Device Treiber. Dieser Treiber kann entweder direkt mittels physikalischer Pins auf das Device zugreifen, oder andere Treiber dazu benutzen.
Ein Beispiel zu letzterem ist der Treiber der I^2C Schnittstelle, der hier unglücklicherweise in Software geschrieben werden musste, da der verwendete Mikrocontroller kein Hardware-I^2C Interface besitzt.

3.3. Funktion

So ein modellbasierter und in Layer geteilter Aufbau ist was tolles, wenn es um Übersichtlichkeit und Dokumentation geht.

Als fehlt jetzt noch, dass die Software auch irgendwas sinnvolles tut, sprich der funktionelle Aufbau.

3.3.1. Funktionsweise der Module

Wie die einzelnen Module der Software funktionell aufgebaut sind ist aus dem Quellcode ersichtlich, der gemäß doxygen [9] dokumentiert wurde. Um keine redundante Informationen aufzustellen und dem Entwickler Zeit zu ersparen, wurde hier deshalb auf einen genauere Beschreibung verzichtet.

3.3.2. State Machine als zentrales Funktionselement

Die Steuerung der von den einzelnen Modulen abstrahierten Funktionalität wurde in Form einer State Machine realisiert. Der zentrale Teil der Endlosschleife enthält dazu ein switch-case Konstrukt.

Der momentane State der Software ist nach außen transparent. Jedem State is quasi ein entsprechender Displayinhalt zugewiesen, der jedoch dynamisch hinsichtlich des momentanen Benutzers (RFID-Karte aufgelegt), des eingestellten Events und weiterer Faktoren gehalten ist.

Bevor die einzelnen States etwas genauer beschrieben werden werden, ist auf der nächsten Seite die State Machine grafisch abgebildet.

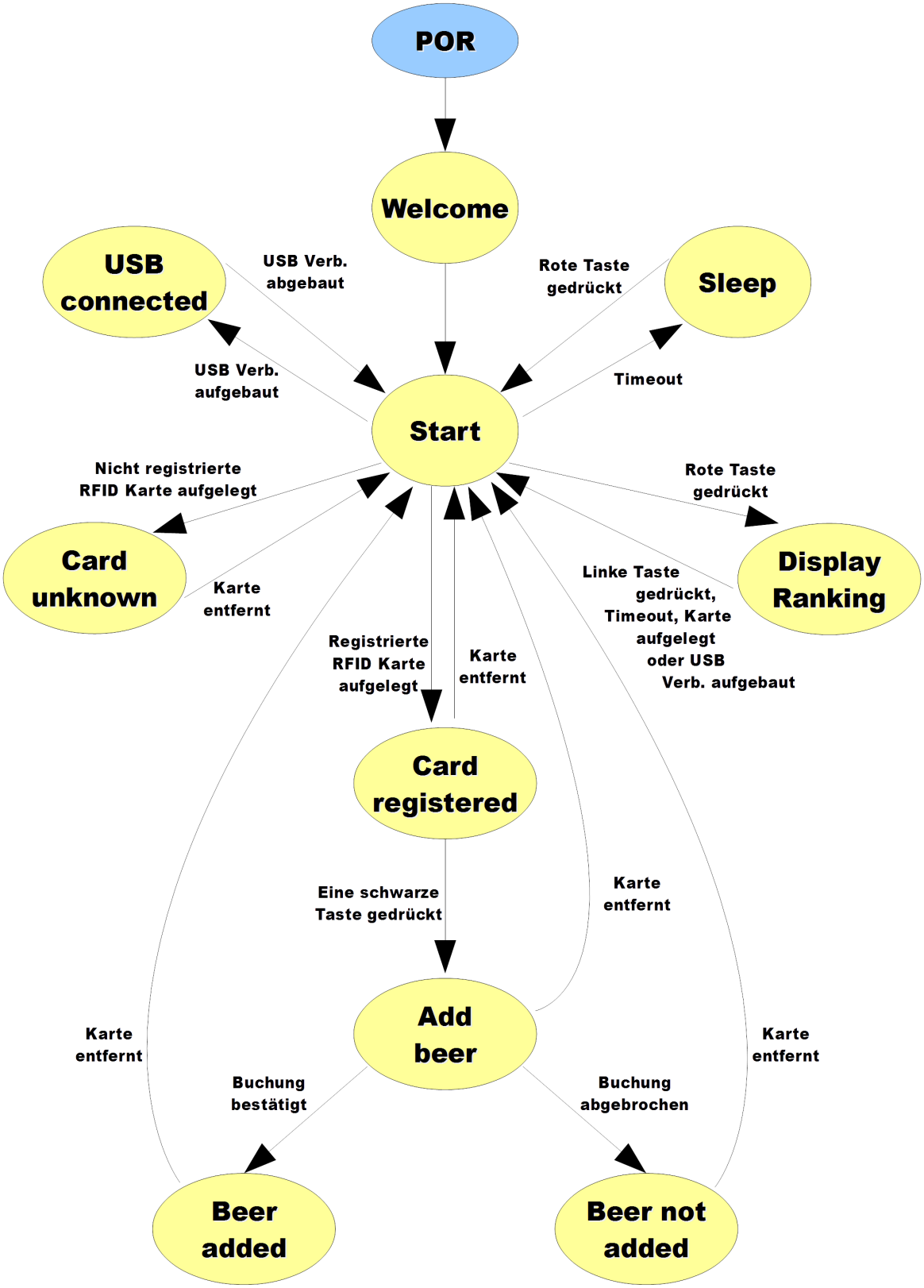

Abbildung 3.2.: State Machine als Herzstück der Funktionssteuerung der Software

State: Welcome

Dieser State dient nur der Information. Er zeigt dem Benutzer an, was er gerade in den Händen hält und wer da Arbeit reingesteckt hat.

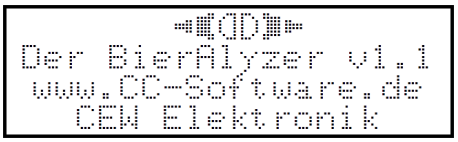

Abbildung 3.3.: Displayinhalt des States *Welcome*

State: Start

Dieser State ist der zentrale Ausgangspunkt. Das Programm checkt hier zyklisch, ob eine RFID-Karte aufgelegt, der rote Knopf (der unter denen im Display angezeigten Pfeilen) zur Anzeige des Trinkerratings gedrückt, oder eine USB Verbindung aufgebaut wurde.

Weiterhin wird Datum und Uhrzeit angezeigt, sowie die Bezeichnung des aktuellen Events.

Nach 15 Sekunden ohne jegliche Vorkommnisse wird aus Energiespargründen in den Sleep State gewechselt.

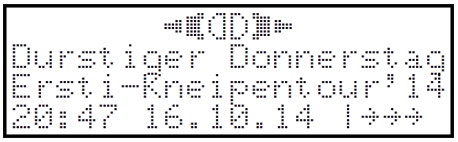

Abbildung 3.4.: Displayinhalt des States *Start*

State: USB Connected

In diesem State werden Pakete mit einem PC per USB ausgetauscht. Mehr dazu im Kapitel 4.

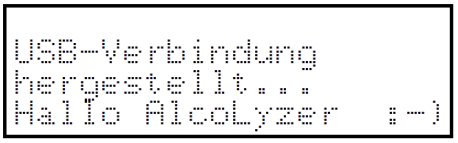

Abbildung 3.5.: Displayinhalt des States *USB Connected*

State: Sleep

In diesem State ist der Mikrocontroller schlafen gelegt, sämtliche Oszillatoren sind abgeschaltet, um minimalen Strombedarf zu garantieren. Weiterhin sind externe ICs im Standby und die Hintergrundbeleuchtung des LC-Displays ist aus.

Um den Mikrocontroller wieder zu aktivieren muss die rote Taste gedrückt werden, die einen asynchronen Interrupt auslöst.

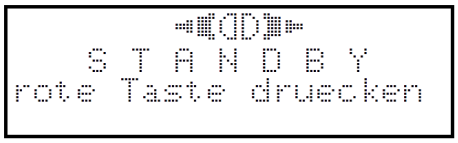

Abbildung 3.6.: Displayinhalt des States *Sleep*

State: Display Ranking

Beim Wechsel in diesen State wird ein Trinker Ranking berechnet. Dieser Vorgang dauert ein paar Sekunden, da der Insertion Sort-ähnliche Algorithmus zum Sortierend der Trinker anhand ihrer konsumierten Biermenge ein quadratisches Wachstum aufweist.

Nach der Berechnungszeit wird das Ranking mit Zahlenwerten und Balkendiagrammen dargestellt. Die maximale Balkenanzahl richtet sich nach der Biermenge, die die Liste anführt, auch wenn mittels der mittleren Tasten im Ranking auf- und abgeblättert wird.

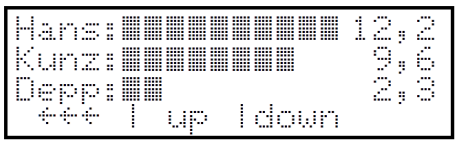

Abbildung 3.7.: Displayinhalt des States *Display Ranking*

State: Card Unknown

Wird eine RFID-Karte aufgelegt, dessen Seriennummer nicht im EEPROM registriert ist, so wird dieser State aktiv. Die entsprechende nicht registrierte Seriennummer wird angezeigt.

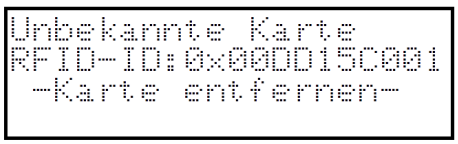

Abbildung 3.8.: Displayinhalt des States *Card Unknown*

State: Card Registered

Wenn die aufgelegte RFID-Karte intern registriert ist, wird der entsprechende Benutzer/Trinker mit seiner bereits erbrachten Trinkleistung angezeigt. Mittels der drei schwarzen Tasten kann jetzt der interne Bierzähler inkrementiert werden.

Der interne Zähler ist ein Byte breit, interpretiert wird in Deziliter. Sollte der Zähler über 25,5 Liter inkrementiert werden, läuft der Zähler über und beginnt bei 0. Dieser Fall wurde nicht abgefangen, da aus Sicht des Autors derjenige, der an einem Event so viel säuft, bestraft gehört!

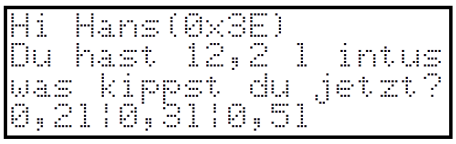

Abbildung 3.9.: Displayinhalt des States *Card Registered*

State: Add Beer

Hier wird eine Bestätigung zur Buchungsbeabsichtigung verlangt. Die Buchung kann auch durch Entfernen der Karte abgebrochen werden.

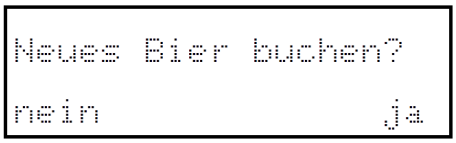

Abbildung 3.10.: Displayinhalt des States *Add Beer*

State: Beer Added

Nach einer Buchung wird so lange gewartet bis die RFID-Karte entfernt wird.

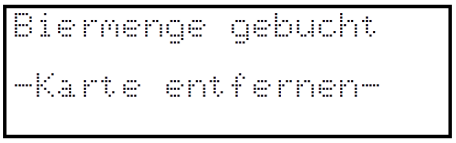

Abbildung 3.11.: Displayinhalt des States *Beer Added*

State: Beer Not Added

Nach Abbruch einer Buchung muss die RFID-Karte entfernt werden.

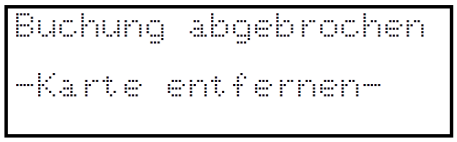

Abbildung 3.12.: Displayinhalt des States *Beer Not Added*

3.4. Peripheriebenutzung des Mikrocontrollers

Der verwendete Mikrocontroller besitzt einiges an interner Peripherie (bspw. Timer). Da das ein und selbe Peripherieelement gleichzeitig nur von einem Modul verwendet werden kann, muss diesbezüglich eine Einteilung und Zuweisung erfolgen.

(GPIO Ports wurden hier nicht als Peripherie deklariert.)

Peripheriename	**Verwendet von Modul**	**Zweckbeschreibung**
8-Bit Timer/Counter0	GPIO Periphery Driver	PWM zur Steuerung der Hintergrundbeleuchtung des LC-Displays. Entprellung der vier Taster. Automatisches Abschalten des Beepers nach voreingestellter Zeit.
16-Bit Timer/Counter1	RFID-Chip Driver	Zeitsteuerung der Dekodierung des Manchester-Kodierten Datensignals.
Extern INT4	Energy Management	Asynchroner Interrupt zum Aufwecken des Mikrocontrollers im Sleep Mode
Extern PCINT12	RFID-Chip Driver	Interruptgesteuerte Erfassung des Logikpegels des Manchester-Kodierten Datensignals.
USB	USB Raw-HID Driver	USB Kommunikation mit dem PC.

Tabelle 3.1.: Benutzung der mikrocontrollerinternen Peripherieelemente

4. USB Kommunikation

4.1. Kommunikation zwischen Mikrocontroller und PC

USB ist einfach und toll, so die Sichtweise des Computerbenutzers der sich zu früherer Zeit mit serieller COM und paralleler LPT Schnittstelle quälte.

USB ist verdammt kompliziert, so die Sichtweise des Programmierers der zu früherer Zeit die programmiertechnisch einfach zu benutzenden, bereits genannten Schnittstellen, genoß.

Da es dem Programmierer widerstrebte, das Rad neu zu erfinden, wurde beim Thema USB Kommunikation auf eine freie Bibliothek zurückgegriffen. Passend zum verwendeten AT90USB Mikrocontroller wurde die Teensy RAW-HID [3] Bibliothek verwendet. Da diese auf der USB-Klasse HID aufbaut, brauch PC-seitig kein Treiber für den BierAlyzer installiert zu werden.

DD Protokoll v1.0

Die hier beschriebene DD Protokollversion ist die v1.1. Die v1.0 wird aus Zeitrationalisierungsgründen nicht beschrieben. Ferner fand sie nur Verwendung in der BierAlyzer Firmware v1.0 und dem AlcoLyzer v1.0. v1.1 löst v1.0 komplett ab.

4.2. PC Programm AlcoLyzer

PC-seitig dient neben den Standard HID Bibliotheken, welche bereits in Windows enthalten sind, die Bibliothek AtUsbHid.dll der Kommunikation mit dem BierAlyzer. Mittels der Wrapper-DLL AtUsbHidJni.dll und der Java Native Interface Bibliothek AtUsbHidJni.jar wird es möglich, die HID Funktionen per Java zu benutzen. Alle Bibliotheken werden von Atmel bereit gestellt.

Die genaue Funktion des AlcoLyzers wird hier nicht beschrieben. Warum? Aus Zeitmangel. Ein an Häufigkeit exponentiell zunehmendes Phänomen in dieser schönen Welt.

Zum Zeitpunkt der Erstellung dieser Dokumentation existiert bereits ein Entwurf eines funktionell abgespeckten AlcoLyzers v1.1, Lite genannt.

Für die Zukunft ist eine nicht-Lite Version des AlcoLyzers v1.1 geplant, der automatisch mit dem Server auf der Homepage des Durstigen Donnerstags kommuniziert.

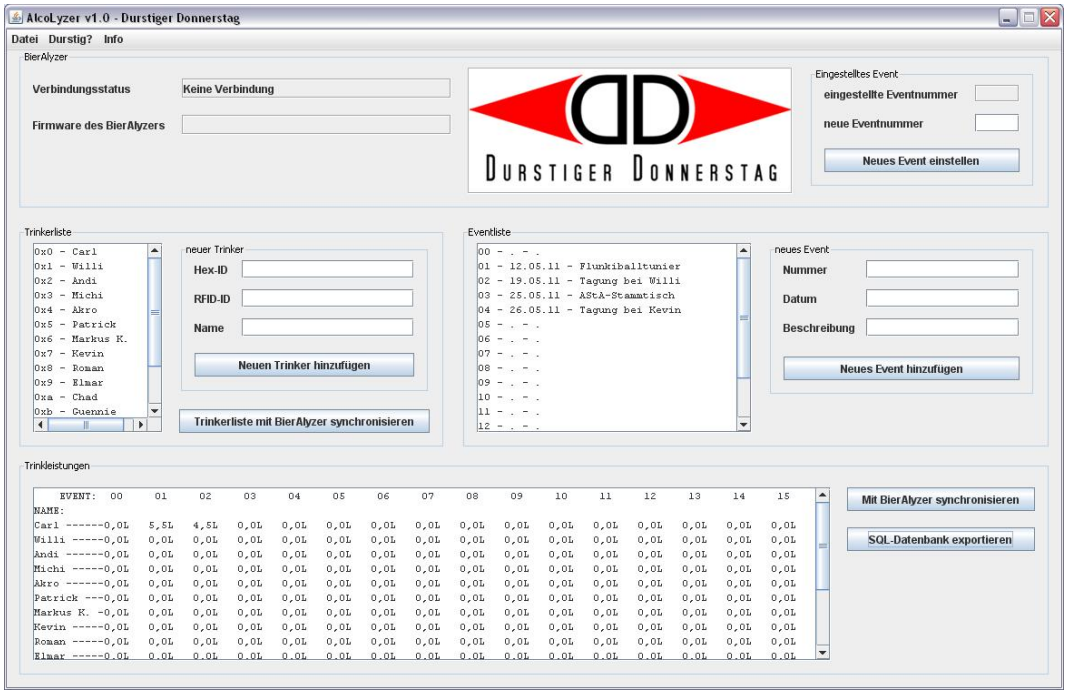

Abbildung 4.1.: AlcoLyzer v1.0, der nur mit der Firmware v1.0 des BierAlyzers läuft

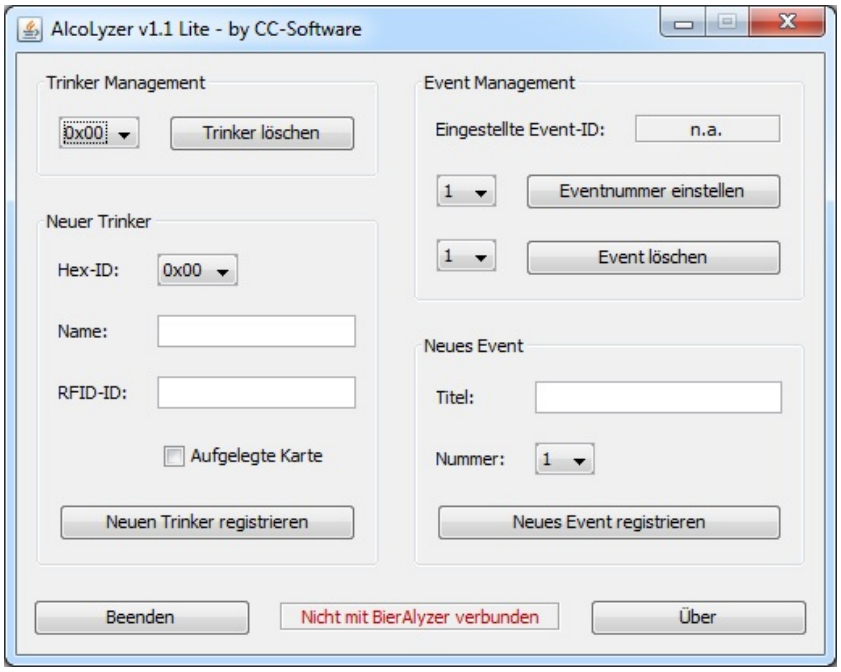

Abbildung 4.2.: GUI-Entwurf des neuen AlcoLyzers v1.1 Lite

4.3. Datenstruktur DD Protokoll v1.1

Aufgrund der verwendeten Bibliotheken und der HID Klasse haben die Datenpakete eine fixe Länge von 64 Byte. Diese sind gemäß DD Protokoll v1.1 folgendermaßen strukturiert:

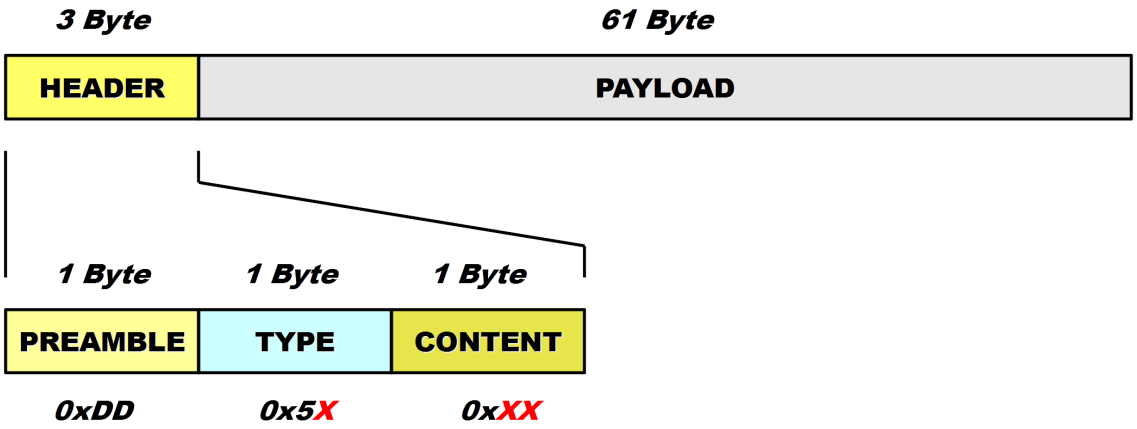

Abbildung 4.3.: USB Paketstruktur gemäß DD Protokoll v1.1

Die Bitmaske der Preamble ist immer `0xDD`.

Das Feld TYPE kann folgende Werte annehmen:

Wert	Bezeichnung	Beschreibung
0x51	Request	Datenanfrage an den BierAlyzer nach: • Usern • Events • Firmware Info • Einstellungen (Uhrzeit/Datum, momentane Eventnr., ...)
0x52	Data	Daten zur Übernahme im BierAlyzer: • (De)Registrierung neuen Users • (De)Registrierung neuen Events • Einstellungen (Uhrzeit/Datum, momentane Eventnr., ...)
0x54	Response	Antwort des BierAlyzers inkl. angefragter Daten auf erhaltene Request-Pakete.
0x55	Acknowledge	Antwort des BierAlyzers auf Data-Pakete, dass Daten übernommen wurden.
0x56	Not Acknowledge	Antwort des BierAlyzers auf Data-Pakete, dass Daten nicht übernommen wurden.

Tabelle 4.1.: Werte für das Feld TYPE des DD Protokolls v1.1

Das Feld CONTENT kann folgende Werte annehmen:

Wert	Bezeichnung	Beschreibung
0x63	User	Payload beinhaltet alle Daten eines Users/Trinkers
0x67	Event	Payload beinhaltet alle Daten eines Events
0x62	Settings	Payload beinhaltet Einstellungen wie: • Uhrzeit und Datum • Eventnummer • Firmware Version
0x70	Ping	Payload wird ignoriert. Dient der Überwachung der Verbindung.

Tabelle 4.2.: Werte für das Feld CONTENT des DD Protokolls v1.1

4.3.1. Payloadstruktur: Content = User

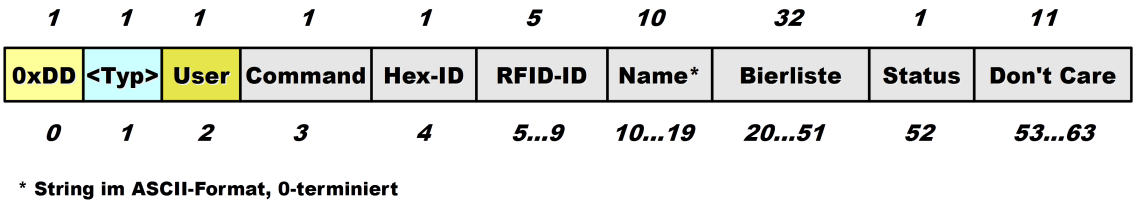

Abbildung 4.4.: Payloadstruktur eines *CONTENT=User*-Pakets

Mögliche Pakettypen:

- **Data**:
  Command = 0xDD - Paket dient zur Registrierung eines neuen Trinkers
  Command = 0x33 - Paket dient dem Löschen eines Trinkers
- **Request**:
  Command = 0xXX - Paket dient zur Datenabfrage eines Trinkers
- **Response**:
  Command = 0xXX - Paket als Antwort auf ein **Request** Paket

Hinweis: **0xXX** steht für nicht beachtetes Byte (don't care).

Das Feld **Status** beinhaltet den Status des Trinkers (Aktiv = 0x00, Inaktiv = 0x01).

4.3.2. Payloadstruktur: Content = Event

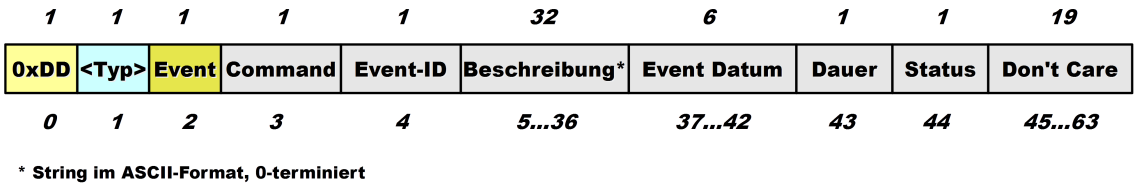

Abbildung 4.5.: Payloadstruktur eines *CONTENT=Event*-Pakets

Mögliche Pakettypen:

- **Data**:
  Command = **0xDD** - Paket dient zur Registrierung eines neuen Events
  Command = **0x33** - Paket dient dem Löschen eines Events
- **Request**:
  Command = **0xXX** - Paket dient zur Datenabfrage eines Events
- **Response**:
  Command = **0xXX** - Paket als Antwort auf ein **Request** Paket

Hinweis: **0xXX** steht für nicht beachtetes Byte (don't care).

Das Feld **Status** beinhaltet den Status des Events (Aktiv = 0x00, Inaktiv = 0x01).

4.3.3. Payloadstruktur: Content = Settings

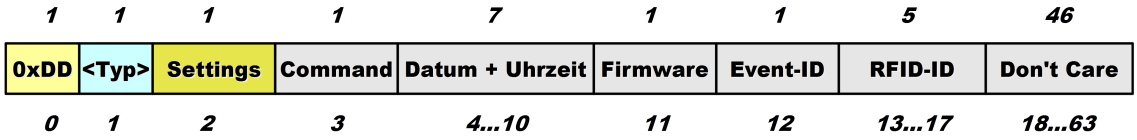

Abbildung 4.6.: Payloadstruktur eines *CONTENT=Settings*-Pakets

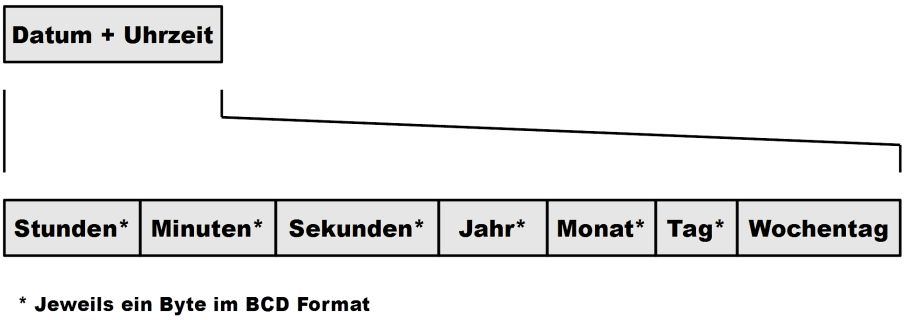

Abbildung 4.7.: Struktur von Datum/Uhrzeit innerhalb eines Settings-Paketes

Mögliche Pakettypen:

- **Data**:
  Command |= **0x01** - Paket dient zur des Datums und der Uhrzeit
  Command |= **0x02** - Paket dient dem Ändern der aktuellen Event-ID

- **Request**:
  Command = 0xXX - Paket dient zur Datenabfrage der Einstellungen / des Status'
- **Response**:
  Command = 0xXX - Paket als Antwort auf ein **Request** Paket

Hinweis: **0xXX** steht für nicht beachtetes Byte (don't care).

Das Feld **Firmware** enthält Informationen über die Firmware des BierAlyzers (`0bMMLLLLRR`):

- **MM**: Major Version
- **LLLL**: Minor Version
- **RR**: Release (0 = experimental, 1 = alpha, 2 = beta, 3 = stable)
- Firmware Version **v1.1 alpha** wird bspw. durch folgenden Wert repräsentiert: `0b01000101` = `0x45`.

Das Feld **Event-ID** enthält die eingestellte/einzustellende Event-ID, auf dem die gebuchten Biermengen gutgeschrieben werden.

Das Feld **RFID-ID** enthält die RFID-ID der momentan aufgelegten Karte. Falls keine Karte aufgelegt ist, haben alle 5 Bytes den Wert 0.

Im Feld **Wochentag** steht der Wert 0x01 für Montag, 0x02 für Dienstag usw.

4.4. Kommunikationsprinzip DD Protokoll v1.1

Das Kommunikationsprinzip ist einfach:

- Wird ein **Request** Paket an den BierAlyzer geschickt, so antwortet dieser mit einem **Response** Paket.
- Wird ein **Data** Paket an den BierAlyzer geschickt, so antwortet dieser mit einem **Acknowledge** Paket, wenn er die Daten übernommen hat. Hat er die Daten nicht übernommen (bspw. Registrierung eines Trinkers mit einer bereits besetzten Hex-ID), dann antwortet er mit einem **Not Acknowledge** Paket.

A. Quellen und Links

[1] AktionsBündnis Durstiger Donnerstag - Website
http://www.durstiger-donnerstag.de

[2] CC-Software
http://www.cc-software.de

[3] Teensy USB RAW-HID
http://www.pjrc.com/teensy/rawhid.html

[4] Stefan Seegel - RFID U2270B Treiber
dahamm@gmx.net

[5] microcontroller.net - Mikrocontroller Community
http://www.mikrocontroller.net

[6] CadSoft EAGLE Schaltplan- und Layouteditor
http://www.cadsoft.de/

[7] Atmel Studio 6
http://www.atmel.com/tools/ATMELSTUDIO.aspx

[8] WinAVR - Open Source Software Entwicklungswerkzeuge für AVR Controller
http://winavr.sourceforge.net/

[9] doxygen - Dokumentationssoftware
http://www.doxygen.org

B. Copyright

Alles hier beschriebene darf gerne nachgebaut und benutzt werden, ganz nach openSource Konzept.

Quelle und Autor und Durstiger Donnerstag müssen jedoch durchweg erwähnt werden!

Ferner dürfen jegliche Bestandteile nicht zu kommerziellen Zwecken verwendet werden!

Auf eine erfüllende Lebensweise während des Studiums - Prost!

Referat Technik - technik@durstiger-donnerstag.de

www.durstiger-donnerstag.de