HUMBOLDT-UNIVERSITÄT ZU BERLIN
MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT
INSTITUT FÜR INFORMATIK

Sicherheit in KNX-Netzen

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

eingereicht von: Jan Baudis
geboren am: 13.08.1984
geboren in: Berlin
Gutachter/innen: Prof. Dr. Jens-Peter Redlich
Prof. Dr.-Ing. Beate Meffert

eingereicht am: ..... verteidigt am: .....

Zusammenfassung

KNX ist ein etablierter, über 25 Jahre alter, Bus im Bereich der Gebäude- und Heimautomatisierung. In den letzten Jahren wurden zusätzliche Spezifikationen zur Verschlüsselung sowie zur Gewährleistung der Authentizität der Daten erarbeitet (u. a. KNX Data Security). Im Rahmen dieser Bachelorarbeit wurden unterschiedliche verfügbare Software- und Hardwarekomponenten analysiert mit dem Ziel, diese Spezifikationen zukünftig prüfen zu können sowie gegebenenfalls KNX-Komponenten damit nachzurüsten oder exponierte Strecken des Busses nachträglich abzusichern.

Ein erster Ansatz um dies zu ermöglichen (Proof of Concept(PoC)) wird vorgestellt. Zusätzliche Schwerpunkte sind ein einfacher sowie günstiger Nachbau und ein modulares und einfaches Design, wobei die Integration in bestehende KNX-Netze ebenfalls einfach und wenig invasiv gehalten werden soll. Die Analysen und Umsetzungen zur Hardware sind weitestgehend komplett, die Software erlaubt bisher jedoch nur ein Senden sowie Empfangen von Paketen. Beides könnte zukünftig, im Rahmen einer denkbaren Masterarbeit, erweitert werden.

Abschließend wird der Entwurf zu KNX Data Security aus dem Jahre 2013 tiefer gehend untersucht. Es wird gezeigt, dass der Entwurf fachliche sowie begriffliche Fehler aufweist. Ebenso wird eine erste mögliche Schwachstelle aufgezeigt. Die endgültige Spezifikation aus dem zweiten Quartal 2016 war zum Zeitpunkt dieser Arbeit nicht verfügbar.

Abstract

KNX is a well-established, 25 years old, bus used for home- and building automation. In the last few years specifications were added regarding the encryption and authenticity of the data (e.g. KNX Data Security). This bachelor thesis analyzes different available hardware and software components which could be (re-)used to test the specifications in the future and eventually could be used to upgrade existing KNX Components or to secure publicly accessible parts of the Bus.

A Proof of Concept to realize these goals will be shown. Additionally, the device is easy and cheap to build as well it is easy and sparsely invasive to existing KNX-Networks. The design is simple and modular. The basic Design is done. At the moment, only basic functions like receiving and sending a package are implemented. More features can be added in the distinct future as a part of a potential master thesis.

Finally, the „KNX Data Security“-draft written in 2013 is examined. The draft shows a lack of technical and terminological knowledge. There will be also an outline for a possible weakness. The final specification from the second quarter of 2016 was not yet available at the time when the thesis was written.

Inhalt

Einleitung

KNX ist ein Feldbus welcher zur Gebäude- sowie Heimautomatisierung genutzt wird und entstand als Weiterentwicklung aus dem Europäischem Installationsbus (EIB) [1]. KNX unterstützt unterschiedliche Übertragungswege und bietet eine geringe Datenrate (z. B. arbeitet KNX TP-1 mit zwei- bzw. vieradrigen Kabeln und einer Spannung von typischerweise 30 V. Die Datenrate beträgt 9,6 kbit/s [2]). Im weiteren Verlauf beziehen wir uns – sofern nicht anders genannt – stets auf KNX TP-1. Die Auswahl der Aktoren, Sensoren und weiteren verfügbaren Geräte ist zahlreich und weit gefächert. Alle Geräte, die von der KNX Association (KNXA) zertifiziert sind, sind miteinander kompatibel [3]. Die Spezifikationen sind überwiegend frei und seit Januar 2016 kostenlos verfügbar [4]. Die KNXA bezeichnet KNX als ersten weltweiten offenen Standard für Haus- und Gebäudesystemtechnik [3].

KNX bietet erst mit der Erweiterung KNX Data Security (teilweise auch mit dem übergeordnetem Begriff Data Secure bezeichnet) Möglichkeiten zur Authentifizierung und Verschlüsselung der Nutzdaten. Die Engineering-Tool-Software (ETS), welche der Programmierung der KNX-Geräte dient und von der KNXA herausgegeben wird, wurde am 12.04.2016 um die entsprechenden Funktionen erweitert [5]. Geräte mit KNX Data Security konnten bis dato (5. Februar 2017) jedoch nicht ausfindig gemacht werden.

Vorausgegangen sind mehrere Hacks und Kritik von Sicherheitsforschern [6] [7] [8]. KNX sah die Sicherheit als untergeordnet an und betonte stets, dass die Angreifer i.d.R. physischen Zugriff brauchen. Erst in den letzten Jahren wurde das Thema Sicherheit von der KNXA aktiv bearbeitet. Informatiker verschiedener Universitäten wurden jedoch bereits früher aktiv. Günter Westermeir von der TU München beschrieb und entwickelte bereits 2004 mit Secure EIB (SEIB) eine Lösung [9]. Im Jahr 2005 wurde von Wolfgang Granzer im Rahmen seiner Diplomarbeit an der TU Wien EIBSec entworfen [10]. Beide Lösungen beschreiben z. B. die Verschlüsselung der Nutzdaten sowie den Schutz der Datenauthentizität. Diese wurden jedoch nie von der KNXA in die Spezifikationen aufgenommen. KNX Data Security weist jedoch große Ähnlichkeit zu Secure EIB auf.

Um KNX (Data Security) zu analysieren und um z. B. exponierte (also einem Angreifer potenziell zugängliche) Kabelstrecken mit KNX Data Security zu schützen, wird im Rahmen dieser Bachelorarbeit ein KNX-Gateway/Interface skizziert, dessen Machbarkeit überprüft, gegen andere bereits vorhandene Geräte abgegrenzt und des Weiteren als Proof-of-Concept (PoC) umgesetzt. Ein wichtiges Ziel ist dabei der Betrieb über die Spannungsversorgung des Busses. Ebenso wird Wert auf eine einfachere Erweiterung der Hardware und Software gelegt, um weitere Funktionen umzusetzen. Die Möglichkeit von KNX im Klartext nach KNX Data Security zu „übersetzen“ sowie die Gateway-Funktionalität werden voraussichtlich in einer potenziellen Masterarbeit implementiert.

Vorbereitungen: Erfolgreich lässt sich der Bus abhören und es können Daten gesendet werden. Dies soll u. a. eine Analyse von KNX (Data Security) ermöglichen. Es gibt leicht unterschiedliche Möglichkeiten zum Aufbau. Diese sind überwiegend verhältnismäßig einfach und günstig, erfordern jedoch durchgehend Löt- und – zumindest rudimentäre – Mikrocontrollererfahrung. Der Quellcode soll öffentlich verfügbar gemacht werden.

Aufgabenstellung

Motivation und Ziel

KNX überträgt alle Daten im Klartext, selbst wenn es sich um drahtlose Kommunikation handelt (KNX RF). Dies ist aus Sicht der IT-Sicherheit ist dies kein akzeptabler Zustand. Im zweiten Quartal 2016 veröffentlichte die KNXA mit KNX Data Secure eine Spezifikation zur Verschlüsselung und Sicherung der Authentizität. Attraktiv erschien zunächst die Verwendung von KNX IP-Gateways u. a. zur Analyse von KNX gerade deswegen, weil die IP-Welt dem Informatiker sehr vertraut ist. Zum Absichern einer exponierten Strecke oder zur Realisierung z. B. eines Intrusion Detection Systems (IDS) ist der Preis jedoch hoch. Dies warf die Frage auf, ob es eine günstigere Alternative gibt bzw. realisierbar ist um mit dem KNX Bus zu kommunizieren.

Somit war es das Ziel ein Gerät (eine White Box) zu entwerfen, mit dem exponierte Busabschnitte mit KNX Data Security gesichert werden können und ein IDS o. ä. zu realisieren. Das Gerät sollte ein einfacher und günstiger Nachbau werden, welches eine gute Integrierbarkeit sowie eine lange Verfügbarkeit der Komponenten aufweist. Die vorliegende Bachelorarbeit schafft dafür die Grundlagen.

Ein weiterer Fokus lag auf der Erweiterbarkeit und Offenheit. So ist es grundsätzlich möglich, eine KNX RF-Schnittstelle bzw. ein KNX RF-Gateway durch wenig Aufwand ebenfalls zu ermöglichen, beispielsweise durch Verwendung des CC1101 von TI [11]. Dieser wird bereits für KNX RF Systeme eingesetzt. Die Verfügbarkeit ist (Stand Januar 2017) gut und durch die große Verbreitung dieses Chips wird er vermutlich lange verfügbar bleiben.

In den folgenden Kapiteln werden zunächst die Grundlagen von KNX besprochen. Darauf folgend werden bereits verfügbare Hardware- und Softwaresysteme betrachtet. Der primäre Fokus der Analyse liegt dabei auf der (Teil-)Verwendbarkeit. Es folgt die Vorstellung des PoC zur White Box sowie eine detaillierte Analyse des „KNX Data Security“-Entwurfs. In einem Fazit werden abschließend die Ergebnisse betrachtet.

Grundlagen

KNX-Grundlagen

Was ist KNX?

KNX ist ein Feldbus, der zur Gebäude- bzw. Heimautomatisierung verwendet wird. Dieser ist eine Weiterentwicklung des Europäischen Installationsbus (EIB). EIB wurde 1990 von der European Installation Bus Association (EIBA) entwickelt. Die ersten Geräte erschienen 1991 auf dem Markt [12]. KNX erschien als Weiterentwicklung aus den unterschiedlichen Bussystemen EIB, BatiBus und EHS. Häufig findet man Geräte, die als für EIB/KNX geeignet beschrieben und als solche gekennzeichnet sind. Dabei ist KNX TP-1 gemeint, welches kompatibel zu EIB ist [1].

Die federführende Organisation hinter KNX ist die KNX Association (KNXA), die sich aus ca. 370 Unternehmen zusammensetzt [3]. Diese akkreditiert z. B. die Testlabore, die die Zertifizierung der Geräte durchführen. Ebenso erstellt sie die Erweiterungen der KNX Spezifikationen.

KNX ist unter diversen Normen in einer Vielzahl von Ländern zugelassen und bezeichnet sich selber als ersten weltweiten offenen Standard für Haus- und Gebäudesystemtechnik [3]. KNX nutzt unterschiedliche Medien und Protokolle, wie Kabel und Funk. Bei kabelgebundenen Systemen gibt es unterschiedliche Spezifikationen zur Datenübertragung wie z. B. TP-0, TP-1 und Powerline. Ebenso gibt es die Möglichkeit, KNX über IP zu übertragen. Der Schwerpunkt dieser Bachelorarbeit liegt dabei auf der kabelgebundenen Übertragung nach der Spezifikation TP-1. Die Spannung beträgt auf dem Bus typischerweise 30 Volt. Dies liegt im Bereich für Sicherheitskleinspannungen (engl. Safety Extra Low Voltage (SELV) [13]).

Die Spezifikationen sind seit Januar 2016 kostenlos [4] erhältlich. Einschränkend sind jedoch nicht alle Spezifikationen in der aktuellen Version enthalten, z. B. ist KNX Data Security nur als Entwurf [14] in den Spezifikationen 2.1 verfügbar. Jedoch ist die Spezifikation seit dem zweiten Quartal 2016 fertig.

Die Aktoren, Sensoren etc. von KNX sind vergleichsweise teuer und die Datenrate ist mit 9600 Bit/s gering. Die maximale Kabellänge ist im Vergleich zu anderen Heim- und Gebäudeautomatisierungsbussen hoch. Erlaubt ist ein Abstand von 700 Metern zwischen zwei physischen Geräten in einem Segment [15 S. 6]. Die Maximale Kabellänge eines Segmentes ist auf 1000 Meter begrenzt. Topologisch sind mehrere Formen bzw. Mischungen folgender Formen erlaubt: Stern- sowie baumförmig und linear. KNX-Installationen können im Gesamten mehrere Kilometer groß sein.

KNX bietet dabei ein ähnliches Adressierungsschema wie IPv4. Falls ein einzelnes Gerät angesprochen werden soll, wird nach Gebiet (Area), Linie (Line) und Gerät (Device) adressiert. Eine gültige Adresse ist beispielsweise die 1.1.4. Die Adressen von Area und Line sind dabei jeweils vier Bit und die vom Device acht Bit groß [15]. Um eine Gruppe von Geräten anzusprechen, können Gruppenadressen definiert werden. Diese sind ähnliche aufgebaut, wobei die Begrifflichkeiten und die Größe des Adressraums geringfügig anders definiert sind.

Die Einsatzgebiete von KNX sind mindestens ebenso zahlreich wie die von anderen Automatisierungsbussen und -systemen. Die Anzahl der unterschiedlichen Gerätegruppen liegt zurzeit bei über 7000, welche von weltweit über 370 Herstellerfirmen angeboten werden [3]. Die Zertifizierung soll dabei gewährleisten, dass alle Geräte zueinander kompatibel sind. Ursprünglich zur

Gebäudeautomatisierung gedacht, fand KNX jedoch in den letzten Jahren zunehmend ebenfalls Verbreitung im Bereich der Heimautomatisierung.

Grundlagen des KNX-Busses nach TP-1

Grundlagen der Bitübertragungsschicht

TP-1 ist eine Spezifikation von KNX zur kabelgebundenen Übertragung der Daten. Diese wurde als eine der ersten entworfen und verabschiedet. Sie lässt sich in fast jeder KNX-Installation antreffen. Aufgrund dessen wird im Rahmen dieser Bachelorarbeit der weitere Fokus auf TP-1 liegen. Im Folgenden werden daher nur die Grundlagen für KNX TP-1 umrissen. KNX TP-0, KNX RF etc. werden hier nicht behandelt. Die Definitionen der Datenpakete sind jedoch, unabhängig vom Medium, in großen Teilen identisch.

Als physikalisches Medium werden i.d.R. Kupferkabel mit zwei bzw. vier Adern verwendet, wobei letztere die gängigste Variante ist. Häufig werden bei vieradrigen Kabeln nur zwei Adern für den Bus verwendet, das zweite Paar wird entweder nicht verwendet oder bietet eine weitere Spannungsversorgung welche meist ähnlich der Busspannung ist. Die Busspannung ist eine Gleichspannung, die typischerweise bei 30 V lieg. Laut Spezifikation sind Spannungen zwischen 21 und 32 V erlaubt [15 S. 9]. Ein Gerät darf dabei bis zu 12 mA aus dem Bus beziehen. Die Datenrate beträgt dabei 9600 bit/s und ist zeichenorientiert. Es wird pro Zeiteinheit (typischerweise 104 µs) nur 1 Bit codiert, woraus die Datenrate folgt. Der normale Zustand definiert dabei die Binärzahl „1“ [15 S. 8]. Die Spezifikation sieht vor, dass dieser in diesem Zustand nicht mehr als 0,3 V ansteigt bzw. nicht mehr als um 2 V fällt. Ein größerer Spannungsabfall könnte ggfs. als eine binär codierte „0“ gewertet werden. Die Charakteristika sind in Tabelle 1 aufgeführt.

Characteristics	Description TP1-64	Description TP1-256
Medium	shielded Twisted Pair	shielded Twisted Pair
Topology	linear, star, tree or mixed	Linear, star, tree or mixed
Baud rate	9600 bps	9600 bps
device supplying	normal: bus powered devices optional: remote powered devices	normal: bus powered devices optional: remote powered devices
device power consumption	3 mA to-12 mA	3 mA to-12 mA
Power Supply Unit (PSU)	DC 30 V	DC 30 V
Number of PSU’s per physical Segment	max. 2	max. 2
Number of connectable devices per physical Segment	max. 64	max. 256
Number of addressable devices per physical Segment	max. 255	max. 255
Total cable length per physical Segment	max. 1000 m	max. 1000 m
distance between two devices	max. 700 m	max. 700 m
Total number of devices in a network	over 65 000 (with Bridges)	over 65 000
Protection against shock	SELV (Safety Extra Low Voltage)	SELV (Safety Extra Low Voltage)
Physical signal	balanced baseband signal encoding	balanced baseband signal encoding

Tabelle 1 - Übersicht der Charakteristika von TP-1 [15 S. 6]

Als Binär „0“ wird ein Spannungsabfall (maximal um 10,5 V) in den zeitlichen Grenzen von 25 µs und 70 µs (ts) – typischerweise 35 µs (t_{active}) – gewertet. Nach dem Spannungsabfall folgt ein Spannungsanstieg, der maximal 69 µs ($t_{equalisation}$) andauert. Die Spannungsabweichung nach 104 µs darf maximal nur -0,35 V bzw. +1,8 V relativ zu der ursprünglichen Spannung betragen (Uend). In Abbildung 1 wird diese Spannungsänderung über die Zeit (t) grafisch dargestellt. Die Y-Achse beschreibt dabei die Spannung (U).

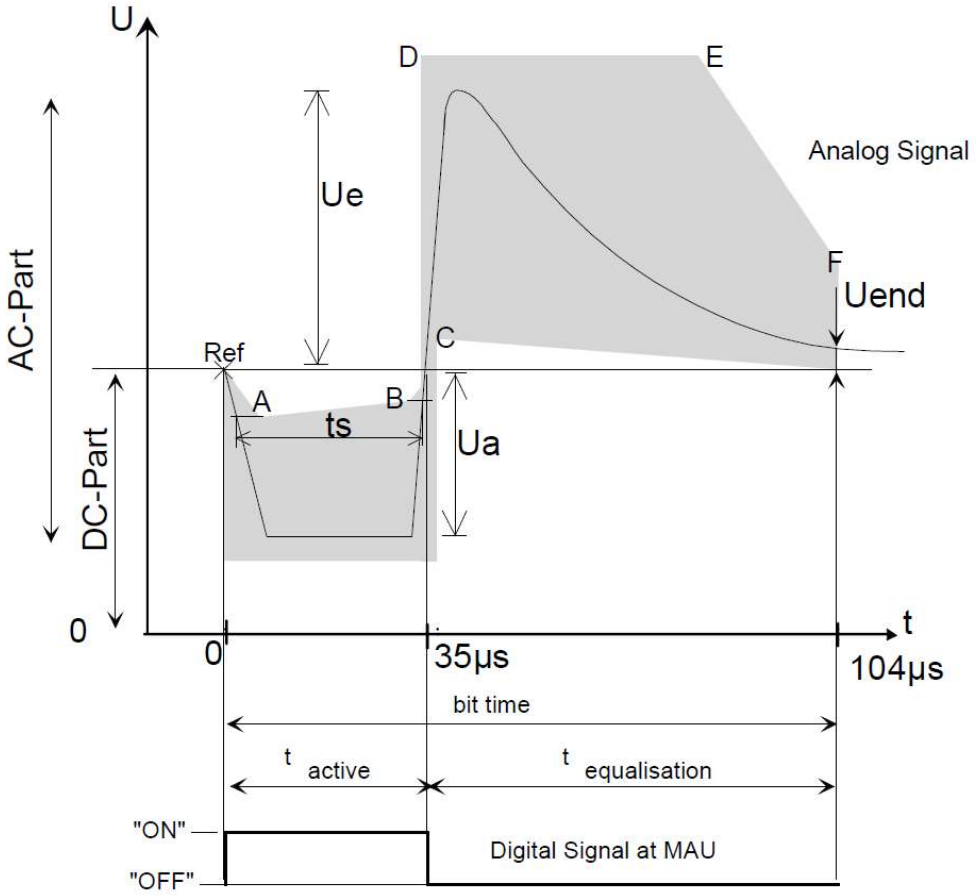


Abbildung 1 - "0"-Bit Frame [15 S. 9]

Wenn eine Gruppe von Geräten angesprochen wird, ist es teilweise gewünscht, dass alle Geräte (z. B. bei ACKs) gleichzeitig antworten, wodurch sich t_{active} entsprechend innerhalb der Grenzen von ts verlängert. Ansonsten ist eine Kollisionserkennung nach CSMA/CA [15 S. 12] definiert. Für die Übertragung eines Bytes sind weitere Spannungsgrenzen definiert [15 S. 11].

Zur Erkennung für den Empfänger, dass ein Datenbyte gesendet wird, wird eine „0“ auf den Bus codiert. Danach folgen die Datenbits D0-D7 sowie ein Paritäts- und ein Stoppbit. Falls mehrere Byte übertragen werden sollen (dies entspricht dem Normalfall) werden zwischen dem Stoppbit und dem folgenden Startbit 204 µs keine Spannungsveränderung erzeugt (dies entspricht der benötigten Zeit zur Übertragung von zwei Bit) [15 S. 17-18]. Zur Verdeutlichung stellt die Abbildung 2 die Kodierung der digitalen Daten als analoges Signal sowie den Ablauf der Übertragung eines Bytes dar. Die Y-Achse stellt im obersten Koordinatensystem die Spannung dar, im zweiten wird der Zustand der Sendeeinheit beschrieben (ob an („On“) oder aus („Off“)). Im letzten werden die dazugehörigen codierten Bit dargestellt. Die X-Achsen beschreiben die Zeit bzw. das n-te Bit.

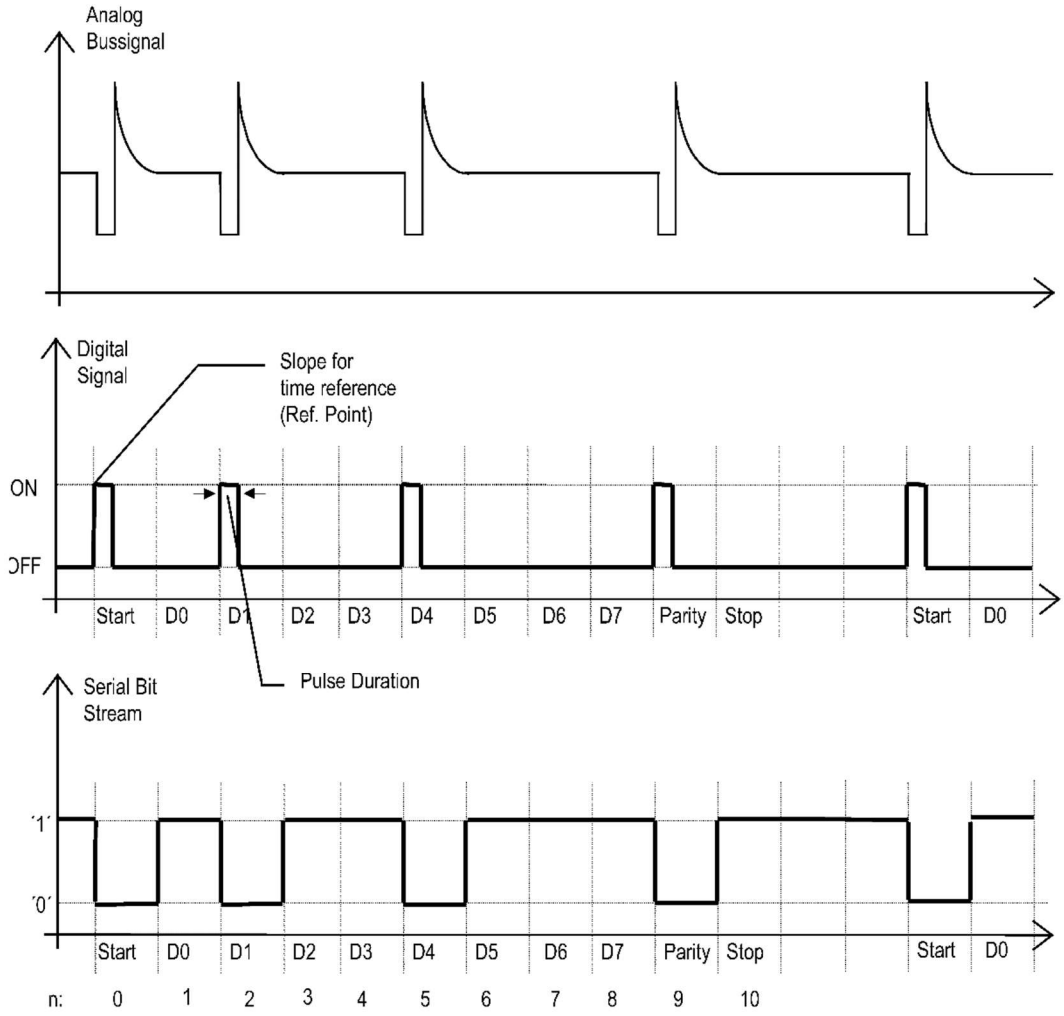


Abbildung 2 – Signalcodierung [15 S. 17]

Durch diese Spannungsschwankungen werden gewisse Ansprüche an die Busteilnehmer zwingend und ebenfalls muss der technische Aufbau diese Spannungseinbrüche überhaupt ermöglichen. Ein KNX-Netzteil enthält dazu ein spezielles Bauteil (im Weiteren KNX-Drossel genannt) auf der Busleitung[1]. Des Weiteren müssen zu große Leistungsschwankungen der Busteilnehmer vermieden werden. Die Timings und die Belastung des Busses z. B. beim Ein- bzw. Ausschalten sind dabei genau definiert [15 S. 12-16]. So darf ein Gerät beim Einschalten beispielsweise nur maximal 20 ms den Bus stören, um zu verhindern, dass Telegramme verloren gehen [15 S. 13]. Damit sich Störungen nicht auf die komplette KNX-Installation auswirken, existiert durch Bridges oder Router eine galvanische Trennung zwischen den Linien.

[1] Die KNX Drossel ist ein elektronisches Bauteil welches einen Einbruch der Spannung erlaubt. Detaillierte Erläuterungen können dem Anhang A oder dem Patent zu einen kommerziellem KNX Netzteil [54] entnommen werden.

Nachdem im vorherigen Abschnitt die Bitübertragungsschicht besprochen wurde, kommen wir nun zur Sicherungsschicht. Es werden nur die für diese Bachelorarbeit wichtigsten Prinzipien und Datenformate angesprochen. Die hier besprochenen und weiteren Definitionen sind umfangreich in den Spezifikationen zu finden [15 S. 27-37]. Die Datenübertragung beginnt stets beim Oktett 0 und gesendet wird nach dem Prinzip LSB First. Für KNX TP-1 sind dabei drei unterschiedliche Datenframes definiert – „L_Data“, „L_Poll_Data“ sowie „Acknowledgment“ (ACK). Abbildung 3 zeigt ein typisches KNX-Datenpaket nach dem „L_Data Frame“-Format, welches wohl – abgesehen von Bestätigungspaketen – am häufigsten anzutreffen ist. Ein „L_Data Frame“ existiert dabei in zwei unterschiedlichen Varianten. Entweder als „L_Data_Standard Frame“ oder als „L_Data_Extended Frame“. Das „L_Data_Standard Frame“-Format ist dabei auf fast 14 Bytes Benutzerdaten begrenzt, wohingegen das erweiterte Format bis zu 254 Bytes enthalten kann [15 S. 27].

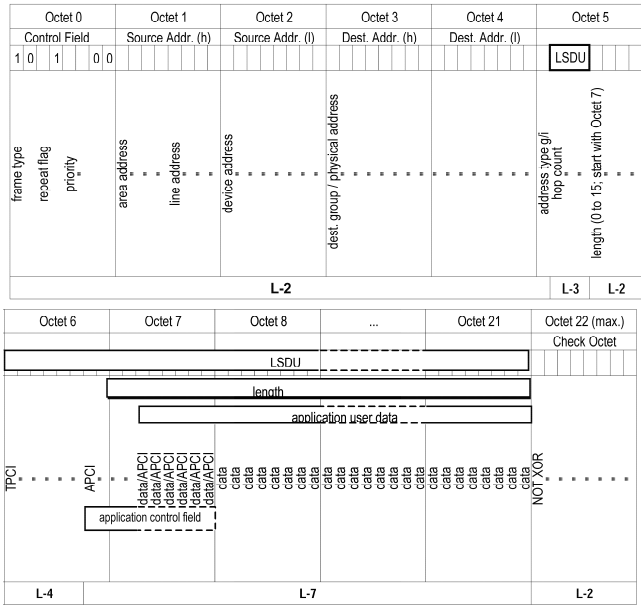


Abbildung 3 – „L_Data_Standard Frame“-Format [15 S. 28]

Für jedes Datenpaket ist das erste Oktett als „control field“ definiert. In diesem wird festgelegt, um welchen Frametypen es sich handelt, was über die Bits 6 und 7 definiert wird. Bit 3 und 4 definieren die Priorität und Bit 5 legt fest, ob dieses Datenpaket erneut gesendet wurde.

Exemplarisch wird folgend kurz das „L_Data_Standard Frame“-Format angesprochen. Die anderen Formate sind, wie bereits erwähnt, in der Spezifikation [15 S. 27-37] zu finden. Die Oktette 1 und 2 enthalten die Quelladresse, wohingegen die Oktette 3 und 4 die Zieladresse enthalten. Das siebte Bit in Oktett 5 legt dabei den Adresstypen der Zieladresse fest. KNX TP-1 kennt individuelle und Gruppenadressen. Zu beachten ist, dass es im KNX-Netzwerk jedoch auch Geräte ohne Adresse geben kann, z. B. besitzen Bridges i.d.R. keine Individuelle Adresse, bestätigen jedoch den Empfang eines Datenpaketes durch ein ACK [15 S. 21]. Router besitzen eine individuelle Adresse sowie Kenntnis über die Geräte, die in den Subnetzwerken vorhanden sind. Diese Kenntnisse nutzt der Router, um bestimmte Pakete zu filtern oder weiterzuleiten [16 S. 30].

In den weiteren Oktetten folgen Daten u. a. über die Information der Größe der Nutzdaten, ein Hop Zähler sowie natürlich die Nutzdaten selber. Der Hop Zähler wird dabei genutzt um zu erkennen von wie vielen Kopplern (Router und Bridges) ein Datenpaket bereits weitergeleitet wurde, wobei KNX

TP-1 maximal 6 Koppler zwischen 2 Geräten vorsieht [15 S. 22] – ggfs. wird das Datenpaket verworfen[2]. Das letzte Oktett hilft der Erkennung von Übertragungsfehlern. Dieses enthält die Paritätsdaten (ungerade Parität), welche über die vorherigen Oktette gebildet werden.

Abhängig vom Aufbau des KNX-Netzwerkes kann mindestens jedes Gerät einer Linie die Daten auf dem Bus lesen. Jedes dieser Geräte prüft nun, ob es als Empfänger gilt. Welche Fälle das sind ist unter [15 S. 38] beschrieben. Daraufhin wird durch den bzw. die Empfänger das Datenpaket auf Bitfehler geprüft. Der Sender wird i.d.R. durch ein ACK, NACK oder BUSY-Frame informiert. Es existieren jedoch auch Pakete, die keine Bestätigung z.B. durch ein ACK verlangen. BUSY signalisiert dabei gegenüber dem Sender, dass zurzeit keine Ressourcen verfügbar sind. Alle anderen nicht adressierten Geräte ignorieren das Datenpaket.

In den Systemspezifikationen sind u. a. der Ablauf und weitere Dienste gegenüber den höheren Schichten genau beschrieben. Von besonderem Interesse für diese Bachelorarbeit ist abgesehen von den „L_Data“ Diensten, der „L_Busmon“ Dienst, der es ermöglicht, alle Datenpakete auf dem Bus mitzulesen. Wenn dieser Dienst verwendet wird, muss der Receiver/Transceiver sich zwingend in dem Monitor Modus befinden [15 S. 41]. In diesem Modus ist der Receiver/Transceiver rein passiv und sendet niemals Daten in Richtung des Busses. Des Weiteren werden selbst fehlerhaft empfange Daten weitergereicht.

Auf höhere Schichten und insbesondere auf den detaillierten Inhalt auf Applikationsebene wird im Rahmen dieser Bachelorarbeit nicht eingegangen. Im Rahmen dieser Arbeit sind dort jedoch zum einen die Datapoint Types (DPTs) von Interesse sowie zum anderen das „External Message Interface“ (EMI) bzw. „common EMI“ (cEMI). DPTs definieren die Interpretation von Teilen der Nutzdaten, beispielsweise als Gleitkommazahl oder als ASCII-Zeichen [17]. EMI und cEMI sind besonders von Interesse, falls ein KNX Entwickler eine Bus Coupling Unit (BCU) [18] verwendet. Oder verschiedene Medien oder Protokolle verwendet werden. Eine TP1 BCU1 beispielsweise enthält einen KNX-Transceiver sowie einen Mikrocontroller [18 S. 7] und Teile des KNX-Stacks [18 S. 8]. EMI wird dabei zur Kommunikation mit der BCU verwendet, cEMI hingegen z. B. beim Einsatz eines KNX nach IP-Gateways. In der Software ETS werden bei aktiviertem Busmonitor-Mode mit einem KNX nach IP-Gateway cEMI-Pakete ausgegeben.

[2] Die System Spezifikation ist hier nicht eindeutig, siehe [15 S. 22] und [16 S. 30].

Datenverschlüsselung sowie Sicherung der Authentizität

Nach den Grundlagen der Datenübertragung von TP-1 wird hier kurz die Sicherheit bzw. die Authentizität der per KNX übermittelten Daten betrachtet. Zu Beginn von EIB spielte diese eine eher untergeordnete Rolle, wie es Mitte/Ende der 1980er durchaus üblich war. Bei der späteren Weiterentwicklung zu KNX Anfang der 1990er wurde an dieser Auffassung nichts geändert. Damals übertrug EIB bzw. KNX die Daten ohne Schutz der Authentizität oder Verschlüsselung. Erst ungefähr zehn Jahre später widmeten sich erste Universitäten dem Thema Sicherheit in KNX/EIB-Netzwerken [10] [9]. Besonders intensiv beschäftigte sich die Automation Systems Group der TU Wien mit dem Thema [19]. Diese erkannte früh, welche Implikationen die fehlende Verschlüsselung hatten und entwarf bereits Möglichkeiten wie Verschlüsselung in den KNX-Bus Einzug erhalten könnte. Ein Beispiel ist das von Wolfgang Granzer in seiner Diplomarbeit entworfene EIBSec aus dem Jahre 2005.

Die KNXA nahm diese Entwürfe zwar dankend und honorierend an, änderte jedoch nichts an den Spezifikationen und betonte stets das KNX sicher sei. Ein möglicher Grund, dass Entwürfe wie z. B. EIBSec nie den Weg in die Spezifikationen gefunden hat, war mutmaßlich der damit einhergehenden großen Änderungen und der ebenfalls damit einhergehenden Inkompatibilität zu bestehenden Geräten geschuldet. Die KNXA betonte regelmäßig, dass ein Angreifer physikalischen Zugriff zum System bräuchte. Anzumerken ist, dass zu dieser Zeit jedoch schon IP-Gateways verfügbar und im Einsatz waren. Anfang der 2010er Jahre gerieten KNX Systeme dann zunehmend negativ in die Schlagzeilen [8] [6]. Die KNXA verwies weiterhin öffentlich auf den benötigten physikalischen Zugriff und zeigte in einem Fall Fehler des Installateurs auf. Dieser hatte ein KNX nach IP Gateway in einem für Hotelgäste erreichbarem WLAN nicht abgesichert. Als weitere Reaktion veröffentlichte die KNXA eine Sicherheitscheckliste [20] sowie ein Positionspapier [21].

Intern arbeitete die KNXA seit 2013 an der Spezifikationserweiterung KNX Data Security, die einen Schutz der Authentizität und Verschlüsselung für die Nutzdaten von KNX TP-1 bringt. Die endgültige Spezifikation firmiert unter KNX (Data) Secure[3] [22] und beinhaltet ebenfalls eine Spezifikation zur verschlüsselten Übertragung von KNX Daten über IP.

[3] KNX Secure sowie KNX Data Secure werden häufig synonym verwendet. Die Verwendung der Begriffe durch die KNXA ist in diesem Punkt inkonsistent.

KNX-Gateways/Interfaces

In diesem Kapitel werden die unterschiedlichen KNX-Gateways sowie Interfaces vorgestellt und ihre (Teil-)Wiederverwendung diskutiert. Es wird jeweils die Hard- und Software getrennt betrachtet. Die Übersicht ist nicht abschließend.

Vorworte zur Analyse der Hardware

Zur Analyse des KNX-Busses bietet es sich an, die Daten auf einer möglichst niedrigen Schicht zu betrachten und zusätzlich am PC auswerten zu können. Der häufig für die Kommunikation mit dem KNX Bus verwendete Transceiver TPUART (2,2+) kümmert sich dabei i.d.R. um die Schichten eins und zwei. Dieser ist zum Zeitpunkt dieser Arbeit gut erhältlich und weit verbreitet. Er ist somit prädestiniert für die Aufgabenstellung.

Nach Möglichkeit soll auf bereits Bestehendes aufgebaut werden. Somit werden in diesem Kapitel die unterschiedlichen, mir bekannten und frei verfügbaren, Hardwareentwürfe analysiert und vorgestellt. Es wird ein möglichst einfach nachzubauender Entwurf gesucht. Alternativ wird ein für die Aufgabenstellung einfach portierbarer Entwurf gesucht (in diesem Fall auf ein ATXMEGA – die Diskussion dazu ist unter Vorüberlegungen zu finden).

Die Auswahl an Hardwaredesigns ist zahlreich. Die Anbindung an den KNX-Bus ist in der Mehrzahl durch den TPUART bzw. dessen Nachfolger TPUART2(+) realisiert. Die Anbindung basiert überwiegend auf dem im Datenblatt abgebildeten Referenzdesign. Der Mikrocontroller zur weiteren Verarbeitung dahinter variiert. Teilweise werden Wandler bzw. Gateways eingesetzt, um andere Protokolle und Schnittstellen zur weiteren Verarbeitung verwenden zu können, wie z. B. UART nach USB Wandler. Die TU Wien hat innerhalb der Automation Systems Group in diesem Bereich bereits einiges vorzuweisen und sticht mit ihrem Board namens KNXCalibur heraus.

Es gibt viele weitere, den hier vorgestellten sehr ähnliche, Projekte. Gemein haben – wie bereits erwähnt – fast alle Projekte den Transceiver Chip TPUART (2,2+) von Siemens. Nicht frei erhältliche Produkte werden im Rahmen dieser Arbeit nicht betrachtet. Des Weiteren sind von den hier vorgestellten Möglichkeiten nur das KNX IP Gateway und die KNX USB-Schnittstelle von der KNXA zertifiziert.

Vorworte zur Analyse der Software

Die Analyse der Software stellt sich als deutlich schwieriger dar. Im Folgenden werden nun ebenfalls die verfügbaren Softwarelösungen untersucht. Die Unterschiede sind teils enorm und den unterschiedlichen Anforderungen geschuldet. Häufig ist die Software für eine bestimmte Hardware ausgelegt, wobei fast alle die Verwendung des TPUARTs gemein haben.

Bei der Software wird (fast) nur frei verfügbare betrachtet. Viele der hier vorgestellten Softwarelösungen sind für genau einen Hardwareentwurf konzipiert worden – z. B. KNXCalibur.

Bei der Analyse und Bewertung sind u. a. die Dokumentation, die Portierbarkeit auf einen ATXMEGA (die Wahl des ATXMEGA wird später unter den Vorüberlegungen diskutiert) – und damit einhergehend die Programmiersprache – sowie die Qualität des Quelltextes von besonderem Interesse.

KNXCalibur

Hardware

Das bereits angesprochene Board KNXCalibur wird leider nicht mehr gepflegt. KNXCalibur ist ein von der TU Wien vor über zehn Jahren entworfenes, sehr vielseitiges und damit mächtiges Board mit zahlreichen Schnittstellen z. B. nach USB, Ethernet und natürlich KNX. Es kann durch ein Zusatzboard erweitert werden. Mit diesem ist z. B. der Zugriff auf DCF77, Feuchtigkeits-, Licht- und weiteren Sensoren möglich ist [23].

Das Hardwaredesign sowie die Software sind öffentlich verfügbar [24]. Die Hardware ist jedoch schwer zu beschaffen. Z. B. ist der verwendete Mikrocontroller ein Fujitsu MB90330 (dieser war zum Zeitpunkt dieser Bachelorarbeit nicht mehr auffindbar bzw. bestellbar). Das Board ist aufgrund der komplexen Platine und den überwiegend verwendeten SMD Bauteilen nicht einfach für den Nachbau geeignet. Die Webseite zu KNXCalibur ist während der Erstellung dieser Arbeit von den Servern der TU Wien verschwunden. Das Hardwaredesign und die Firmware sind jedoch – zumindest noch – verfügbar [19]. Zur Anbindung des KNX-Busses wird der TPUART [25] von Siemens verwendet.

Durch die nicht gegebene Verfügbarkeit des Mikrocontrollers und dem auf SMD Bauteilen ausgelegtem Design disqualifiziert sich dieses Board, selbst in Teilen, für unsere Zwecke.

Software

Die Software wurde in C geschrieben und verfolgt einen modularen Ansatz. Der Fokus lag auf Plattformunabhängigkeit. Da KNXCalibur weitere Hardware wie z. B. einen Ethernetcontroller oder mit dem Erweiterungsboard auch z. B. DCF77 unterstützt, wurde hier nur der Code betrachtet welcher direkt der Kommunikation mit dem KNX-Bus dient.

Wie bereits erwähnt ist der Mikrocontroller nicht mehr erhältlich. Die Mikrocontroller mit ähnlichem Design von Fujitsu sind schwer zu beschaffen und vergleichsweise teuer. Der Portierungsaufwand auf einen ATXMEGA von Atmel, ist nach Einsicht des mikrocontrollerspezifischen Designs sehr hoch. Des Weiteren sind z. B. die Interrupt Service Routinen (ISR) lang und beinhalten größere Ausgaben zum TPUART sowie dem UART-Interface zum PC – ISRs sollten jedoch möglichst kurz sein. Insbesondere die Verwendung der UART-Schnittstelle in ISRs ist nicht empfehlenswert und kann zu Problemen führen. Hardware UART-Schnittstellen lösen meist selbst, z. B. beim Empfangen, Interrupts aus. Der fehlenden Verwendung des DEBUG Makros ist es geschuldet [19], dass der Debug-Code überwiegend mit kompiliert und dementsprechend auch ausgeführt wird.

Der Quellcode enthält zwar Keywords, die auf eine Verwendung von Doxygen [26] zur Dokumentation hinweisen. Jedoch ist weder eine von Doxygen generierte Dokumentation dabei noch sind Keywords durchgehend verwendet worden.

Der Quellcode oder Teile davon disqualifizieren sich für die Weiterverwendung. Die Implementierung weist an vielen Stellen Lücken auf oder ist extrem lang. Ebenfalls ist viel toter Code vorhanden. Selbst der als hardwareunabhängig gekennzeichnete Teil ist in der vorliegenden Form nicht verwendbar. Da dieser in großen Teilen nicht hardwareunabhängig ist. Hinzu kommen die bereits angesprochenen langen ISRs. Ebenso wurden große Teile der Software unter keine Lizenz gestellt, was die Verwendung durch dritte erschwert. Zusätzlich wird die Software nicht mehr gepflegt. Die Webseiten zu KNXCalibur sind größtenteils nicht mehr verfügbar.

PiGator plus Verwendung des TPUART-Moduls

Hardware

PiGator ist eine Trägerplatine für den Kleinstcomputer Raspberry PI (RPI). Diese beinhaltet z. B. eine Real Time Clock (RTC) und einen Anschluss für OneWire. Sie ist in 2 Bauformen erhältlich [27] [28]. Entwickelt und verkauft werden diese Module durch Busware. Sie können auf den 26/40-pin GPIO Header [29] des RPI gesteckt werden und um z. B. ein PiGator Modul erweitert werden. Für diese Arbeit ist das EIB/KNX Modul [30] von Interesse. Die Platine verwendet den uns bereits bekannten TPUART von Siemens. Das Design ist – wie zu erwarten – stark an das Referenzdesign des TPUART angelehnt. Busware überarbeitet erfahrungsgemäß in einstelligen Jahreszyklen seine Produkte. So gab es bereits andere Module mit dem TPUART-Chip mit sehr ähnlichem Design, die direkt mit dem RPI verbunden wurden konnten. Ebenso ist ein TPUART USB-Modul verfügbar. Zum Zeitpunkt dieser Bachelorarbeit war kein End-Of-Life des PiGators bzw. für das EIB/KNX Modul festgelegt.

Der TPUART kommuniziert per UART. Der RPI bzw. der SOC des RPI`s bietet jedoch nur einen vollständigen Hardware UART [31] welcher auch das Paritätsbit unterstützt. Da es unterschiedliche Versionen des RPI gibt und die Version 3 an dem UART-Port bereits mit einem Bluetooth-Chip kommuniziert, muss zur Verwendung des PiGator mit dem TPUART Modul dieser erst deaktiviert oder zumindest auf einen anderen Port gelegt werden. Die Vorgehensweisen sind abhängig von der verwendeten Distribution. In der Regel muss ein entsprechendes DTOverlay geladen werden [32].

Das TPUART Modul baut auf dem Referenzdesign auf und ist folglich nicht interessant. Der RPI benötigt für unsere Zwecke zu viel Energie, somit ist der PiGator + TPUART Modul nicht geeignet. Für z. B. Heimautomatisierungsserver oder Analysen des KNX-Busses unter Verwendung verschiedener und vor allem höheren Programmiersprachen ist der PiGator + TPUART Modul jedoch sehr attraktiv.

Software

Als Software kann z. B. knxd verwendet werden, was gegen Ende dieses Kapitels vorgestellt und betrachtet wird.

KNX Busankoppler oder Ähnliches

Hardware

Eine weitere Variante besteht in der Verwendung des TPUART 2 Transceivers im Referenzdesign (BTM 117/12 PCBA) [33]. Ebenso ist ein KNX Busankoppler wie z. B. der Siemens 5WG1 117-2AB12 [34] verwendbar. Beide bieten eine UART Schnittstelle. Zur Kommunikation mit z. B. einem PC ist ggfs. ein UART nach USB Interface nötig, wie z. B. der FTDI FT232R, welcher vorzugsweise fertig im Referenzdesign oder ähnlich aufgebaut ist. Zu beachten sind dabei die Spannungspegel. Der TPUART2 im Referenzdesign verwendet 5 V. Der FT232R muss entsprechend aufgebaut werden oder es muss eine Pegelwandlung erfolgen. Die Spannungspegel müssen bei z. B. Verwendung von 3,3 V auf FTDI FT232R-Seite zwingend in beide Richtungen gewandelt werden, da der TPUART2 im Referenzdesign – wie bereits erwähnt – mit 5 V arbeitet [35] und die Spannungspegel zum TPUART 2(+) erst ab 3,5 V als High interpretiert werden [35 S. 16]. Falls eine Pegelwandlung benötigt wird, kann ein Optokoppler (oder iCoupler wie z. B. der ADUM1201ARZ von Analog Devices [36]) verwendet werden.

Diese Variante bietet eine günstige Möglichkeit mit dem KNX-Bus zu kommunizieren und der Arbeitsaufwand ist gering. Da der TPUART2 nur als QFN Packaging[4] erhältlich ist und sich somit schwer löten lässt, ist die Möglichkeit des Kaufs einer fertigen Platine mit allen entsprechenden Bauteilen interessant. Das BTM 117/12 PCBA stellt 5V als Spannungsversorgung für externe Geräte zur Verfügung. Die Stromabgabe ist jedoch auf maximal 50 mA begrenzt, solange die 20 V Versorgungsspannung nicht verwendet wird. Der oben erwähnte Busankoppler stellt nur maximal 10 mA am 5 V Ausgang und maximal 25 mA am 20 V Ausgang zur Verfügung.

Die Hardware ist günstig und gut verfügbar. Gerade die Kommunikation unter Verwendung des BTM 117/12 PCBA ist eine sehr attraktive Variante und wird im späteren Teil dieser Arbeit verwendet. Das Strombudget von 50 mA bei 5 V lässt uns viel Spielraum. Alternativ kann auch der TPUART [25] im Referenzdesign aufgebaut werden. Das SOIC Packaging lässt sich gut löten und die restlichen benötigten Teile sind durchgehend Standardbauteile. Nachteil ist jedoch das geringe Strombudget am 5 V Ausgang von 10 mA.

Software

Hier kann ebenso als Software knxd verwendet werden, welches gegen Ende dieses Kapitels vorgestellt und betrachtet.

[4] Bei diesem Packaging ragen die Pins seitlich nicht heraus.

Freebus

Hardware

Freebus stellt frei verfügbare Do-It-Yourself (DIY) Anleitungen für KNX/EIB Geräte zur Verfügung. Die Anleitungen umfassen u. a. Tastsensoren, Dimmer, Schaltausgänge und Netzteile. Fertig geätzte Platinen können über deren Shop bezogen werden. Die PCB-Layouts sind frei verfügbar, womit diese auch bei beliebigen Shops erstellt werden können. Freebus ist dabei jedoch ohne Gewähr kompatibel zu EIB/KNX. Freebus setzt dabei nicht auf KNX/EIB Transceiver Chips, sondern nutzt dafür eine eigens entworfene Schaltung [37] wobei u. a. die Einhaltung der Timings per z. B. Mikrocontroller in Software überprüft werden muss. Freebus ist nicht von der KNXA zertifiziert. Die Schaltungen verwenden überwiegend den 89LPC922. Ein Controller-Board welches z. B. den Atmel 168 verwendet ist jedoch ebenfalls entworfen worden.

Die Verwendung u. a. der Sende- und Empfangsstufen erscheint – gerade aus monetärer Sicht – attraktiv, bedeutet jedoch einen erhöhten Programmieraufwand und eine erhöhte Belastung des Mikrocontrollers.

Software

Das Git-Repository ist in einem unübersichtlichem Zustand [38]. Die Controllersoftware ist dabei für die Atmegas 168 und 328 verfügbar sowie dem 89LPC922 und 89LPC936. Alle vier sind 8-Bit Mikrocontroller. Die hardwarenahe Software ist nicht als Quellcode hinterlegt. Dadurch ist es schwierig Freebus als offene oder geschlossene Software einzuordnen, wie im Mikrocontroller.net-Forum ausführlich diskutiert wurde [39]. Im Widerspruch steht auch die Verwendung der GPL Lizenz zu der Aussage auf der Website, *„das[s] alles was auf unserer Seite zu finden ist auf keinen Fall kommerziel[l] vertrieben oder benutzt werden darf!“* [40]. Zur Dokumentation wird Doxygen verwendet bzw. soll verwendet werden. In weiten Teilen des Quellcodes fehlen jedoch die Doxygen spezifischen Schlüsselwörter. Die Quelltextkommentare sind mal in Englisch und mal in Deutsch.

Freebus wird inzwischen kaum noch gepflegt. Die letzten Commits liegen mindestens ein Jahr zurück. Die Dokumentation ist umfangreich, jedoch sind viele Links nicht mehr aktuell oder nicht mehr erreichbar.

Freebus ist ein interessantes und sehr umfangreiches Projekt, disqualifiziert sich jedoch für die Nutzung in anderen Projekten auf Grund der oben benannten Probleme bzgl. der Lizenz sowie des nicht öffentlich verfügbaren Quelltextes der hardwarenahen Software. Es eignet sich jedoch gut als Anlaufstelle für technische Informationen sowie teilweise als Beispielgeber zur Implementierung.

KNX IP Gateway / KNX USB-Schnittstelle

Hardware

KNX IP Gateways sind von der KNXA zertifizierte Geräte, welche die Kommunikation über IP nach KNX ermöglichen. Eine KNX USB-Schnittstelle ermöglicht die Kommunikation zum KNX-Bus hingegen über USB. Beide Wege werden von ETS voll unterstützt und sind vermutlich die am häufigsten verwendete Möglichkeit, die am KNX-Bus verwendeten Geräte per ETS zu programmieren. Beide Geräte werden hier nur wegen Ihrer Verbreitung erwähnt. Durch ihren Preis kommt die Verwendung eines dieser Geräte nicht in Frage. Ebenso sind die Designs nicht frei verfügbar.

Software

Als Software kann natürlich ETS eingesetzt werden. Ebenfalls kann z. B. OpenHAB mit dem KNX-Binding verwendet werden.

Folgend werden Softwarelösungen vorgestellt, die z. B. mit dem PiGator und dem TPUART funktionieren. Es wird somit nicht mehr in Hard- und Software unterteilt.

eibd/knxd

eibd ist ein Teil des BCU SDKs und wurde 2005 von der Automation Systems Group an der TU Wien entwickelt [41]. Es wurde u. a. mit dem Ziel entwickelt, eine frei verfügbare Toolchain zur – vor allem nicht-kommerziellen [42 S. 7] – Entwicklung von KNX Geräten bereitzustellen. Es ist in C geschrieben und ermöglicht es z. B. über den TPUART mit dem KNX Bus zu kommunizieren. Von höherem Interesse für die vorliegende Arbeit ist der TPUART Kernel Treiber. Der Quellcode von eibd sowie dem TPUART Treiber sind überwiegend kommentiert und vollständig in Bezug auf die Funktionen des TPUARTs. eibd ist für Linux gedacht – insbesondere mit der Kernelversion 2.6. Eine ausführliche Dokumentation ist vorhanden [42]. Das BCU SDK und somit eibd werden jedoch nicht mehr gepflegt. Die letzte Version erschien am 6. März 2011.

knxd ist ein Fork von eibd [43]. Es wird aktiv gepflegt und ist, ebenso wie eibd, unter Linux lauffähig. eibd bzw. knxd wird z. B. verwendet, um auf dem RPI mit dem KNX-Modul des PiGators zu kommunizieren. Ebenso kann es als Gateway für das KNX-Binding von OpenHAB verwendet werden – das KNX-Binding und OpenHAB wird in dieser Arbeit gegen Ende dieses Kapitels behandelt. knxd steht unter der GPL-2.0.

eibd/knxd sind für das Ziel dieser Arbeit attraktive Projekte. Vorrangig ist der TPUART Kernel Treiber von Interesse. Er lässt sich zwar nicht einfach portieren – viele Funktionen bzw. Bibliotheken werden verwendet, welche nicht auf einem ATXMEGA vorhanden sind (wie z. B. die printf-Funktionen [44]) – bietet jedoch eine gute Quelle zur Referenz und Inspiration an.

Calimero

Mit Calimero [45] begegnet uns ein weiteres Projekt, welches ihren Ursprung an der TU Wien bei der Automation System Group hat [46]. Calimero ist eine Sammlung von Java-Bibliotheken und Werkzeugen für KNX. Diese erfordern tiefer gehende Java- sowie KNX-Kenntnisse. Es bietet eine breite Unterstützung an Protokollen und vielen, jedoch nicht allen, DPTs an. JavaDoc wird zur Dokumentation verwendet. Calimero wird überwiegend aktiv von (ehemaligen) Mitarbeitern der Automation System Group der TU Wien gepflegt.

Calimero steht unter der GPL mit Classpath Exception [47]. Das Projekt wurde erst auf Sourceforge [48] gehostet, ist inzwischen jedoch nach GitHub umgezogen. Es ist umfangreich und gut dokumentiert.

Calimero ist (fast) nur in Java geschrieben, womit es nicht auf einem ATXMEGA produktiv verwendet werden kann. Somit ist es für das Ziel der vorliegenden Arbeit nicht relevant. Calimero wird in dieser Übersicht nur aufgeführt, da es in z. B. dem KNX-Binding für OpenHAB verwendet wird (siehe nachfolgender Abschnitt). Dadurch hat es eine hohe Relevanz im Bereich der Heim- und Gebäudeautomatisierung.

OpenHAB KNX-Binding

OpenHAB ist eine in Java geschrieben Softwarelösung zur Heim- bzw. Gebäudeautomatisierung. Besonders mächtig wird es durch die Verwendung von Bindings und weiteren Add-ons [49]. Das KNX-Binding [50] bietet die Möglichkeit KNX Geräte in OpenHAB einzubinden. Dieses setzt dabei stark auf die Bibliothek Calimero. Damit gelten z. B. auch die Einschränkungen bei den DPTs. Abgesehen von einem Wiki Eintrag auf der zugehörigen GitHub-Seite gibt es keine Dokumentation. Für weitere Dokumentation wird auf den Quellcode von Calimero verwiesen. Die Quellen des KNX-Bindings werden unter der EPL veröffentlicht.

Da das KNX-Binding stark auf Calimero aufsetzt, gilt für dieses Binding dasselbe wie für Calimero.

Umsetzung des Proof of Concepts

Realisierung der Hardware des Proof of Concepts

Vorüberlegungen zur Hardware

Um der Erweiterbarkeit und dem einfachen Nachbau Rechnung zu tragen, wurde ein modularer Ansatz gewählt. Ebenso wurden die Möglichkeiten betrachtet, welche der Bus bereits liefert. Des Weiteren soll möglichst wenig in bestehende Installationen eingegriffen werden und der normale Betrieb des KNX-Busses soll nicht gestört werden.

Im gesamten Projekt wird dabei möglichst auf SMD-Bauteile verzichtet, insbesondere auf Packages wie QFN. Es wurde darauf geachtet, dass die verwendeten Bauteile lange verfügbar und günstig erhältlich sind. Wenn möglich sollte auch auf bereits Vorhandenes aufgesetzt werden – die Diskussionen sind bei dem jeweiligen Projekt in der im vorherigen Kapitel erstellten Übersicht nachzulesen. Zusätzlich soll eine gute Dokumentation zur Verfügung gestellt werden.

Der TPUART (2, 2+) von Siemens als KNX Transceiver ist eine naheliegende Wahl. Dieser ist günstig erhältlich und weit verbreitet. Die Verwendung legt weitere Parameter für kommende Komponenten fest. Der TPUART liefert 5 V mit bis zu 10 mA. Der TPUART2 (+) bietet zusätzlich eine Versorgungsspannung von 3,3 V und – viel wichtiger – bis zu 50 mA. Ist der TPUART, welcher im SOIC-Package verfügbar ist, noch relativ gut zu löten, sieht es bei dem TPUART2(+) schon deutlich schwieriger aus. Dieser ist nur im QFN-Package erhältlich. Es besteht jedoch die Möglichkeit, diesen im Referenzdesign bereits verbaut zu erwerben. Zu beachten sind noch die Systemspezifikation von TP-1 zum u. a. Anschaltverhalten [15 S. 12-16]. Der TPUART2 sorgt dabei für die Einhaltung dieser Spezifikationen z. B. bei schnellen Änderungen der Last, welche von ihm versorgt werden [35 S. 11].

Da 50 mA bei Weitem nicht genug für z. B. einen RPI sind, rückte der Fokus in Richtung Mikrocontroller. Die Chips von Atmel sind weit verbreitet, gut erhältlich und gut dokumentiert. Häufig existiert für viele Schnittstellen von Atmel ein bereitgestellter Beispielcode. Des Weiteren sind umfangreiche und zahlreiche Tutorials zum Einstieg verfügbar. Es werden mindestens zwei Hardware UARTs – um zwei TPUARTs gleichzeitig ansprechen zu können – benötigt und die Energieaufnahme sollte möglichst gering sein. Nach Durchsicht der Produktdatenbank von Atmel fiel die Wahl auf den ATXMEGA64A4U. Dieser hat eine geringe Energieaufnahme von ungefähr 10 mA unter Verwendung von 3,3 V bei 32 MHz sowie fünf Hardware USARTs[5]. Zusätzlich bietet er eine USB-Schnittstelle, mehrere integrierte Taktgeber – mit ausreichender Genauigkeit und Geschwindigkeit – sowie eine Crypto-Engine für AES-128. AES-128 wird für KNX Data Security verwendet.

Um ggfs. eine abgesicherte KNX-Strecke zu realisieren, wurden mehrere Möglichkeiten betrachtet. Im Freebus Projekt ist eine DIY-Anleitung zum Bau eines Netzteils zu finden. Jedoch gibt es Anmerkungen, dass dieses evtl. nur für 1-2 Geräte [51] zuverlässig arbeitet. Die Bauteilkosten belaufen sich auf etwa 20 € ohne Platine [52]. Die reine KNX-Drossel ist deutlich günstiger herzustellen. Durch diese ließ sich unter Verwendung der häufig auf dem zweiten Adernpaar einer KNX Installation mitgeführten Spannungsversorgung ein weiterer Bus realisieren. Im Forum von Mikrocontroller.net wurde durch den Benutzer „Jörg S.“ eine Drossel von Busch und Jäger zerlegt

[5] USART ist eine Variante von UART und ermöglicht ebenfalls eine synchrone Datenübertragung. Diese wird jedoch im Rahmen der Arbeit nicht verwendet. Im Weiteren wird der Port UART bezeichnet.

und nachkonstruiert [53]. Ebenso wurde dort auf ein Patent mit detaillierten Darstellungen zu einer Drossel verwiesen [54].

Somit sind die Vorüberlegungen abgeschlossen. Diese werden im Folgenden realisiert, verworfen oder ggfs. erweitert.

Diskurs zum Zusammenbau und zum Hardwaredesign

Für die weiteren Schritte kamen zum einen der TPUART [25], welcher im SOIC-Package verfügbar ist und nach dem Referenzdesign aufgebaut wurde, und zum anderen das Referenzboard mit TPUART2 [33] zum Einsatz (im Weiteren wird verkürzend nur noch von BTM2-PCB gesprochen) – der Grund ist im Anhang B erläutert. Der TPUART wurde nur testweise aufgebaut, da dieser den Strom für externe Geräte auf 10 mA begrenzt. Mit der BTM2-PCB sind hingegen 50 mA bei 5 V möglich.

Der TPUART bzw. TPUART2 ist nun funktional und verwendbar. Zu beachten ist dabei, dass der Chip mit dem KNX-Bus (oder einer entsprechenden Spannungsquelle, die der Busspannung entspricht) zwingend verbunden sein muss, um sich in einem operativen Zustand zu befinden.

Wie bereits in den Vorüberlegungen festgestellt haben wir nur ein geringes Strombudget von maximal 50 mA [35 S. 12], das direkt vom BTM2-PCB und damit vom Bus kommt. Das BTM2-PCB stellt uns eine 5 V und eine 20 V Spannungsquelle zur Verfügung. Der Mikrocontroller benötigt jedoch zwingend 3,3 V. Somit benötigen wir einen Spannungsregler, der auf 3,3 V regelt[6].

Zur Wandlung der Spannung von 5 V auf 3,3 V entschied ich mich auf Grund der geringen Differenz zwischen Ein- und Ausgangsspannung für einen LDO-Spannungsregler. Diese sind günstig und bieten einen akzeptablen Wirkungsgrad. Einzig die Neigung zum Schwingen ist für die Anwendung als Spannungsquelle für Mikrocontroller nicht wünschenswert. Konkret verwendet wurde der LF33CV [55]. Der Aufbau entspricht weitestgehend der Referenzschaltung, dieser wurde nur durch eine Rückflussdiode erweitert [56]. Auf lange Sicht muss sich zeigen, ob die Spannungsquelle stabil und störungsarm arbeitet. Eventuell werden weitere Maßnahmen zur Unterbindung z. B. der Restwelligkeit nötig.

Der ATXMEGA64A4U wurde im TQFP Package verwendet. Zum Verlöten von TQFP reicht ein wenig Löterfahrung aus. Der ATXEMGA64A4U wurde auf ein Breakoutboard gelötet und daraufhin nach den Empfehlungen aus dem Dokument AVR1012 [57] auf einem Breadboard aufgebaut. PDI wurde zur Programmierung und zum Debuggen verwendet. Als Programer wurde der ICE von Atmel verwendet.

Im Weiteren wurde die BTM2-PCB mit dem ATXMEGA verbunden. Zu beachten ist die nötige Pegelwandlung. Die Spannungspegel dürfen an den Eingängen des ATXMEGA 3,6 V nicht übersteigen und müssen somit zwingend in beide Richtungen gewandelt werden. Der TPUART2 auf dem BTM2-PCB arbeitet mit 5 V [35] und die Spannungspegel zum TPUART2 werden erst ab 3,5 V als High interpretiert [35 S. 16]. Zur Pegelwandlung kommt der iCoupler ADUM1201ARZ von Analog Devices [36] zum Einsatz. Eine galvanische Trennung ist durch unseren Schaltungsaufbau nicht gegeben.

[6] Der TPUART2 bietet zwar die Möglichkeit anstatt 5 V, 3,3 V zur Verfügung zu stellen, benötigt dazu jedoch einen anderen Aufbau. Dadurch wäre es ebenfalls möglich die Spannungspegel der UART-Schnittstelle auf 3,3 V zu ändern.

Der Schaltplan ist, wie alle anderen, im Anhang C zu finden. Die Drossel bzw. die Erweiterung des KNX-Busses wird auf Grund des Umfangs gesondert betrachtet.

Analyse der Energieaufnahme

Betrachten wir zuerst den Energiebedarf des ATXMEGA64A4U. Wir verwenden den internen Oszillator mit 32 MHz. Laut Datenblatt benötigt unsere Konfiguration 12 mA sowie 465 µA (Typ.) für den 32 MHz internen Oszillator. Die UART-Schnittstelle schlägt mit typ. 2,5 µA bei 9600 Baud zu Buche [58 S. 117-118]. Der Spannungswandler hat einen maximalen Eigenbedarf von 12 mA [55 S. 17]. Der Pegelwandler hat eine Stromaufnahme von maximal fast 2 mA laut Datenblatt [36 S. 17].

Somit sind wir weit unterhalb der 50 mA Grenze. Dies sind jedoch nur theoretische Werte, da elektrische Bauteile, produktionsbedingt, leicht unterschiedliche reale Bedarfswerte haben. Ebenso fehlen u. a. die Kondensatoren in unserer Betrachtung.

Betrachten wir nun die reale Energieaufnahme. Als Spannungsquelle kam ein Steckernetzteil von Voltcraft mit 24 V und 150 mA Ausgangsleistung zum Einsatz. Dahinter wurde eine KNX-Drossel aufgebaut, sodass wir einen KNX-Bus zur Verfügung haben. An diesem wurde ein Taster und das BTM2-PCB angeschlossen. Wobei an dem BTM2-PCB der Spannungswandler etc. angeschlossen wurde. In Abbildung 4 ist der Aufbau und die Messung der Stromaufnahme abgebildet. Es wurde eine LED zur leichteren Fehlersuche beim Aufbau verwendet. Der Schaltplan ist unter Anhang C zu finden.

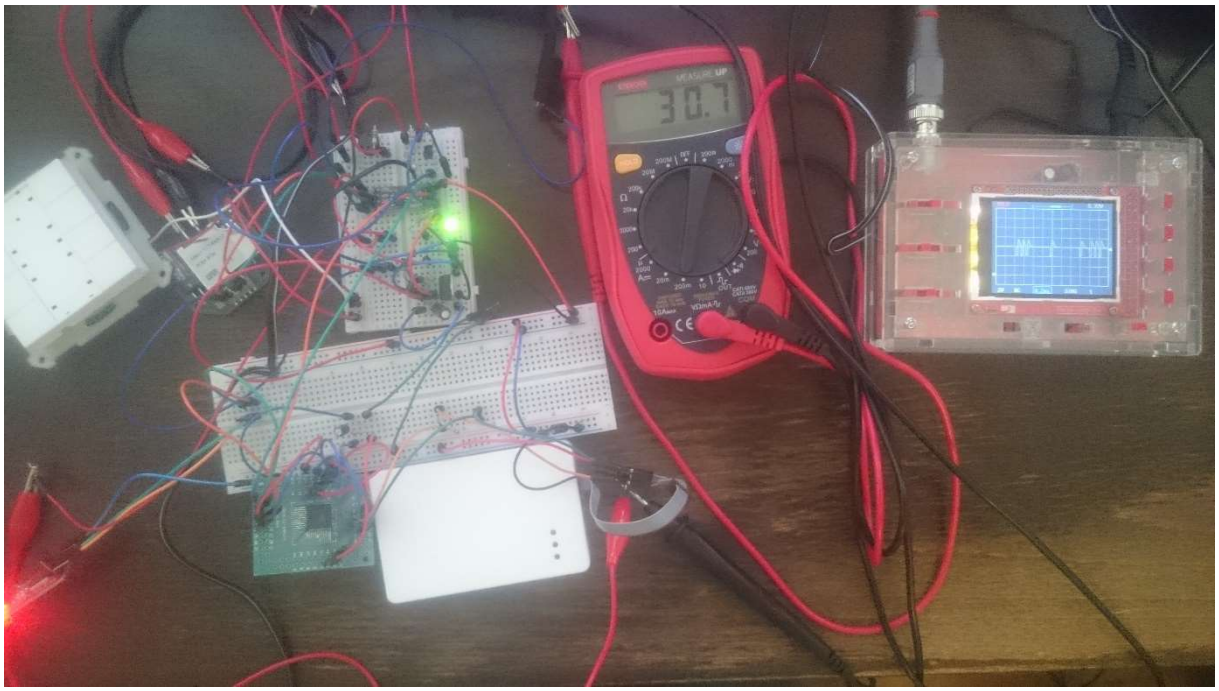


Abbildung 4 – Messung der Stromaufnahme

Die Stromaufnahme lag mit LED bei 30,7 mA in der Abstufung 200mA mit dem Etekcity Measure Up MSR-R500 welches laut Datenblatt „DC Current: 2000µA/20mA ±(1%+2), 200mA ±(1.2%+2), 10A ±(2%+5) [59]“ als Messfehler hat. Der Messfehler beträgt somit: I_abw = 0,24 * 30,7 mA + 0,2 mA = 7,568 mA bzw. gerundet 7,6 mA (Der Messfehler durch die Temperaturabweichung wird ignoriert, da keine Angaben vorhanden sind) [60]. Somit liegt die reale Stromaufnahme bei 30,7 ± 7,6 mA. In diesem Aufbau nimmt die LED einen Strom von ungefähr 12 mA auf, womit wir auf eine maximale Stromaufnahme der Schaltung ohne LED von 26,3 mA kommen.

Zum Debuggen und zur einfachen Interaktion wurde an einen weiteren UART-Port des ATXMEGA ein UART nach USB Interface mit einer Baudrate von 57600 Bauds angeschlossen. Somit waren zum Zeitpunkt der Messung 2 UART-Ports aktiv.

Falls der TPUART2 in einem anderen Aufbau verwendet wird, sind im Anhang B Messungen zur Leistungsaufnahme hinter einer 3,3 V Spannungsquelle zu finden.

Diskurs zur Erweiterung des KNX-Busses

Betrachten wir nun die Möglichkeit einen weiteren, von dem anderen bezüglich der KNX-Signalübertragung, unabhängigen Busabschnitt, angeschlossen an einen weiteren KNX-Transceiver, zu ermöglichen. Dies soll zukünftig die kryptografisch abgesicherte Strecke ermöglichen, so dass dieser Busabschnitt exponiert werden kann. Der ATXMEGA64A4U bietet insgesamt 5 UART-Ports, sodass wir nur einen weiteren Port initialisieren und mit z. B. einer weiteren BTM2-PCB verbinden müssen.

Zur Erweiterung des KNX-Busses kommen viele Wege in Frage. Prinzipiell müssen wir den Bus erweitern, ohne dass die Signale auf dem jeweils anderen Bus induziert werden oder vom KNX Modem detektiert werden können. Ebenfalls muss für beide die Funktionalität gewährleistet bleiben. Idealerweise wird keine weitere Spannungsversorgung benötigt.

Somit wird zunächst der Nachbau der KNX-Drossel betrachtet. Im Anhang A wird dieser vorgestellt und besprochen. Durch Verwendung dieser, sowie durch das Hinzufügen eines Tiefpasses zur Filterung der Signale, können wir einen weiteren KNX-Bus erzeugen. Dies ist in der KNX Spezifikation nicht vorgesehen – die simulierten Werte verletzten jedoch nicht die Spezifikation.

Als Tiefpass wird ein LC-Glied[7] verwendet. Für die Simulationen wird die bereits beschriebene Drossel leicht modifiziert und eingesetzt. Die Grenzfrequenz muss dabei entsprechend tief gewählt werden – die Signale dürfen auf den jeweils anderen Bus keine bzw. eine nur möglichst geringe Auswirkung haben. In der Simulation hat sich eine Kombination aus einer Spule von 10 mH und einem Kondensator von 2200 µF als eine sehr gute Lösung erwiesen. Selbst bei extremer Belastung des Busses weicht die Versorgungsspannung nur in einem Bereich von weniger als 40 mV ab. Weitere Details sowie die Simulationen sind im Anhang A zu finden. Die Verfügbarkeit sowie der Preis passen ebenfalls ins Konzept.

Durch z. B. eine Suppressordiode und eine Feinsicherung wäre es zusätzlich möglich, den zugrundeliegenden KNX-Bus zu schützen, z. B. falls jemand hohe Spannungen an die exponierte Leitung anlegt.

[7] Ein LC-Glied besteht aus einer Spule und einem Kondensator. Ein RC-Glied – bestehend aus einem Widerstand und einem Kondensator – lieferte schlechtere Ergebnisse.

Realisierung der Software des Proof of Concepts

Vorgaben zur Software

Durch den Einsatz des ATXMEGA64A4U ist die Wahl der Programmiersprache stark eingeschränkt. Da wir sehr hardwarenah sind und ebenfalls effizient und platzsparend programmieren müssen, wird C verwendet. Alternativen wären z. B. Assembler oder C++. Die bereits behandelten Projekte können, wie bereits diskutiert, nicht verwendet werden. Es wird modular programmiert.

Als Entwicklungsumgebung wird Atmel Studio – ein angepasstes MS Visual Studio – verwendet. Atmel hat dies u. a. durch eine komplette Toolchain erweitert. Dies ermöglicht es uns direkt aus Atmel Studio unseren Mikrocontroller zu programmieren. Zur Programmierung und zum Debuggen wird PDI verwendet. Atmel Studio ermöglicht das umfangreiche Setzen von u. a. Breakpoints sowie die Betrachtung und die Manipulation des Speicherinhalts des Chips. Als Programmer wird der Atmel ICE verwendet. Git bzw. GitLab wird zur Versionsverwaltung verwendet.

Einschränkungen und Implikationen durch die Hardware

Die KNX-Spezifikation [15 S. 37-41] bzw. der TPUART2-Chip [35 S. 21-34] sehen unterschiedliche Dienste vor, z. B. zum Senden eines Pakets sowie zum Aktivieren des Busmonitor Modus. Wobei im Datenblatt des TPUART`s weitere Chip-spezifische Dienste wie z. B. das Setzen der Adresse gegeben sind. Es liegt somit nahe die Dienste dementsprechend zu implementieren.

Der ATXMEGA64A4U bietet uns 4 kByte SRAM sowie 64 kByte Flash. Zum z. B. Speichern der Adresse kann der EEPROM mit einer Größe von 2 kByte genutzt werden [61]. Atmel gibt für die XMEGA`s u. a. Namenskonventionen vor [62], welche beim Programmieren umgesetzt wurden. Darüber hinaus gibt Atmel Empfehlungen und Tipps zum Optimieren des Quellcodes [63]. Soweit möglich wurden Zeichenketten im Programmspeicher untergebracht, um den SRAM zu entlasten, was eine Empfehlung aus dem vorher genannten Dokument ist. Atmel stellt dazu das „PROGMEM“-Makro und die „pgm_read_byte“-Funktion durch das Einbinden der entsprechenden Header-Datei „avr/pgmspace.h“ zur Verfügung. Die grundlegenden UART-Funktionen [64] [65] und die grundlegenden Funktionen zum Kalibrieren und der Verwendung der Clock [66] [67] wurden von Atmel übernommen[8].

Weitere Anmerkungen

Zur einfachen Kommunikation und zum Übermitteln von Debugnachrichten wurde einer der UART-Ports mit einem UART nach USB-Wandler verbunden und entsprechend programmiert. Es wurde ein DEBUG-Makro gesetzt, so dass die entsprechenden Debugnachrichten etc. nur in der Debug

[8] Der Quellcode von Atmel zur Kalibrierung enthielt ein Bug der gepatcht werden musste.

Konfiguration vom Atmel Studio kompiliert und ausgeführt werden. Die Kommunikation zum PC z. B. für die Debugnachrichten findet über UART mit einem UART nach USB-Wandler statt.

Konfigurationen wie z. B. die Verwendung der UART-Ports werden in den jeweiligen Header-Dateien vollzogen.

Dokumentation

Die Dokumentation wird durch Doxygen [26] realisiert. Doxygen generiert anhand von Schlüsselwörtern ähnlich wie JavaDocs eine Dokumentation und erstellt darüber hinaus z. B. Abhängigkeits- und Aufrufgraphen. Doxygen ist frei verfügbar und kostenlos erhältlich.

Detaillierte Erklärung zur Implementierung

Der ATXMEGA64A4U[9] verwendet beim Starten stets den 2 MHz internen Taktgeber. Bei Verwendung des internen 32 MHz Taktgebers, empfiehlt es sich diesen vorher zu kalibrieren. Zur Kalibrierung wird der interne 32 kHz Referenztaktgeber aktiviert und verwendet. Nach erfolgter Kalibrierung werden der 2 MHz und der 32 kHz Taktgeber deaktiviert sowie die Konfiguration wird gesperrt. Zusätzlich wird jeweils gewartet bis die Taktgeber einen stabilen Zustand erreichen bevor diese verwendet werden bzw. auf den 32 MHz Taktgeber umgeschaltet wird. Die Konfiguration des UART-Ports ist abhängig vom verwendeten Takt, so dass bei Änderungen des Takts die Register entsprechend geändert werden müssen.

Danach werden die UART-Ports konfiguriert und aktiviert – die Funktionen orientieren sich stark an den Beispielen von Atmel [64] [65]. Zurzeit wird ein Port zur Kommunikation mit dem BTM2 PCB verwendet und ein weiterer zur Kommunikation mit dem UART nach USB-Wandler. Über letzteren kann interaktiv mit dem TPUART2 bzw. KNX-Bus kommuniziert werden.

Zur einfacheren Handhabung wurden Funktionen zum Senden und Empfangen kompletter Zeichenketten implementiert. Ebenfalls wurden Interrupt Service Routinen definiert, sowie Funktionen, um direkt Zeichenketten aus dem Programmspeicher zu versenden. Teile des Codes stammen dabei aus den Beispielen von Atmel [65] [64].

Zur interaktiven Kommunikation wurde eine textbasiertes User Interface implementiert, welches die Mehrheit der TPUART2-Dienste unterstützt. Es können Daten gesendet werden, wobei auch die Berechnung der Checksumme implementiert ist. Ebenfalls sind größtenteils Plausibilitätschecks sowie Textdokumentationen umgesetzt. Es sind jedoch bei Weitem nicht alle Dienste implementiert, z. B. fehlt der Dienst zum Anfragen von Daten. Alle Dienste, die im Rahmen des PoCs benötigt wurden sind implementiert, z. B. zum Senden oder zum Aktivieren des Busmonitor-Modus. In Abbildung 5 ist ein Beispiel zu sehen. Dort sind ebenfalls die bereits implementierten Befehle zu erkennen. Der Output gibt teilweise Hilfestellung zum Ausfüllen der nächsten Felder.

[9] Der ATXMEGA64A4U besitzt drei interne Systemtaktgeber: 32 kHz, 2 MHz und 32 MHz.

```
Implemented TPUART Functions are:
U_Reset.request = reset_r
U_State.request = state_r
U_ActivateBusmon = act_busmon
U_ProductID.request = prod_r
U_ActivateBusyMode = act_busymode
U_ResetBusyMode = res_busymode
U_SetAddress = setaddr
U_AckInformation = ackinfo
U_L_Data-Services = senddata
Sending U_ProductID.request...
Waiting(50ms) for the U_ProductID.response...
Response:
1000001
U_ProductID.request finished!
Sending U_ProductID.request...
Waiting(50ms) for the U_ProductID.response...
Response:
1000001
U_ProductID.request finished!
Enter Flags:
1 = Addr; 2 = Busy; 3 = NACK;
Sending U_AckInformation-Service...
U_AckInformation-Service finished!
U_L_DataStart-Service!
---------------------------------
| EIB-Control Field
---------------------------------
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
---------------------------------
| F | F | R | 1 | c | c | 0 | 0 |
---------------------------------
FF = Frame Format; 10 Standard Length; 00 ext; 11 L_POLLDATA
R = Repeatflag; 1 = not repeated, 0 = repeated
cc = 00 system; 10 alarm; 01 high; 11 normal
Enter EIB-Control field!(As Hex with leading 0x otherwise as decimal or if it begins with 0 than its octal!)
```

Abbildung 5 - UI Beispiel

Detaillierte Erklärungen u. a. zu den einzelnen Funktionen können in der Doxygen-Dokumentation nachgelesen werden (auf der beigelegten CD enthalten – diese wird zusammen mit dem Quellcode noch veröffentlicht). Im Anhang E ist ein kleines Beispiel zu finden. Doxygen kann vor einer Auswertung der Makros, diese belegen; somit können u. a. die Debugpfade inkludiert bzw. exkludiert werden.

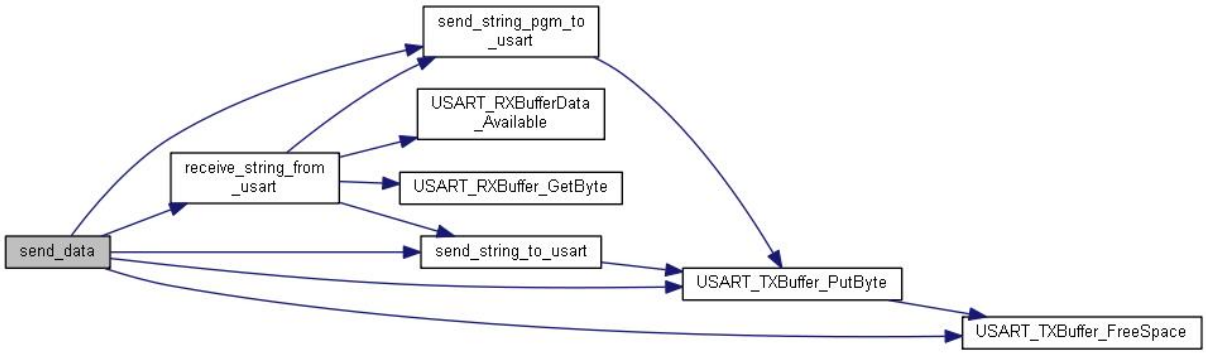


Abbildung 6 - Aufrufgraph der „send_data()“-Methode

Abbildung 6 stellt beispielhaft den Aufrufgraph der Methode „send_data()“ dar. Hier ist gut die Funktionsweise der Methode sowie die modulare Programmierung zu erkennen. Zeichenketten, die empfangen oder (aus dem Programmspeicher) gesendet werden, werden durch die entsprechenden Methoden in Byte zerlegt und gesendet bzw. byteweise Empfangen und zu Zeichenketten konkatiniert – gut zu erkennen an den RX- und TX-Methoden.

Im Anhang B ist eine kurze Beschreibung von aufgetretenen Problemen zu finden.

Ausblick

In einer beabsichtigten Masterarbeit soll tiefergehend betrachtet werden, ob das BTM2 PCB gegen einen TPUART2+-Aufbau geändert werden kann. Zusätzlich soll untersucht werden, ob der TPUART (2, +) nach dem Vorbild von Freebus ersetzt werden kann [68] [37].

Ebenso sind weitere Optimierungen der Schaltungen geplant. Die zurzeit nicht verwendeten Sleep Modes sollen ebenfalls implementiert werden – wobei bei eventueller Verwendung eigener Sende- und Empfangsstufen die Stromänderungslimits auf dem Bus [15 S. 13] genau betrachtet werden müssen.

Die Leitungen wie z. B. Reset und Save vom TPUART2+ [35 S. 7] werden zurzeit nicht ausgewertet, dies soll sich in Zukunft ebenfalls ändern. Zurzeit sind nur Schaltpläne der Hardware verfügbar (siehe Anhang C), ein PCB-Layout wäre im Rahmen einer Masterarbeit wünschenswert.

Da ebenfalls noch weitere UART-Ports zur Verfügung stehen, wäre die Prüfung zur Umsetzung eines KNX RF-Interfaces möglich. Als Funk-Transceiver könnte z. B. der CC1101 [11] verwendet werden.

Des Weiteren ist die Implementierung von Routing- und Gateway-Funktionen sowie KNX Data Security denkbar. Ebenfalls die Erweiterung der Schaltung um einen zweiten Transceiver. Bei entsprechender Verfügbarkeit von Geräten mit „KNX Data Security“-Implementierungen ist eine Prüfung dieser ebenfalls möglich.

Analyse des KNX Data Security Entwurfs

Allgemeines

In der frei verfügbaren KNX-Spezifikation v2.1 ist bisher nur ein Entwurf von KNX Data Security zu finden [14]. Die letzte Änderung in diesem Entwurf ist vom 14. Mai 2013. Die endgültige Spezifikation – welche mir jedoch nicht zur Verfügung steht – ist im zweiten Quartal 2016 erschienen [22]. Der Entwurf sieht vor, den „Application Layer“ in „Plain Application Layer“ und dem neu definiertem „Secure Application Layer“ aufzuteilen. Unser Augenmerk wird somit auf dem „Secure Application Layer“ liegen. In dem vorliegenden Entwurf sind manche Angaben nicht konsistent und nicht vollständig, so dass dies nur eine vorläufige Analyse darstellt.

Unterschiede des Frameaufbaus

Falls der „Secure Application Layer“ verwendet wird, ändern sich die Frames erst ab dem siebten Oktett sowie die Bits 0 und 1 des sechsten Oktetts. Somit verbleiben u. a. das Kontrollbyte sowie die Sende- und Empfangsadressen im Klartext. Andernfalls wäre es nicht einfach möglich „KNX Data Security“-fähige Geräte in bestehende Netzwerke einzubinden. Ebenfalls immer im Klartext jedoch durch ein MAC[10] abgesichert werden die darauffolgenden 68 Bit übertragen [14 S. 18-19]. Diese enthalten u. a. die 48 Bit große Sequenznummer.

Details und Analyse zur Verschlüsselung sowie Wahrung der Authentizität

KNX Data Security sieht zurzeit eine Möglichkeit zur Verschlüsselung und Sicherung der Authentizität der Nutzdaten vor bzw. eine Möglichkeit, um nur die Authentizität der Nutzdaten zu gewährleisten. CCM bzw. KNX Data Security unterteilt diese in sogenannte zusätzlichen Daten („additional Data“ (A)) und Nutzdaten („Payload“(P)) [14 S. 88] wobei die Bezeichnung nicht einheitlich verwendet wird [14 S. 18]. Zur Sicherung der Authentizität wird AES im CCM Modus[11] verwendet, wobei die Schlüssellänge 128 Bit beträgt. Der Aufbau der Datenblöcke „B_0“ bis „B_n“ über welche der MAC gebildet wird, lässt sich in dem Entwurf nachlesen [14 S. 17, 88]. Der Aufbau von „B_0“ ist dabei KNX-spezifisch. Durch Verwendung des CCM Modus wird zuerst ein MAC über alle Nutzdaten (A und P) gebildet. Dieser wird daraufhin inklusive der Nutzdaten (P) (diese können sich von den Nutzdaten der MAC-Generierung unterscheiden) im Counter Modus verschlüsselt. KNX bezeichnet die Nonce[12] inkl. Counter als Block Counter „Ctr“, der erste Block Counter („Ctr_0“) enthält u. a. die Sequenznummer sowie einen 8 Bit großen Zähler [14 S. 17]. AES-CCM mit leerer Payload wird benutzt, um die Authentizität von unverschlüsselten Datenpaketen zu gewährleisten. Als Initialisierungsvektor wird in

[10] Message Authentication Code
[11] In KNX Data Security wird der MAC über weitere Daten (A) gebildet als die Verschlüsselung (über die Daten P). Der MAC gewährleistet u.a. die Authentizität der Quell- und Zieladresse [22].
[12] Nur einmal verwendete Zahl

beiden Fällen 0 verwendet. Der erste 128 Bit große Block „B_0“ ist dabei ähnlich aufgebaut wie der Block Counter „Ctr_0“, wobei sie sich im ersten Oktett immer unterscheiden [14 S. 17]. Der erstellte MAC wird dabei auf 32 Bit gekürzt – wobei der Entwurf der Spezifikation bei der Erstellung der MAC nicht Eindeutig ist [14 S. 18,90]. Bei beiden Verfahren werden die Blöcke falls nötig mit Nullen aufgefüllt [14 S. 88]. Nicht klar bzw. vielleicht nicht korrekt ist, dass beim Modus „Verschlüsselung und Authentizität“ 32 Bit für den MAC am Ende des Frames angegeben sind. Dieser ist bereits im Ciphertext enthalten [14 S. 90, 19]. Vermutlich ist gemeint, dass der Ciphertext diesen 32 Bit großen MAC enthält.

CCM unter Verwendung von AES bietet bei korrekter Verwendung einen sehr guten Schutz. Jedoch wird das Schutzniveau durch das Kürzen des MAC von 128 Bit auf 32 Bit deutlich gesenkt. Die Datenrate auf dem Bus beträgt 9,6 kBit/s. Ein MAC-gesichertes Paket ist im kürzesten Fall 19 Bytes groß, d.h., es ist unter idealen Bedingungen möglich 63 Frames pro Sekunde zu senden. Somit hat man 50% der möglichen MACs in 394 Tagen per Brute-Force durchprobiert. In KNX RF Netzen ist dies noch dramatischer, da KNX RF eine Datenrate von 16,384 kBit/s hat. Falls ein KNX-Netz durch z. B. Linienkoppler aufgeteilt wird und Geräte auf z. B. mehrere Linien aufgeteilt wurden, welche über ein gesichertes Multicast [14 S. 52-55] [69 S. 11] angesprochen werden können, kann dieser Angriff auf den jeweils unterschiedlichen Linien parallel durchgeführt werden, womit die Angriffsdatenrate ggfs. erhöht werden kann. Dies ist abhängig von den Kopplern, z. B. ob Koppler die Pakete weiterleiten [16 S. 30].

Die Sequenznummer wird stets inkrementiert und muss stets höher als die vorangegangene sein [14 S. 21]. Somit werden Replay-Angriffe unterbunden. Falls ein Gerät ausgetauscht wird, werden diese Daten ausgelesen und auf das neue Gerät geschrieben. Falls es keine Möglichkeit gibt an diese Daten zu kommen, wie z. B. bei einem Defekt von KNX-Geräten wurde ein KNX-Service zum Abgleich der Sequenznummer spezifiziert [14 S. 22]. Die im KNX-Netzwerk verbliebenen Geräte oder zumindest eines davon kennen die letzte Sequenznummer des ausgewechselten Geräts – ein Gerät genügt diesem Service. Dieses Gerät wird mit dem „S-A_Sync“-Service angesprochen, um die letzte eigene sowie fremde (also die vom im Netzwerk verbliebenem Gerät) valide Sequenznummer zu erfragen. Dabei wird dem Gerät eine 32 Bit große Challenge gestellt. Diese wird verschlüsselt übertragen, wobei die Angaben in dem Entwurf erneut so nicht stimmen können [14 S. 22-24]. Der Ciphertext ist mindestens 32 Bit groß durch die Inkludierung der MAC und müsste durch eine 32 Bit große Challenge somit 64 Bit groß sein. Vermutlich ist wieder gemeint, dass der Ciphertext den MAC enthält. Ebenfalls wird in dem Entwurf vorgegeben, dass die komplette „S-A_Sync_Req“-PDU mit CCM gesichert wird, die Grafik der Frames zeigt jedoch nur die Challenge und die MAC als verschlüsselte Daten [14 S. 22] [22].

KNX-Geräte werden mit einem Schlüssel ausgeliefert (Factory Default Setup Key(FDSK) [14 S. 64]). Dieser ist pro Gerät unterschiedlich und kann z. B. auf das Gerät gedruckt werden. Dieser Schlüssel wird bei der ersten Kommunikation mit dem Gerät verwendet um u. a. den „Security Tool Key [14 S. 60]“ dem Gerät mitzuteilen. Der „Security Tool Key“ wird vom ETS generiert und an die entsprechenden Geräte verschlüsselt verteilt. Dieser Schlüssel wird u. a. zur Konfiguration mit dem ETS verwendet. Beide Schlüssel müssen sicher gespeichert und verwendet werden, so dass diese nicht per z. B. JTAG ausgelesen werden können. Die Kommunikation unter Verwendung dieser Schlüssel wird immer und ausschließlich im CCM Modus mit Verschlüsselung und Authentizität betrieben [14 S. 42, 60].

Die Kommunikation untereinander ist jeweils mit einem spezifischen Schlüssel geschützt, welcher bei der Programmierung vergeben wird. Dazu halten die Geräte u. a. eine Schlüsseltabelle [14 S. 56] sowie eine Sequenznummerntabelle vor [14 S. 57-60].

Weitere Sicherheitsmechanismen

KNX Data Security sieht auch eine Zugriffskontrolle sowie die Verwendung von Rollen vor [14 S. 38]. Die Implementierung und Verwendung ist jedoch keine Pflicht. Ebenso sind Teile davon in der KNX Data Security Spezifikation nicht spezifiziert [22].

Des Weiteren sieht KNX Data Security vor, dass Fehler wie z. B. die Verwendung von einer zu kleinen Sequenznummer – was auf einen Replay-Angriff hindeuten kann – notiert werden [14 S. 47]. Eine Behandlung dieser Fehler wird vorgeschlagen. Dies ist jedoch nicht genauer spezifiziert.

Weitere Fehler und Inkonsistenzen im KNX Data Security-Entwurf

Der Entwurf beschreibt im Anhang B [14 S. 92-94] den Hintergrund und die Motivation. Wobei hier der CFB Modus beschrieben wird. Bedenklich ist die Mitsendung des IVs bei jedem Paket, so dass alle empfangenen Pakete entschlüsselt werden können, falls welche verworfen wurden [14 S. 93], da der Inhalt des z. B. ersten Pakets von KNX leicht zu erraten ist. Somit genügt uns das Abhören eines validen verschlüsselten Datenpakets. Durch die Mitsendung eines selbst erstellten IVs – da dieser mit dem entschlüsselten Text per XOR verknüpft wird – führt dies zu einem entsprechenden von einem Angreifer kontrolliertem Klartext. Verwunderlich ist ebenso, dass in diesem Anhang von KNX RF und einer Sequenznummer von 32 Bit Größe gesprochen wird. Der Anhang B scheint jedoch keine Anwendung in der Spezifikation zu finden, obwohl an einer Stelle auf ihn verwiesen wird [14 S. 15]. Warum dieser Abschnitt dennoch in dem Entwurf verblieben ist, ohne auf die hier gemachten Fehler hinzuweisen ist, nicht nachvollziehbar.

Fazit zum Entwurf

Als Referenz zur Spezifikation wurden Paper von der NIST [70] [71] sowie der RFC3610 [72] verwendet. Durch die inkonsistenten und teils widersprüchlichen Angaben in dem vorliegenden Entwurf ist eine Bewertung schwierig. Positiv ist die Verwendung einer grundsätzlich gut erprobten und sicheren Verschlüsselung. Dass die Implementierung auf Basis der oben erwähnten Dokumente erfolgen soll, ist ebenfalls positiv zu bewerten. Ebenso die Verwendung von Sequenznummern wodurch Frische[13] eingebracht wird. Problematisch ist jedoch die an vielen Stellen inkonsistente

[13] Durch Frische (engl. freshness) kann der Empfänger die Aktualität prüfen und somit alte bzw. bereits empfangene Pakete beispielsweise verwerfen. Durch Frische können Replay-Angriffe unterbunden werden.

Beschreibung sowie das teilweise Abweichen von den Beschreibungen und Empfehlungen der NIST. Mehrere Beispiele wurden in den vorherigen Absätzen genannt. Die Verwendung einer gekürzten MAC ist sehr kritisch zu betrachten: Falls die Datenrate deutlich erhöht wird, kann der Schutz der Authentizität als schwierig zu gewährleisten angesehen werden.

Ergebnis

Durch KNX-Transceiver Chips und einem Mikrocontroller ist es möglich, ein sich durch den Bus versorgendes Gerät zu bauen. Dieses kann u. a. dazu verwendet werden, nicht KNX Data Security fähige Geräte mit KNX Data Security abzusichern. Ebenso können exponierte Strecken abgesichert werden. Darüber hinaus ist es möglich, den KNX-Bus zu attackieren, zu stören oder z. B. zu segmentieren, indem das Gerät als „Linien-Koppler“ verwendet wird. Diese Möglichkeiten sind nicht implementiert. Im Rahmen dieser Arbeit wurde nur die Möglichkeit (PoC) gezeigt, dass eine solche White Box grundsätzlich und mit vertretbarem Aufwand gebaut werden kann – eine Anleitung dazu ist im Anhang D zu finden.

Mit KNXCalibur wurde bereits vor über 10 Jahren ein Gerät entworfen, welches umfangreiche Analysen oder Attacken auf den KNX-Bus ermöglicht. Die Kommunikation ist ebenfalls seit Längerem z. B. über ein TPUART-Board und einem Raspberry Pi unter Verwendung von knxd möglich. Jedoch benötigen all diese Geräte eine extra Spannungsversorgung oder können nicht auf einem Mikrocontroller betrieben werden. Mit Freebus wäre dies zwar grundsätzlich möglich, jedoch ist der Quellcode nicht frei verfügbar und somit nicht anpassbar. Des Weiteren stehen Angaben auf der Website im direkten Widerspruch zur verwendeten Softwarelizenz.

Wesentliche Vorteile zu bereits verfügbaren Lösungen sind die einfache Integrierbarkeit der White Box, der einfache und günstige Nachbau sowie die Offenheit (wozu auch die Dokumentation zählt) dar. Das in künftiger Arbeit zu erfüllende Ziel bestehende Geräte „KNX Data Security“-fähig zu machen oder damit (Teil-)Strecken abzusichern, wurde bisher anderweitig weder realisiert noch betrachtet.

Im Rahmen dieser Arbeit wurde ebenfalls KNX Data Security analysiert. Mir ist bisher keine weitere Analyse bekannt, dennoch sind die hier gewonnen Erkenntnisse vermutlich nicht vollständig. Ebenso sind diese nur vorläufig, da die Analyse auf einem Entwurf basiert.

KNX Data Security verwendet CCM in 2 Modi, entweder mit Authentizität und Verschlüsselung oder nur mit Authentizität. Der Entwurf hinterlässt gemischte Gefühle. Einerseits macht er einen soliden Eindruck, da zur Implementierung auf die Dokumente z. B. von der NIST verwiesen wird. Andererseits werden Begriffe durcheinandergeworfen und der Entwurf ist nicht immer klar. Des Weiteren ist der Modus zur Gewährleistung der Authentizität praktisch angreifbar, wobei der Zeitaufwand, abhängig vom Aufbau des KNX Netzwerks, jedoch hoch ist. Trotzdem ist von der Verwendung dieses Modus aus meiner Sicht abzuraten.

Analysen von KNX Data Security Geräten waren im Rahmen dieser Arbeit nicht möglich, da zum Zeitpunkt dieser Arbeit keine Geräte verfügbar waren.

Anhang A

KNX Drossel

Zur Erweiterung des KNX-Busses benötigen wir die Möglichkeit eine KNX-Drossel nachzubauen. Anlaufstelle hierfür waren das Freebus Projekt sowie das Mikrocontroller.net Forum. Das Freebus Projekt stellt eine DIY Anleitung zum Bau eines KNX-Netzteils zur Verfügung. Ein Teil dieses Aufbaus ist somit die KNX Drossel, diese ist jedoch vermutlich nur für 1-2 Geräte empfehlenswert [51]. Im Mikrocontroller.net Forum wurde eine Drossel nachkonstruiert und der Forumsbenutzer „MaWin“ vereinfachte diese. Der Nutzer Jörg S. wiederum simulierte diese mit Spice und stellte die Datei für seine Simulation zur Verfügung [73]. Anzumerken ist, dass die Medium Attachment Unit (MAU) (für diese Arbeit reicht es zu wissen, dass die MAU der Teil des z. B. TPUART ist, welcher die Signale auf den Bus erzeugt bzw. auswertet) für die Simulation (Abbildung 7 und Abbildung 8) stark vereinfacht wurde.

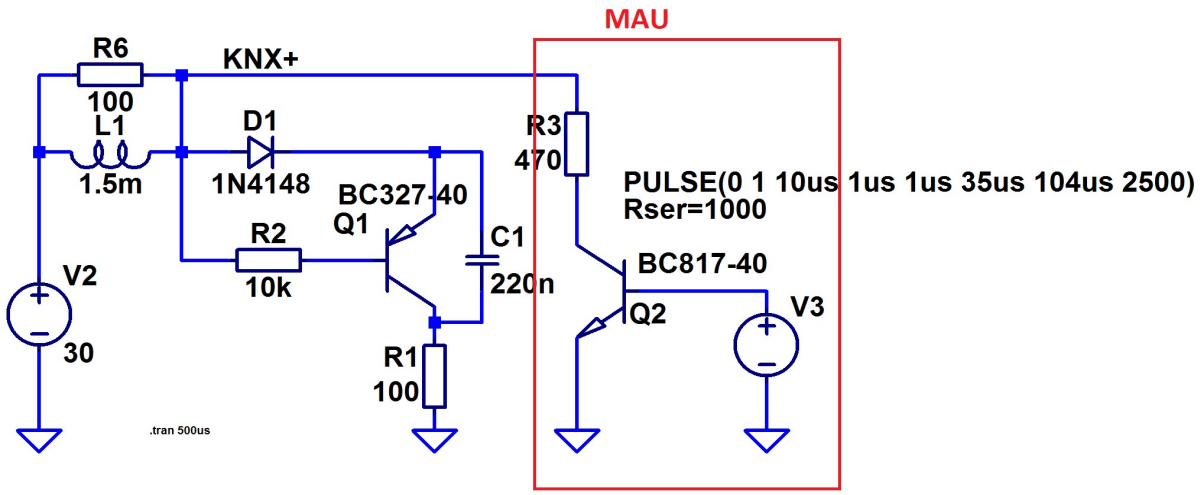


Abbildung 7 - Drosselaufbau („MaWin“) mit leichten Änderungen

In Abbildung 7 ist der Simulationsaufbau dargestellt, dieser wurde leicht modifiziert. Die Induktivität wurde von 2,5 mH auf 1,5 mH gesenkt. Dies ändert die Simulation nur marginal – die 1,5 mH Induktivität war schlicht schneller zu beschaffen.

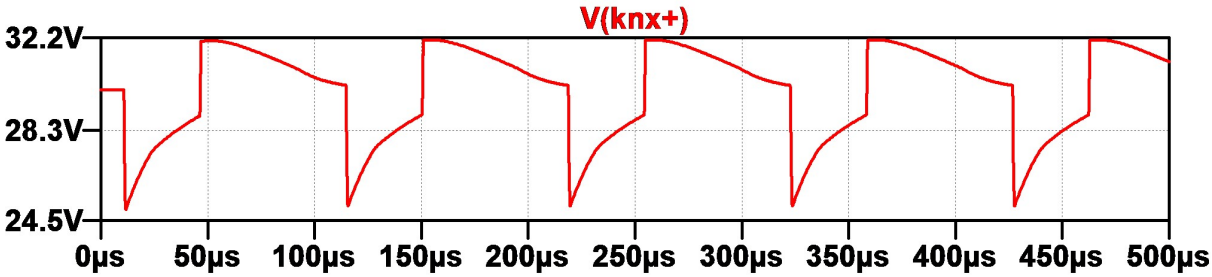


Abbildung 8 - Drosselsimulation (MaWin) mit leichten Änderungen

In Abbildung 8 ist der Spannungsverlauf dargestellt. Die Y-Achse beschreibt die Spannung, die X-Achse stellt die Zeit dar.

Die simulierten Spannungsverläufe der analogen Signale verstoßen dabei nicht gegen die Spezifikationen einer „0“ [15 S. 9]. Somit wurde die KNX-Drossel nachgebaut und das Signal mit Hilfe eines Oszilloskops (DSO 138 von JYE Tech [74]) unter Verwendung einer echten MAU (KNX Taster BE-TA5508.01 von MDT [75]) überprüft. Die Ergebnisse waren positiv, sie lagen deutlich in den Grenzen der Spezifikation[14] und wurden einwandfrei interpretiert wie in der Abbildung 9 und Abbildung 10 zu sehen ist. Zu beachten ist, dass nach dem Prinzip LSB First übertragen wird.

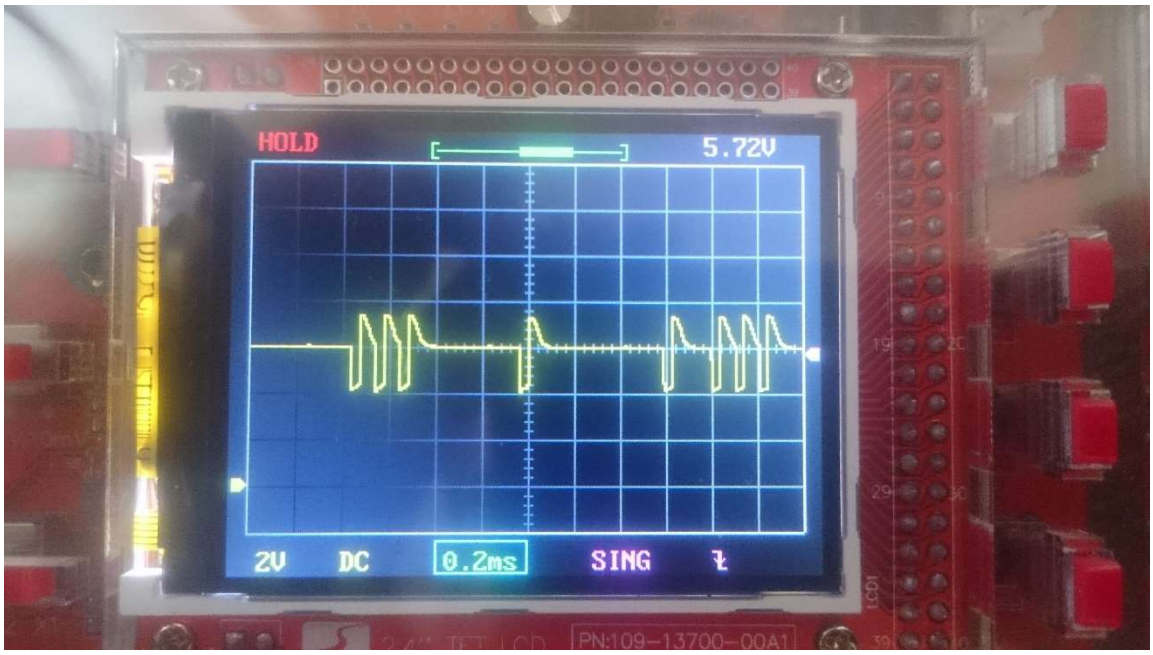


Abbildung 9 – Oszillatorwerte – Bitte Fußnote beachten[15]

Abbildung 10 - Per TPUART2 im Busmonitor Mode erhaltene Daten zu dem obigen analogen Signalverlauf

Die Bauteile – alle durchweg Standardbauteile, wie sie häufig verbaut werden – sind gut verfügbar und die Schaltung ist einfach nachzubauen.

[14] Anzumerken ist jedoch, dass die Spezifikation einen höheren Spannungsanstieg empfiehlt.

[15] Die angezeigte Spannung des Oszillators ist nicht korrekt. Das Verhältnis jedoch schon, womit 8 V/div die korrekte Angabe wäre. *Der Fehler liegt an einer zu hohen Spannung zwischen V- und GND, häufig ein Problem bei Fälschungen des DSO138.*

Erweiterte KNX Drossel

Hier wird kurz die Simulation zur erweiterten KNX Drossel behandelt. Die ursprüngliche Drossel stammt vom Benutzer „MaWin“ bzw. die Spice-Simulation vom Benutzer „jörg-s“ aus dem Mikrocontroller.net-Forum [53] und wurde leicht modifiziert. Ziel war es aus einem bestehenden Bus einen zweiten Bus zu erzeugen wobei die Signale, des einen, sich nicht auf dem anderen übertragen dürfen. In der Abbildung 11 ist der Simulationsschaltplan dargestellt.

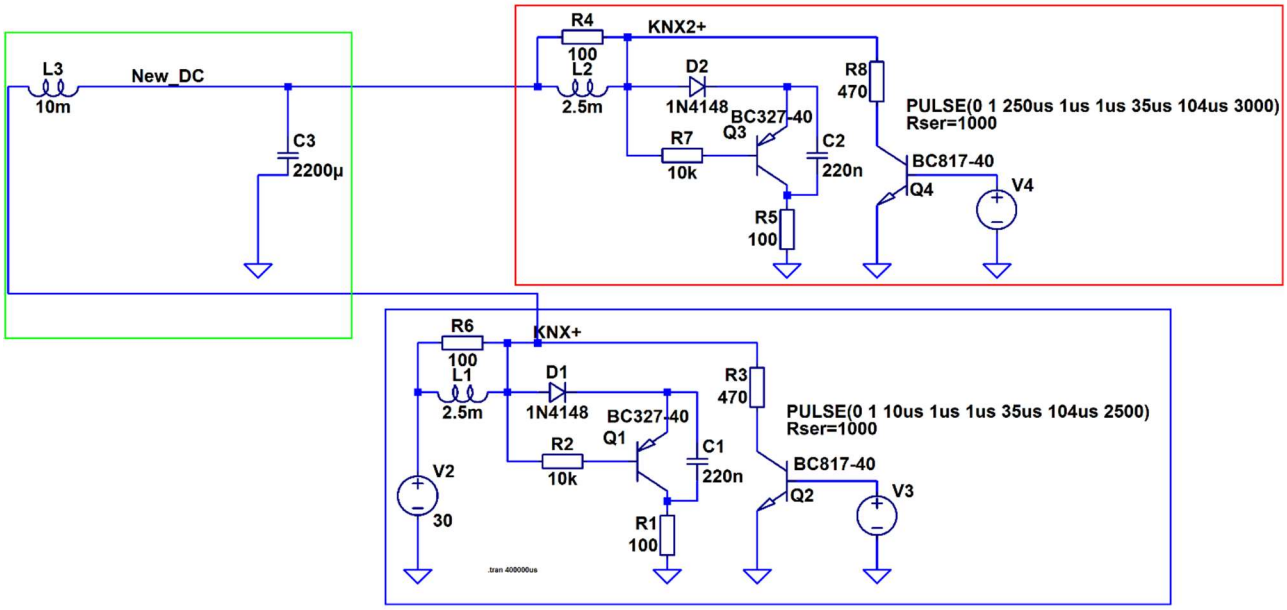


Abbildung 11 - Schaltplan Simulation zweiter KNX-Bus

Die folgenden Abbildungen zeigen die Signal- bzw. Spannungsverläufe von den oben beschriebenen KNX-Bussen an. Wobei der zweite KNX-Bus (V(knx2+)) aus dem ersten KNX-Bus (V(knx+)) gespeist wird. Die Y-Achse Zeit die Spannung an. V(new_dc) zeigt die Spannung nach dem LC-Glied an. Die X-Achse stellt die Zeit dar. Für die Simulation wurde dabei eine hohe Belastung des Busses gewählt.

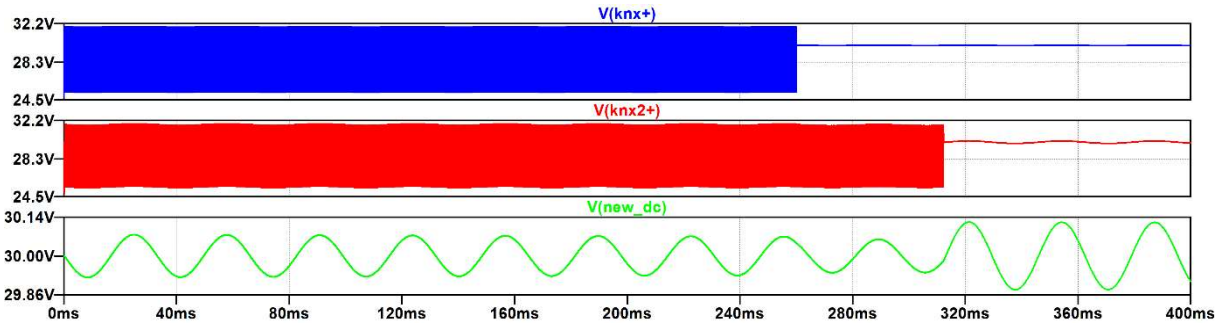


Abbildung 12 - Simulation des Spanungsverlaufs über die Zeit - 400ms

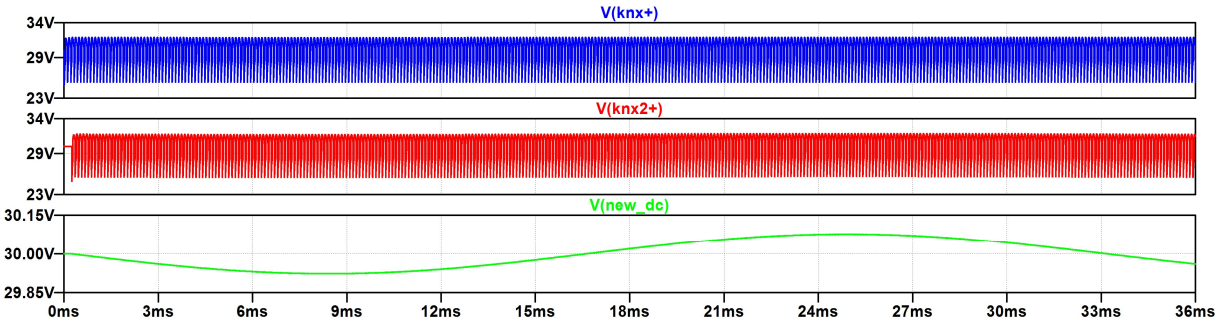


Abbildung 13 - Simulation des Spannungsverlaufs über die Zeit - 36ms

In Abbildung 14 sind die Signalverläufe gut erkennbar. Diese sind innerhalb der KNX-TP1 Spezifikation. In der Realität wird es eine solche Belastung des KNX-Busses vermutlich nicht geben. Diese wurde jedoch extrem gewählt um zu erkennen ob der zweite KNX-Bus ausreichend stabil ist und das LC-Glied gut gewählt wurde, so dass keine Signale auf den zweiten KNX-Bus durchschlagen.

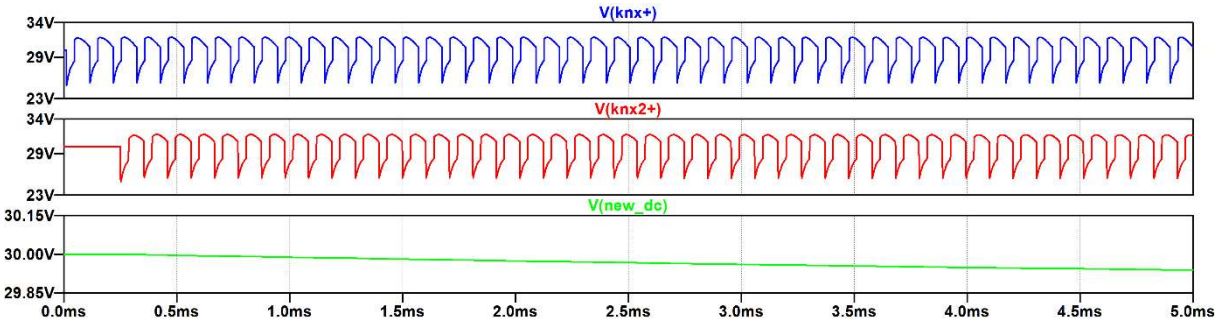


Abbildung 14 - Simulation des Spannungsverlaufs über die Zeit - 5ms

Die Schaltung kann gegebenenfalls durch eine Suppressordiode und eine Feinsicherung erweitert werden. Auf diese Weise können schadhafte Einflüsse auf dem zweiten KNX-Bus – welcher möglichweise exponiert ist – Schaden von dem ersten abhalten bzw. deutlich verringern.

Anhang B

Diskurs zur Verwendung des TPUART (2,2+)

Folgend werden die ersten Schritte bei der Realisierung der Hardware beschrieben, welche zur Wahl des BTM2-PCB geführt haben.

Der TPUART2(+) [35] ist nur im schwer zu lötendem QFN36 Packaging verfügbar. Somit wurde recherchiert ob der Arbeitsaufwand und somit die Verwendung vertretbar ist. Als Einleitung zum verlöten, wurde sich an einem YouTube-Video orientiert [76]. In diesem wird Heißluft verwendet. Bei diesem Versuch zerstörte ich vermutlich den Chip – dies stellte sich später raus. Für eine potenzielle Masterarbeit ist ein alternativer Versuch, unter Verwendung der Tipps und Alternativen von Mikrocontroller.net [77], denkbar.

Durch ein Multimeter wurde verifiziert, dass keine Lötbrücken zwischen den Pins entstanden sind. Die Funktion war jedoch nicht gegeben und der Defekt wurde erst mit Hilfe eines Logikanalyzer ersichtlich. Somit wurde im weiterem das BTM2 PCB verwendet. Dieses enthält den TPUART2 im Referenzdesign.

Somit wird empfohlen den TPUART2 im fertigen Referenzdesign zu beziehen. Dies konterkariert jedoch die Bemühung die White Box möglichst preisgünstig zu erstellen – der Preis beträgt ungefähr 40€ [78]. Da es sich nur um die Entwicklung eines PoC handelt, wird dies jedoch als vertretbar erachtet.

Stromaufnahme unter Verwendung einer externen 3,3 V Spannungsquelle

Bei der Messung hinter einer 3,3V Spannungsquelle wird nur ein aktiver UART-Port mit 57600 Bauds (mit einem USB UART Interface mit einem PC verbunden) verwendet. Real gemessen wurden aufgerundet 15,7 mA (unter Verwendung des Schaltplan Reduzierter KNX Proof of Concept mit 3,3 Volt aus dem Anhang C) in der Abstufung 20mA mit dem Etekcity Measure Up MSR-R500 welches laut Datenblatt „DC Current: 2000µA/20mA ±(1%+2), 200mA ±(1.2%+2), 10A ±(2%+5) [59]“ als Messfehler hat. Der Messfehler beträgt somit: I_abw = 0,2 * 15,68 mA + 0,02 mA = 3,156 mA bzw. gerundet 3,2 mA (Der Messfehler durch die Temperaturabweichung wird ignoriert, da keine Angaben vorhanden sind) [60]. Somit liegt die reale Stromaufnahme bei 15,7 ± 3,2 mA.

Es wurden keine Stromsparmechanismen des ATXMEGA verwendet. Durch Verwendung dieser oder durch die Reduzierung des Taktes auf 2 MHz, könnte deutlich Energie eingespart werden. Durch die Verwendung eines 2 MHz Taktes würde sich der Strombedarf deutlich senken. Laut Datenblatt auf, von den anfangs max. 12 mA, auf max. 1,4 mA [58 S. 117-118], sodass wir mit Peripherie bei vermutlich unter 10mA landen sollten: Dies würde es übrigens ermöglichen den TPUART zu verwenden. Da es dessen Strombudget aber fast komplett ausnutzt, wären wir bei Erweiterungen und evtl. Änderungen eingeschränkt.

Probleme bei der Realisierung

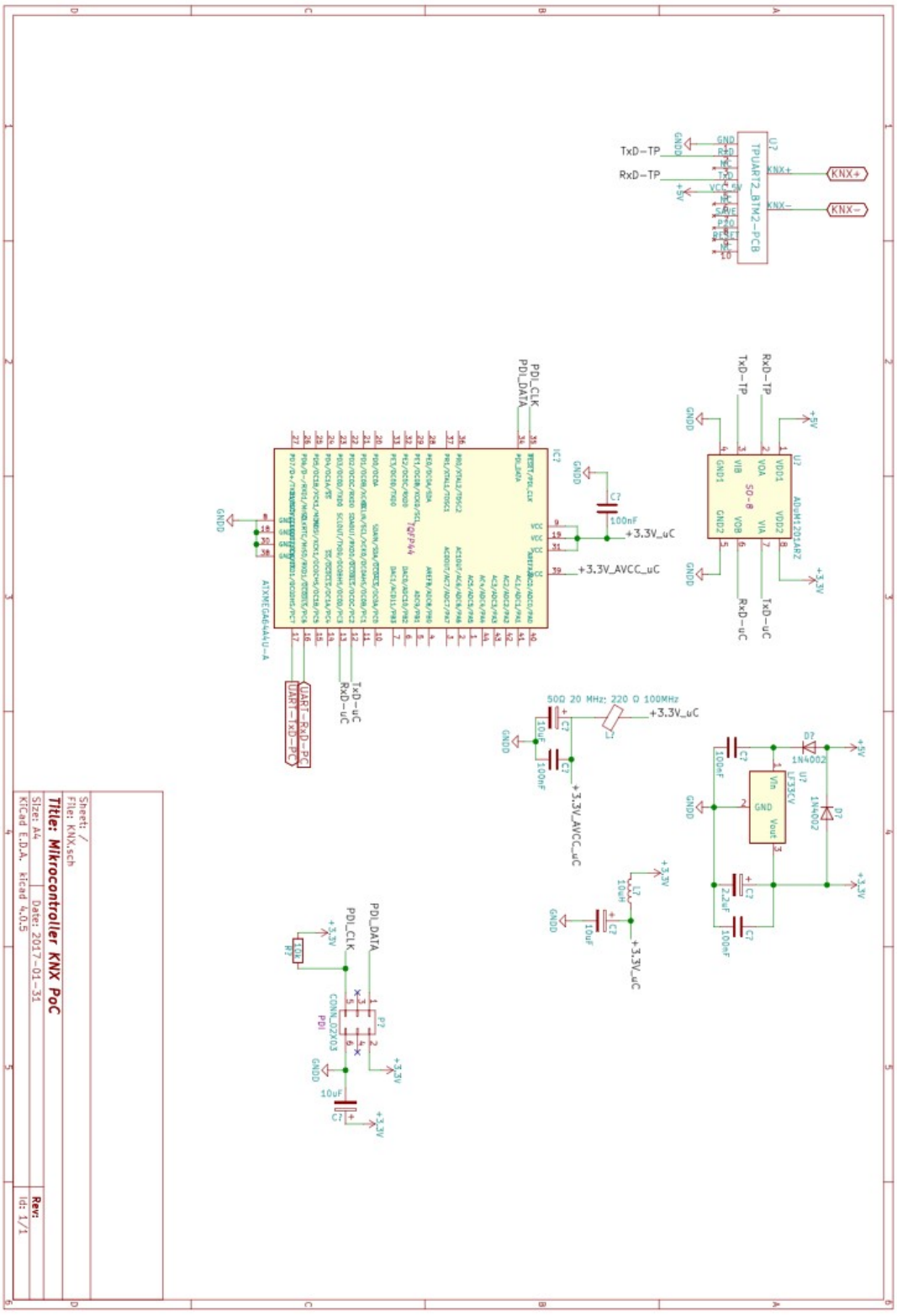
Bereits angesprochen wurden die Probleme beim Löten des TPUART2. Hier wurde zur Fehleranalyse der Logic Analyzer Logic 4 von Saleae verwendet [79]. Die passende Software dazu kennt bereits viele Protokolle, u. a. UART. Durch Analyse der Logikpegel konnte ich somit feststellen, dass der Chip – vermutlich durch die Hitze beim Löten – nicht mehr funktionsfähig war.

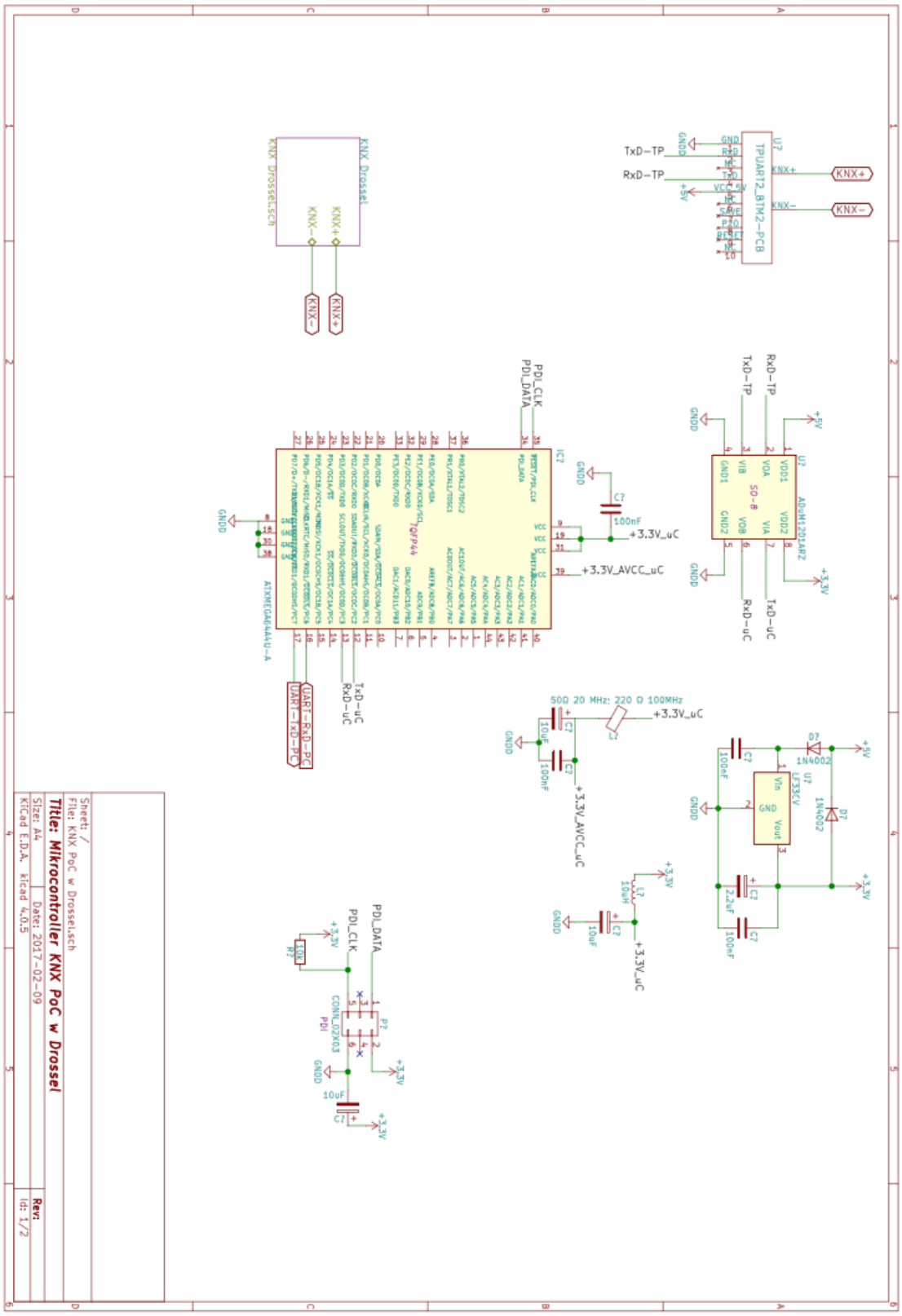
Ein weiteres Problem waren die Datenblätter vom TPUART und TPUART2(+) von Siemens. Es wurde im Datenblatt des TPUART in der „General Description“ angegeben, dass dieser 3,3V liefert [25 S. 1], dies ist jedoch nicht der Fall wie später im Datenblatt ersichtlich [25 S. 5 und 23]. Der TPUART 2(+) liefert jedoch beides [35 S. 7 und 15]. Hinzukamen die teilweise sehr kurz gehaltenen Beschreibungen, vor allem die des Busmonitor Modus wodurch die Implementierung stark erschwert wurde.

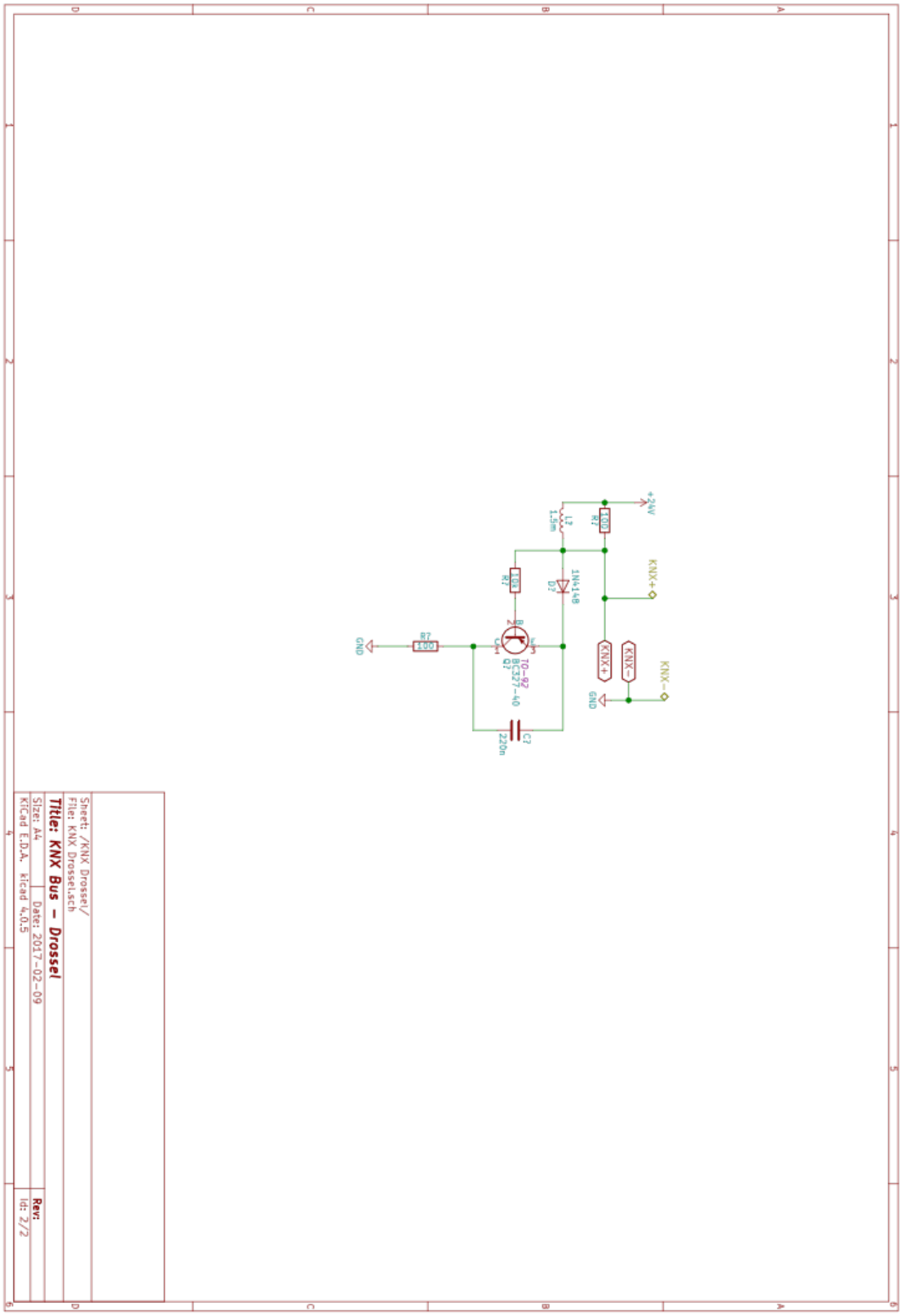
Anhang C

Alle Schaltpläne sind ebenfalls als Schematic-Datei auf CD enthalten.

Mikrocontroller KNX Proof of Concept - Schaltplan




KNX Drossel.sch
KNX Drossel
KNX-
KNX+
TxD-TP
RxD-TP
+5V
TPUART2_BTM2-PCB
KNX+
KNX-
PDI_CLK
PDI_DATA
100nF
+3.3V_uC
+3.3V_AVCC_uC
TxD-uC
RxD-uC
ADuM1201ARZ
1N4002
LF33CV
500 20 MHz: 220 Ω 100MHz
10uF
2.2uF
10uH
CONN_02X03
PDI
UART-RxD-PC
UART-TxD-PC
Sheet: /
File: KNX PoC w Drossel.sch
Title: Mikrocontroller KNX PoC w Drossel
Size: A4
Date: 2017-02-09
KiCad E.D.A. kicad 4.0.5
Rev:
Id: 1/2

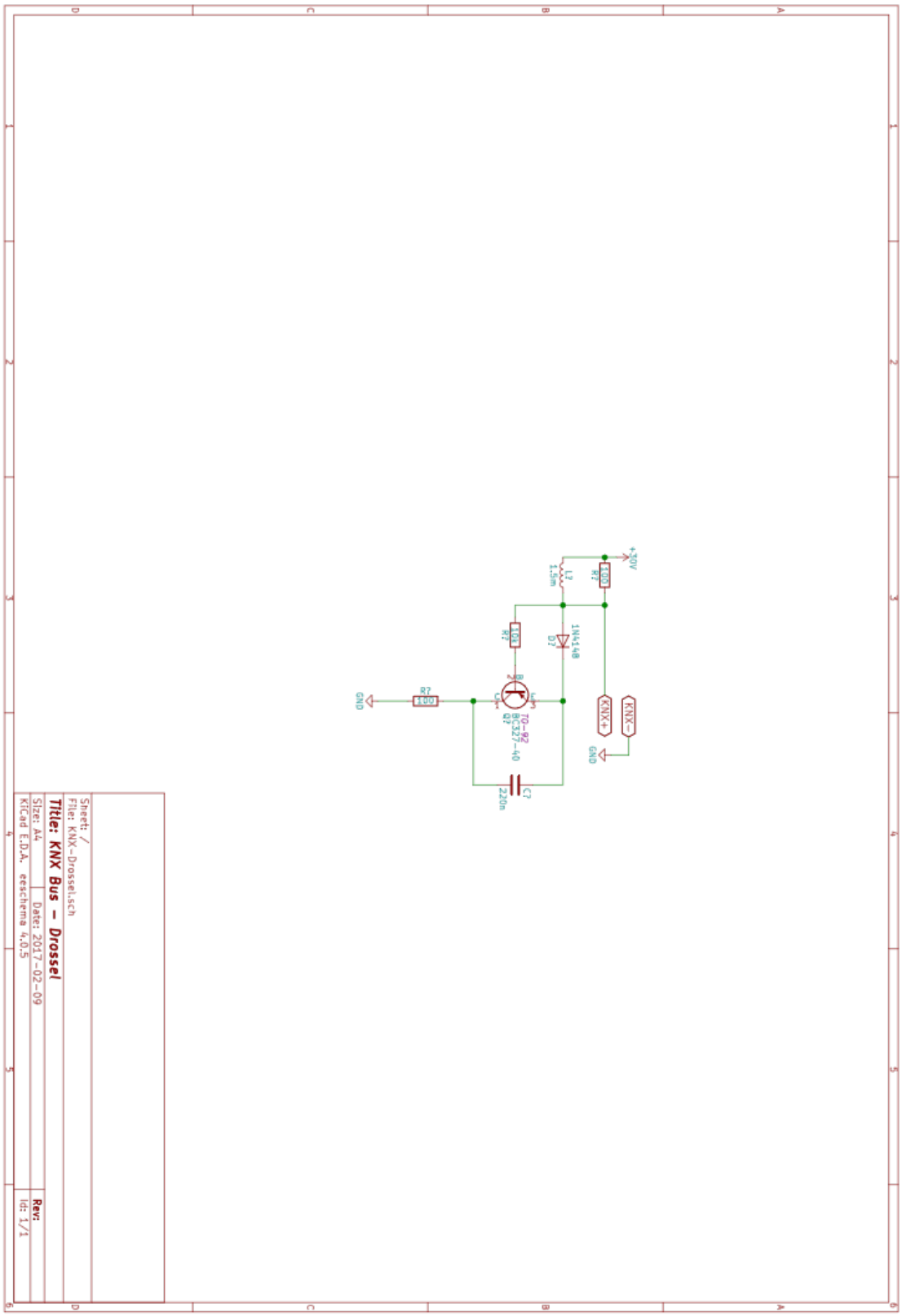


Sheet: /KNX Drossel/
File: KNX Drossel.sch

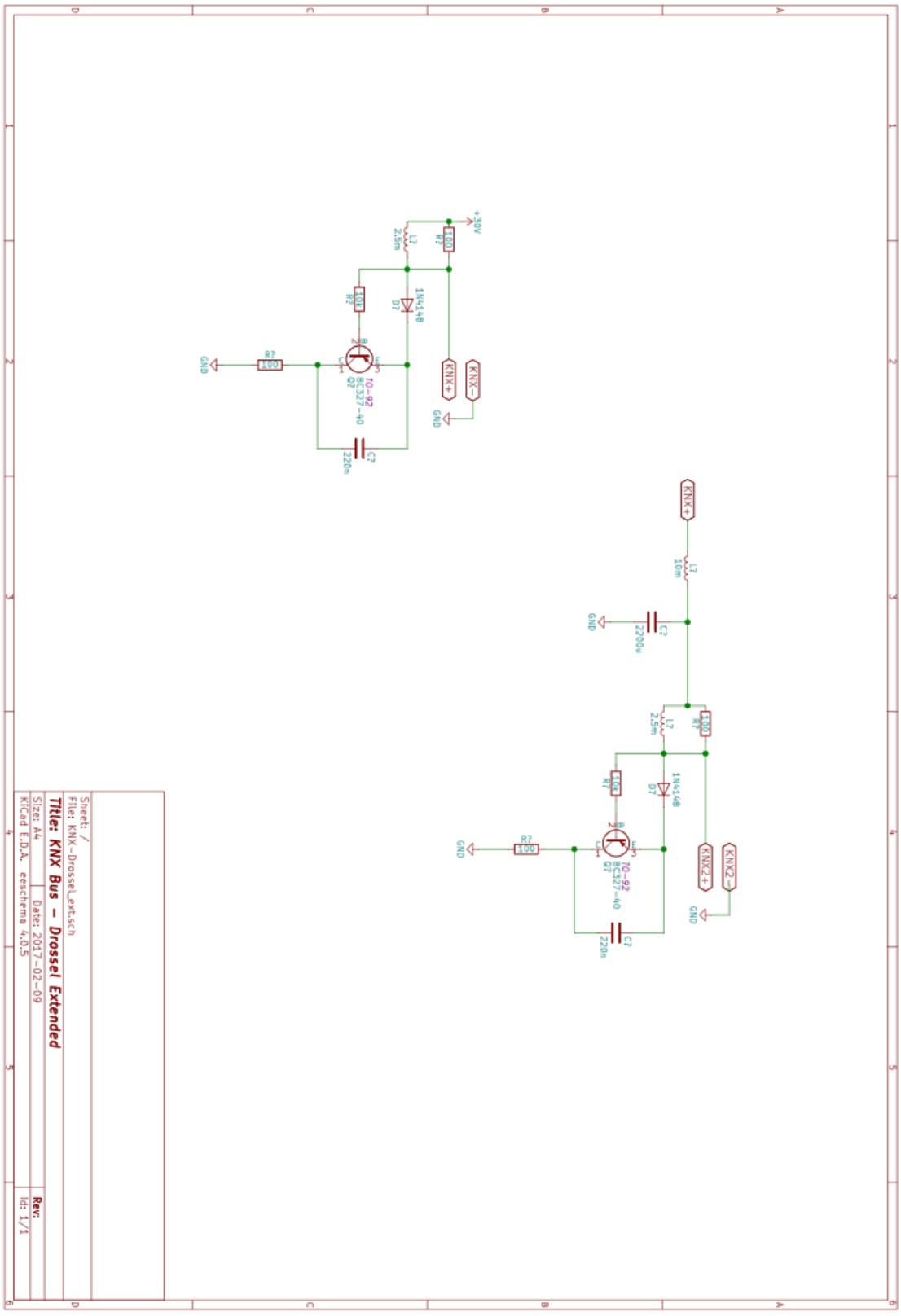
Title: KNX Bus – Drossel

Size: A4 | Date: 2017-02-09 | Rev:
KiCad E.D.A. kicad 4.0.5 | Id: 2/2

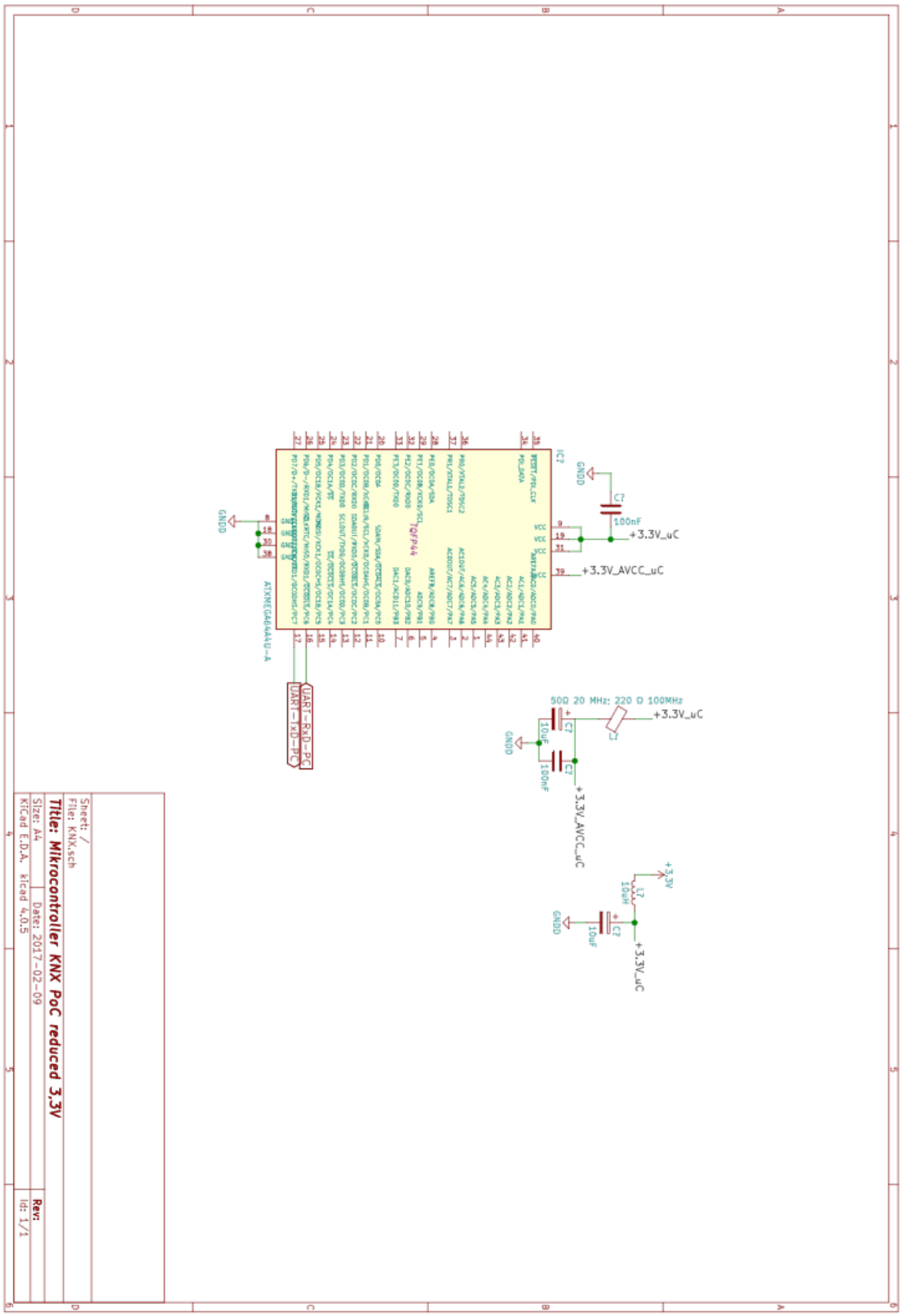
KNX Drossel – Nachbau - Schaltplan



Erweiterte Drossel - Schaltplan



Reduzierter KNX Proof of Concept mit 3,3 Volt



Anhang D

Anleitung zum Aufbau des Proof of Concepts

Im Anhang C findet sich der Schaltplan für den Proof of Concept. Falls kein KNX Netzteil vorhanden ist, ist ebenfalls ein Schaltplan mit selbstgebauter Drossel dort zu finden. Abbildung 15 zeigt ein Beispielaufbau. Unten links ist ein UART USB Wandler zu sehen und unten rechts das PDI Datenkabel zum Atmel ICE [80]. Diese sind in dem Schaltplan nicht explizit aufgeführt – es sind jeweils die entsprechenden Anschlüsse gekennzeichnet.

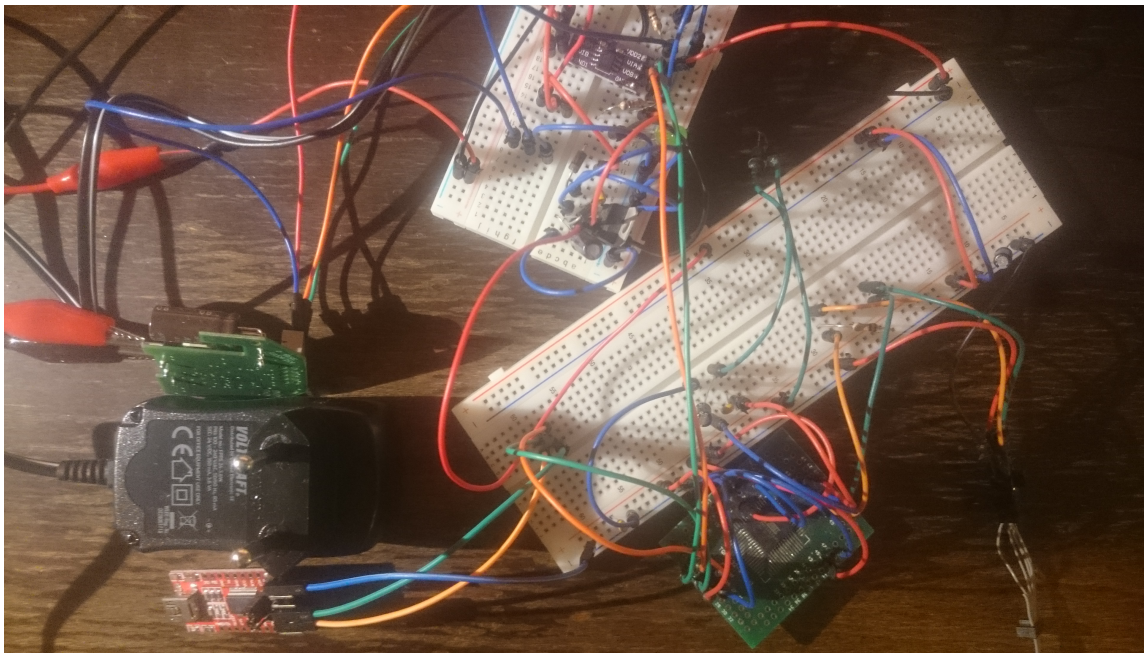


Abbildung 15 - KNX Bus mit Drossel Proof of Concept – Beispielaufbau

Atmel Studio kann unter [81] bezogen werden. Zum Zeitpunkt dieser Arbeit war die Version 7.0.1188 aktuell. Atmel Studio bietet eine komplette Toolchain an und unterstützt den Atmel ICE. Das Atmel Studio Projekt ist auf der CD enthalten. Eine Veröffentlichung ist geplant. In Abbildung 16 ist Oberfläche von Atmel Studio zu sehen. Oben mittig ist ein Dropdown Menü zu erkennen durch welches zwischen „Release“ und „Debug“ gewechselt werden kann. Damit werden ggfs. die entsprechenden Makros aktiv.

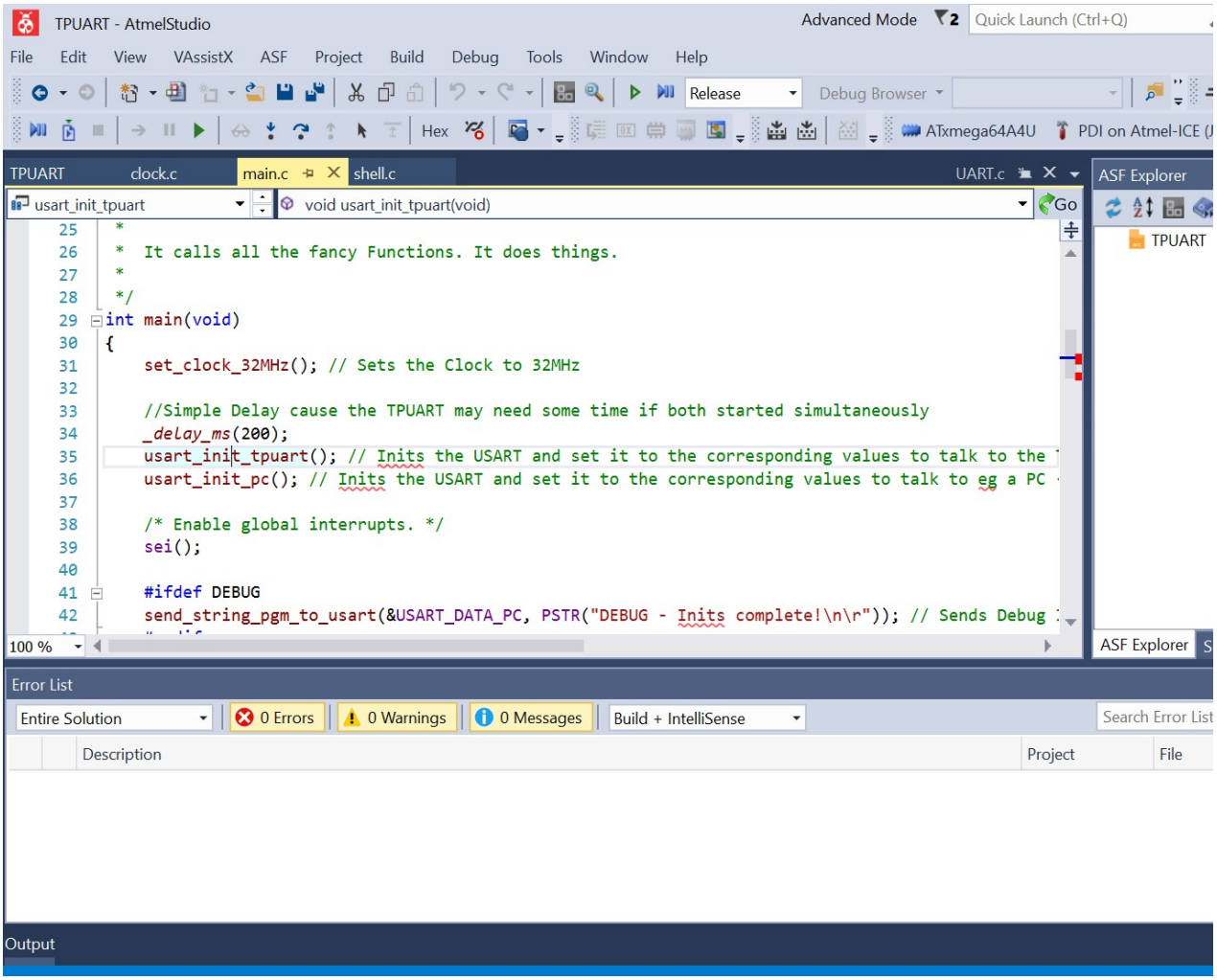


Abbildung 16 - Atmel Studio

Nach der Programmierung kann sich über eine serielle Konsole z. B. über Putty mit dem Mikrocontroller (über ein UART nach USB Wandler) verbunden werden. Die vordefinierten Einstellungen sind folgende:

- 8 Bit ist die Zeichengröße
- Keine Parität
- 1 Stoppbit
- Die Baudrate beträgt 57600 Baud

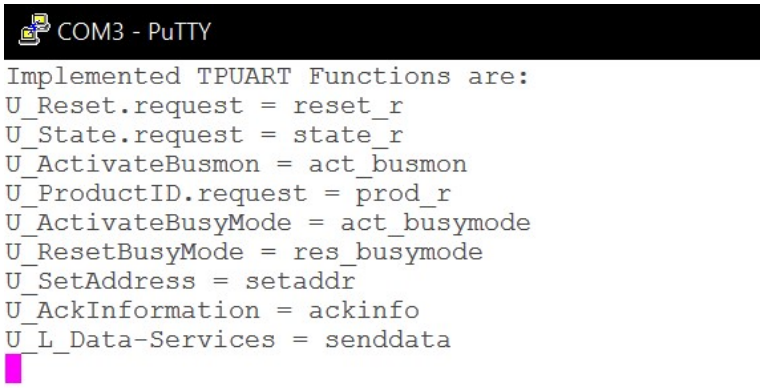


Abbildung 17 - UI Hilfe ohne Debugoutput

In Abbildung 17 ist der Output der Hilfe des UIs[16] zu sehen. Das Debugmakro war bei der Kompilierung und Programmierung nicht aktiv. Zu beachten ist, dass zur Zeit die eingegebenen Zeichen nicht zurück gesendet werden.

In Abbildung 18 ist beispielhaft der „U_SetAddress“-Dienst aufgerufen wurden. Als Adresse wurde 1.1.4 festgelegt. In Abbildung 19 wurde mit aktiviertem Debugmakro kompiliert – als Adresse wurde ebenfalls 1.1.4 verwendet.

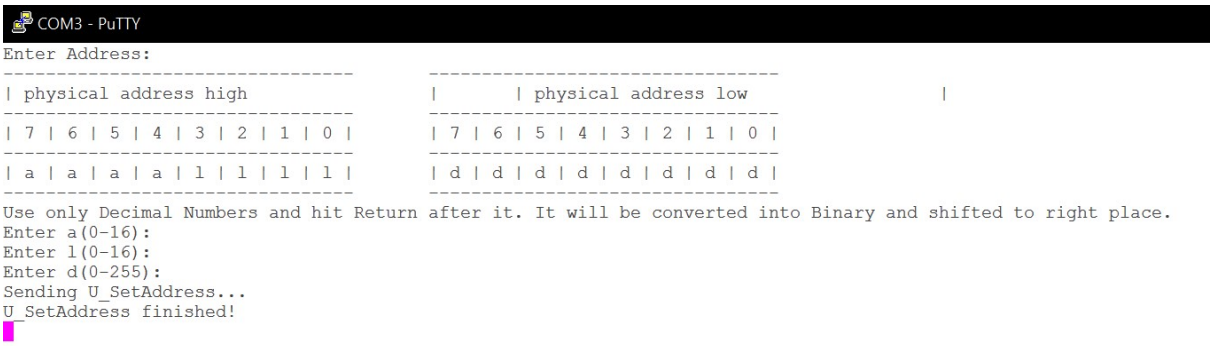


Abbildung 18 - UI – Verwendung des „U_SetAddress“-Services – ohne Debugmakro

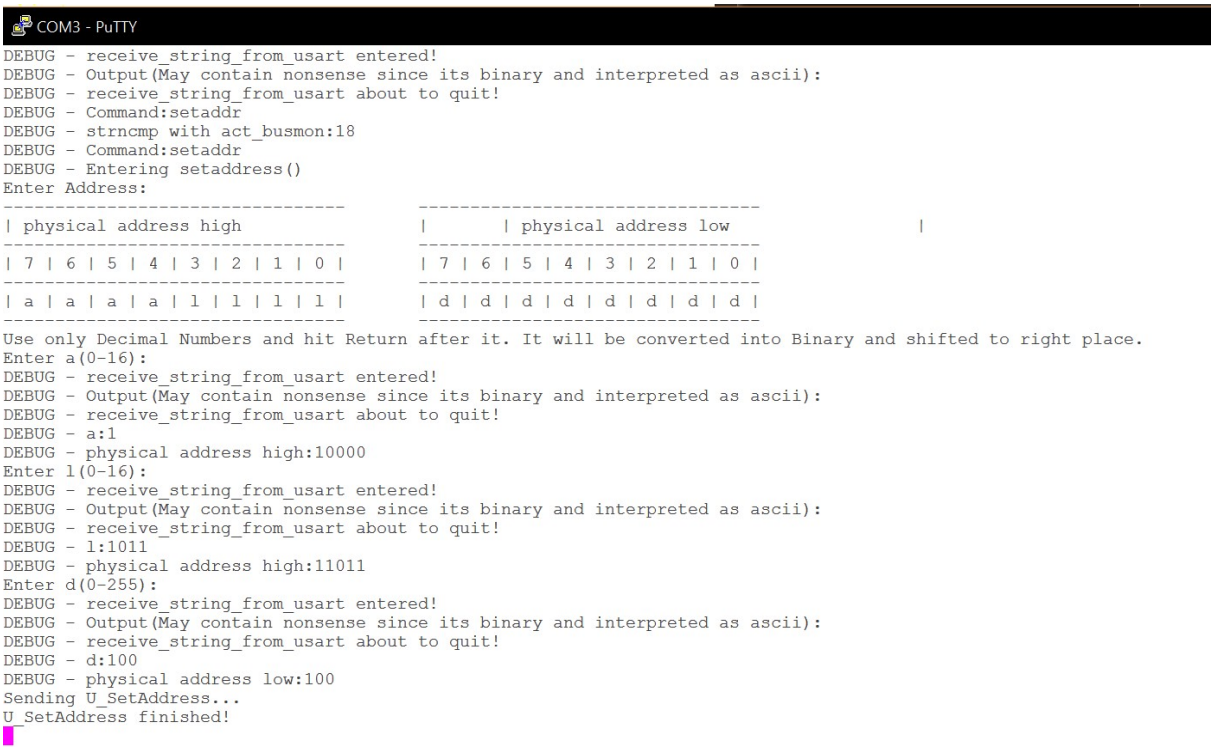


Abbildung 19 - UI – Verwendung des „U_SetAddress“-Services – mit Debugmakro

[16] User Interface

Anhang E

Doxygen Beispiel

Abbildung 20 stellt exemplarisch ein Teil der durch Doxygen erstellten Dokumentation dar.

Function Documentation

§ set_clock_32MHz()

void set_clock_32MHz (void)

This function inits, calibrates and changes the clock.

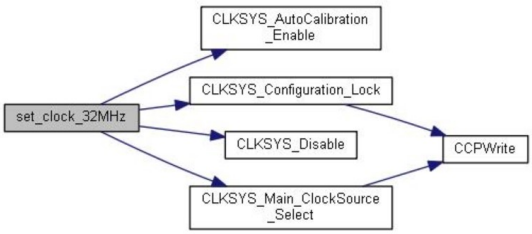
It enables the 32kHz ref Clock for the Autocalibration and also executes the calibration. Afterwards it enables and changes to the 32MHz internal Clock and also disables the 2MHz & 32kHz clocks. Eventually it Locks the current Clock config.

Definition at line **16** of file **clock.c**.

References **CLKSYS_AutoCalibration_Enable()**, **CLKSYS_Configuration_Lock()**, **CLKSYS_Disable()**, **CLKSYS_Enable**, **CLKSYS_IsReady**, and **CLKSYS_Main_ClockSource_Select()**.

Referenced by **main()**.

Here is the call graph for this function:



Here is the caller graph for this function:

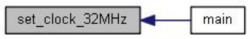


Abbildung 20 – Doxygen - „set_clock32MHz()“-Methode

Literaturverzeichnis

1. Wikipedia. Wikipedia - KNX Standard Ger. [Online] 13. Dezember 2016. [Zitat vom: 26. 12 2016.] https://de.wikipedia.org/w/index.php?title=KNX-Standard&oldid=160597220.

2. —. Wikipedia - KNX Standard#Technik des Netzes Ger. [Online] 13. Dezember 2016. [Zitat vom: 26. 12 2016.] https://de.wikipedia.org/w/index.php?title=KNX-Standard&oldid=160597220#Technik_des_KNX-Netzes.

3. KNX Association cvba. KNX.org - What is KNX. [Online] 20. Oktober 2014. [Zitat vom: 09. Januar 2017.] https://www.knx.org/knx-en/knx/association/what-is-knx/index.php.

4. —. KNXA - Newsletter - Specs free of Charge. [Online] Januar 2016. [Zitat vom: 11. Januar 2017.] https://www.knx.org/knx-en/newsletters/newsletters/january2016/.

5. —. KNX - ETS 5.5.0 - Info. [Online] 12. April 2016. [Zitat vom: 09. Januar 2017.] https://www.knx.org/media/docs/ETS-version-info/version-5-5-0_de.pdf.

6. Thoma, Jörg. Golem - KNX Hack. [Online] 2015. März 24. [Zitat vom: 30. Dezember 2016.] http://www.golem.de/news/knx-schwachstellen-spielen-mit-den-lichtern-der-anderen-1503-113085.html.

7. Antago GmbH. Antago GmbH. [Online] 20. Juni 2014. [Zitat vom: 30. Dezember 2016.] https://www.antago.info/nc/news/aktuelle-meldungen/detail/erebos-antagos-gott-der-dunkelheit-eibknx-hacking/.

8. Beuth, Patrick. Zeit - KNX Hack. [Online] 7. August 2014. [Zitat vom: 30. 12 2016.] http://www.zeit.de/digital/datenschutz/2014-08/black-hat-2014-hotelzimmer-gehackt.

9. Westermeir, Günter. *Diversitäre Zugangs- und Sicherheitsmechanismen angewendet in automatisierten Gebäuden.* [PDF-Dokument] Technische Universität München - Lehrstuhl für Messsystem- und Sensortechnik : s.n., 22. Januar 2004.

10. Granzer, Wolfgang. *Security in Networked Building Automation Systems.* [PDF-Dokument] Vienna University of Technology, Institute of Computer Aided Automation, Automation Systems Group : s.n., 2005.

11. Seem, Patrick. Texas Instruments - Application Note AN067. [Online] 2008. [Zitat vom: 10. Januar 2017.] http://www.ti.com/lit/an/swra234a/swra234a.pdf.

12. Wikipedia. Wikipedia - EIB#Geschichte Ger. [Online] 26. Oktober 2016. [Zitat vom: 26. 12 2016.] https://de.wikipedia.org/w/index.php?title=Europ%C3%A4ischer_Installationsbus&oldid=159102928#Geschichte.

13. —. Wikipedia - Kleinspannung. [Online] 18. Januar 2017. [Zitat vom: 21. Januar 2017.] https://de.wikipedia.org/w/index.php?title=Kleinspannung&oldid=161753975.

14. KNX Association. *KNX Data Security - Application Note 158/13 v02.* [PDF-Dokument] 2013.

15. —. *KNX - System Specifications - Communication Media - Twisted Pair 1.* [PDF-Dokument] 2012.

16. —. *KNX - System Specifications - Communication - Data Link Layer General.* 2013.

17. —. *KNX - System Specifications - Interworking - Datapoint Types.* [PDF-Dokument] 2013.

18. —. *KNX - Basic and System Components/Devices – Minimum Requirements – Standardised solutions - Tests KNX System Conformance Testing - BCUs and BIMs - BCUs.* [PDF-Dokument] 2013.

19. Automation Systems Group. https://www.auto.tuwien.ac.at. [Online] [Zitat vom: 13. Januar 2017.] https://www.auto.tuwien.ac.at/downloads/knxcalibur/.

20. KNX Association cvba. KNX - KNX Secure Checklist. [Online] 03. Mai 2016. [Zitat vom: 22. Januar 2017.] https://www.knx.org/media/docs/Flyers/KNX-Secure-Checklist/KNX-Secure-Checklist_en.pdf.

21. —. KNX - KNX Secure Position Paper. [Online] 23. August 2015. [Zitat vom: 22. Januar 2017.] https://www.knx.org/media/docs/downloads/Marketing/Flyers/KNX-Secure-Position-Paper/KNX-Secure-Position-Paper_en.pdf.

22. —. KNX - KNX Secure for Developers. [Online] 24. August 2016. [Zitat vom: 22. Januar 2017.] https://www.knx.org/knx-en/Landing-Pages/KNX-Secure/KNX-Secure-for-Developers/index.php.

23. Guggenberger, Florian. Automation Systems Group. [Online] 21. Dezember 2008. [Zitat vom: 13. Januar 2017.] https://www.auto.tuwien.ac.at/bib/pdf_TR/TR0143.pdf.

24. Praus, Fritz und Kastner, Wolfgang. *A versatile networked embedded platform for KNX/EIB.* [PDF-Dokument] 2006.

25. Siemens AG. Opternus - TPUART Datasheet. [Online] August 2013. [Zitat vom: 10. Januar 2017.] http://www.opternus.com/uploads/media/TPUART1_Datenblatt_20130806.pdf.

26. Heesch, Dimitri van. Doxygen. [Online] 29. Dezember 2016. [Zitat vom: 23. Januar 2017.] http://www.stack.nl/~dimitri/doxygen/.

27. Busware. busware - PiGator OW. [Online] 15. Dezember 2016. [Zitat vom: 14. Januar 2017.] http://busware.de/tiki-index.php?page=PIG_OW.

28. —. busware - PiGator DIN. [Online] 07. Oktober 2015. [Zitat vom: 14. Januar 2017.] http://busware.de/tiki-index.php?page=POD.

29. eLinux - RPI Header. [Online] 03. Juli 2015. [Zitat vom: 14. Januar 2017.] http://elinux.org/index.php?title=RPi_Low-level_peripherals&oldid=383831.

30. Busware. busware - Shop - PIM TPUART. [Online] [Zitat vom: 14. Januar 2017.] http://shop.busware.de/product_info.php/cPath/1_32_33/products_id/105.

31. Broadcom Europe Ltd. Raspberry PI - BCM2836 Peripherals. [Online] 06. Februar 2012. [Zitat vom: 14. Januar 2017.] https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf.

32. Heinze, Robert. FHEM RPI3 - BT UART. [Online] 01. Januar 2017. [Zitat vom: 14. Januar 2017.] https://wiki.fhem.de/w/index.php?title=Raspberry_Pi_3:_GPIO-Port_Module_und_Bluetooth&oldid=18196.

33. Siemens AG. Siemens AG - Datenblatt BTM 117/12 PCBA. [Online] Mai 2012. [Zitat vom: 11. Januar 2017.] http://www.hqs.sbt.siemens.com/cps_product_data/gamma-b2b/PCBA_UP117-12_datasheet.pdf.

34. —. Siemens AG - TPI - BTM UP117/12 5WG1. [Online] April 2012. [Zitat vom: 23. Januar 2017.] http://www.hqs.sbt.siemens.com/cps_product_data/data/de/tpi/1172ab12_tpi_de-2012-05-23.pdf.

35. —. Opternus KNX EIB TP-UART 2+ - IC Datasheet. [Online] August 2013. [Zitat vom: 03. 01 2017.] http://www.opternus.com/uploads/media/TPUART2__Datenblatt_20130806.pdf.

36. Analog Devices. Analog Deivces - Datenblatt ADuM1201ARZ. [Online] September 2016. [Zitat vom: 11. Januar 2017.] http://www.analog.com/media/en/technical-documentation/data-sheets/ADuM1200_1201.pdf.

37. FreeBus Team. FreeBus - Grundschaltung. [Online] 10. Juni 2007. [Zitat vom: 10. Januar 2017.] https://freebus.org/content/freebus-grundschaltung.

38. —. FreeBus Git. [Online] [Zitat vom: 24. Januar 2017.] https://git.freebus.org/.

39. Mikrocontroller.net - Forum - Freebus = Closedbus. [Online] [Zitat vom: 24. Januar 2017.] https://www.mikrocontroller.net/topic/234486.

40. FreeBus Team. FreeBus - Starten mit FREEBUS. [Online] 21. Januar 2012. [Zitat vom: 24. Januar 2017.] https://freebus.org/content/starten-mit-freebus.

41. Kögler, Martin. EIBd. [Online] [Zitat vom: 23. Januar 2017.] https://www.auto.tuwien.ac.at/~mkoegler/index.php/eibd.

42. —. TU Wien - AUTO - Kögler - BCU SDK Doc. [Online] 06. März 2011. [Zitat vom: 23. Januar 2017.] https://www.auto.tuwien.ac.at/~mkoegler/eib/sdkdoc-0.0.5.pdf.

43. Urlichs, Matthias. GitHub - KNXd. [Online] [Zitat vom: 11. Januar 2017.] https://github.com/knxd/knxd.

44. Schifferle, Christian und Thomas, Martin. Mikrocontroller.net - AVR GCC Tutorial. [Online] [Zitat vom: 25. 12 2016.] https://www.mikrocontroller.net/articles/AVR-GCC-Tutorial#Die_Nutzung_von_sprintf_und_printf.

45. The Calimero Project. Github - Calimero. [Online] [Zitat vom: 11. Januar 2017.] https://calimero-project.github.io/.

46. Malinowsky, Boris, Neugschwandtner, Georg und Kastner, Wolfgang. Github - Calimero - NG. [Online] 2007. [Zitat vom: 11. Januar 2017.] https://github.com/calimero-project/introduction/blob/master/documentation/calimero-ng.pdf.

47. Malinowsky, Boris. Calimero - License. [Online] 28. März 2015. [Zitat vom: 11. Januar 2017.] https://github.com/calimero-project/calimero-core/blob/master/LICENSE.txt.

48. Haumacher, Bernhard, Neugschwandtner, Georg und Malinowsky, Boris. Calimero - Sourceforge. [Online] 14. Februar 2016. [Zitat vom: 11. Januar 2017.] https://sourceforge.net/p/calimero/wiki/Home/.

49. openHAB UG. OpenHAB - Introduction. [Online] [Zitat vom: 23. Januar 2017.] http://www.openhab.org/introduction.html.

50. GitHub - OpenHAB - KNX Binding. [Online] 19. Oktober 2016. [Zitat vom: 23. Januar 2017.] https://github.com/openhab/openhab1-addons/wiki/KNX-Binding/1ca08f6cf8918b0d14f08651814d33e700ebdb4a.

51. FreeBus Team. Freebus - Drossel. [Online] 11. Mai 2012. [Zitat vom: 10. Januar 2017.] https://freebus.org/comment/24#comment-24.

52. —. FreeBus - Git - Netzteil Teileliste. [Online] 15. November 2009. [Zitat vom: 10. Januar 2017.] https://git.freebus.org/freebus.git/blob/63e81f6f2d7b383699c9a2316dba6e914800be2f:/hardware/module/Netzteil/Teileliste.txt.

53. S., Jörg. Mikrocontroller.net Forum - KNX Drossel. [Online] 11. Juli 2012. [Zitat vom: 10. Januar 2017.] https://www.mikrocontroller.net/topic/264362#2751281.

54. Hagemeyer, Andreas und Bonkhoff, Ingolf. Google Patents - KNX Drossel. [Online] 12. November 2014. [Zitat vom: 10. Januar 2017.] https://www.google.com/patents/EP2802100A1?cl=en.

55. STMicroelectronics. STMicroelectronics - Datenblatt LF33CV. [Online] 03. März 2015. [Zitat vom: 10. Januar 2017.] http://www.st.com/content/ccc/resource/technical/document/datasheet/c4/0e/7e/2a/be/bc/4c/bd/CD00000546.pdf/files/CD00000546.pdf/jcr:content/translations/en.CD00000546.pdf.

56. Schaerer, Thomas. Elektronik Kompendium - Linearregler. [Online] 26. Juni 2014. [Zitat vom: 11. Januar 2017.] http://www.elektronik-kompendium.de/public/schaerer/ureg3pin.htm.

57. Atmel Corporation. Atmel - AVR1012. [Online] 09. März 2010. [Zitat vom: 24. Januar 2017.] http://www.atmel.com/Images/doc8278.pdf.

58. —. Atmel XMEGA A4U Datasheet. [Online] September 2014. [Zitat vom: 03. Januar 2017.] http://www.atmel.com/images/Atmel-8387-8-and16-bit-AVR-Microcontroller-XMEGA-A4U_Datasheet.pdf.

59. Etekcity. Etekcity MSR-R500. [Online] [Zitat vom: 03. Januar 2017.] http://www.etekcity.com/product/100111.html.

60. Wikipedia. Wikipedia - Messfehler Digitalmultimeter. [Online] 11. Juni 2016. [Zitat vom: 03. 01 2017.] https://de.wikipedia.org/w/index.php?title=Digitalmultimeter&oldid=155191600#Berechnung_der_Fehlergrenze.

61. Atmel Corporation. Atmel ATXMEGA64A4U Parameters. [Online] [Zitat vom: 03. Januar 2017.] http://www.atmel.com/devices/ATXMEGA64A4U.aspx?tab=parameters.

62. —. Atmel - AVR1000: Getting Started Wiriting C-code for XMEGA. [Online] 23. Februaur 2008. [Zitat vom: 28. Januar 2017.] http://www.atmel.com/images/doc8075.pdf.

63. —. Atmel - AVR4027: Tips and Tricks to Optimize Your C Code for 8-Bit AVR Microcontrollers. [Online] 10. November 2011. [Zitat vom: 28. Januar 2017.] http://www.atmel.com/Images/doc8453.pdf.

64. —. Atmel - AVR1307: Using the XMEGA USART - PDF. [Online] 23. Februar 2008. [Zitat vom: 28. Januar 2017.] http://www.atmel.com/Images/doc8049.pdf.

65. —. Atmel - AVR1307: Using the XMEGA USART - Quellcode. [Online] 05. November 2008. [Zitat vom: 28. Januar 2017.] http://www.atmel.com/images/AVR1307.zip.

66. —. Atmel - AVR1003: Using the XMEGA Clock System - PDF. [Online] 22. Juli 2016. [Zitat vom: 28. Januar 2017.] http://www.atmel.com/Images/Atmel-8072-Using-the-XMEGA-Clock-System_ApplicationNote_AVR1003.pdf.

67. —. Atmel - AVR1003: Using the XMEGA Clock System - Quellcode. [Online] 14. September 2009. [Zitat vom: 28. Januar 2017.] www.atmel.com/images/AVR1003.zip.

68. FreeBus Team. FreeBus - LPC Controller. [Online] 16. Dezember 2011. [Zitat vom: 10. Januar 2017.] https://freebus.org/content/lpc-controller-version-343-4te-0.

69. KNX Association. *KNX - System Specifications - Communication - Application Layer.* [PDF-Dokument] 2013.

70. Dworkin, Morris. NIST - Recommendation for Block Cipher Modes of Operation - Methods and Techniques. [Online] Dezember 2001. [Zitat vom: 20. Januar 2017.] http://dx.doi.org/10.6028/NIST.SP.800-38A.

71. —. NIST - Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. [Online] Mai 2004. [Zitat vom: 30. Januar 2017.] http://dx.doi.org/10.6028/NIST.SP.800-38C.

72. Whiting, Doug, Housley, Russell und Ferguson, Niels. IETF - RFC3610 - Counter with CBC-MAC (CCM). [Online] September 2003. [Zitat vom: 30. Januar 2017.] https://tools.ietf.org/html/rfc3610.

73. MaWin und S., Jörg. Mikrocontroller.net Forum - KNX Drossel allg. [Online] 2012. [Zitat vom: 25. Januar 2017.] https://www.mikrocontroller.net/topic/264362.

74. JYE Tech Ltd. JYE Tech Ltd. - DSO138 User Manual Rev 05. [Online] 01. März 2016. [Zitat vom: 25. Januar 2017.] http://www.jyetech.com/Products/LcdScope/UserManual_138_new.pdf.

75. MDT technologies GmbH. MDT - Taster Datenblatt. [Online] 18. Februar 2016. [Zitat vom: 25. Januar 2017.] http://www.mdt.de/download/MDT_DB_Taster.pdf.

76. *YouTube - How to Solder QFN MLF chips Using Hot Air without Solder Paste and Stencils.* [Video]. 2010.

77. Mikrocontroller.net - SMD Löten. [Online] 02. Januar 2017. [Zitat vom: 24. Januar 2017.] https://www.mikrocontroller.net/wikisoftware/index.php?title=SMD_L%C3%B6ten&oldid=94729.

78. Opternus Components. Opternus Shop - TPUART2 Board BTM2-PCB. [Online] [Zitat vom: 24. Januar 2017.] http://www.opternus.com/de/siemens/entwicklungs-werkzeuge/tp-uart2-board-btm2-pcb.html.

79. Saleae Inc. saleae.com. [Online] 2017. [Zitat vom: 28. Januar 2017.] https://www.saleae.com/.

80. Atmel Corporation. Atmel - Connecting to a PDI Target. [Online] [Zitat vom: 10. Februar 2017.] http://www.atmel.com/webdoc/atmelice/atmelice.connecting_pdi.html.

81. —. Atmel - Atmel Studio. [Online] [Zitat vom: 09. Februar 2017.] http://www.atmel.com/microsite/atmel-studio/.

82. Future Technology Devices International Limited. FTDI Chip - FT232R Datasheet. [Online] 18. November 2015. [Zitat vom: 14. Januar 2017.] http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf.

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 13.02.2017 .....