Bernburg
Dessau
Köthen

Hochschule Anhalt
Anhalt University of Applied Sciences

emw
Fachbereich
Elektrotechnik, Maschinenbau
und Wirtschaftsingenieurwesen

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Engineering (B. Eng.)

Marcus Emmerich

Vorname Nachname

Elektrotechnik und
Informationstechnik, 2010, 4052419

Studiengang, Matrikel, Matrikelnummer

Thema:

Entwicklung der Nachrüstlösung einer Brandmeldeanlage zur Anbindung und Visualisierung an das KNX System

Prof. Dr. M. Brutscheck

1. Prüfer /Vorsitzender der Bachelorprüfungskommission

Prof. Dr. M. Schnöll

2. Prüfer

24.10.2014

Abgabe am

Selbstständigkeitserklärung

Hiermit erkläre/n ich/wir, dass die Arbeit selbständig verfasst, in gleicher oder ähnlicher Fassung noch nicht in einem anderen Studiengang als Prüfungsleistung vorgelegt wurde und keine anderen als die angegebenen Hilfsmittel und Quellen, einschließlich der angegebenen oder beschriebenen Software, verwendet wurden.

Köthen, 23.10.2014

Ort, Datum | Unterschrift/en der/des Studierenden

Sperrvermerk

Sperrvermerk: ja [ ] nein [X]

wenn ja: Der Inhalt der Arbeit darf Dritten ohne Genehmigung der/des (Bezeichnung des Unternehmens) nicht zugänglich gemacht werden. Dieser Sperrvermerk gilt für die Dauer von X Jahren.

Köthen, 23.10.2014

Ort, Datum | Unterschrift/en der/des Studierenden

Angaben zum Unternehmen

Logo des Unternehmens	
Name des Unternehmens	EAB – G. Sandow GmbH
Name des Betreuers	Dominique Lieder

Kontaktdaten

Anschrift des Standortes, an dem die Arbeit verfasst wurde	EAB – G. Sandow GmbH Handwerkerstraße 2, 06847 Dessau-Roßlau
E-Mail-Adresse des Betreuers	lieder@eab-sandow.de

Kurzfassung

Diese Arbeit beschreibt die Nachrüstlösung einer Brandmeldeanlage der Firma Notifier zur Anbindung und Visualisierung an das KNX System. Die Anbindung soll ermöglichen, dass Informationen der Brandmeldeanlage in Form von Feuer- und Statusmeldungen an dem KNX-Bus übertragen und visualisiert werden können. Die positiven Effekte, die daraus resultieren, sind zum einen die mögliche Verknüpfung mit anderen Teilnehmern im KNX-System und zum anderen die daraus resultierende Möglichkeit, alle vorhandenen Systeme in einem Gebäude visualisiert darstellen zu können. Nach der Recherche, zu dem von dem Systemen verwendeten Schnittstellen und Kommunikationsmöglichkeiten, wurde die Auswahl der benötigten Bauteile getroffen, um die Anbindung der beiden Systeme zu ermöglichen. Die daraus resultierende Verwendung eines Mikrocontrollers erforderte die Entwicklung und Programmierung dessen Software. Anschließend war es möglich durch hardware- und softwaretechnischen Lösungen eine Anbindung der beiden Systeme zu ermöglichen und das daraus gesteckte Ziel, eine Visualisierung durch Verwendung spezieller Software des KNX, zu erreichen.

Inhaltsverzeichnis

1	**Einleitung**	**1**
1.1	Einleitung	1
1.2	Motivation und Zielsetzung	2
1.3	Vorgehensweise	2
2	**Analyse**	**3**
2.1	KNX - System	3
2.1.1	Teilnehmer	3
2.1.2	Programmierung	3
2.1.3	Kommunikation	4
2.2	Brandmeldeanlage der Firma Notifier	5
2.2.1	Bestandteile einer Brandmeldeanlage	6
2.2.2	Schnittstellen der BMZ	7
2.2.3	Informationsdaten	10
2.3	Benötigte Systeme zur Realisierung der Aufgabe	12
2.4	Seriell zu KNX Gateway	16
2.4.1	Zennio SKX OPEN	16
2.4.2	KNX Gateway RS232/485-IP von „Elka Elektronik“	17
2.4.3	KNX-GW-RS232-RS485 von „arcus-eds“	17
2.5	Mikrocontroller	19
2.5.1	Auswahl des Mikrocontrollers/Atmega324A	19
2.5.2	Analyse des zu benutzenden Oszillators	22
2.5.3	Programmierung	23
2.5.4	Programmiergeräte und Evaluationsboards	24
2.6	RS232 zu TTL Wandler	28
3	**Realisierung der Aufgabenstellung**	**29**
3.1	Auswahl geeigneter Bauteile	29
3.1.1	Auswahl des RS232 zu KNX Gateways	29
3.1.2	Auswahl des Programmiergerätes	29
3.2	Anschluss aller Systeme	30
3.2.1	Anschlussverbindung des RS232/KNX Gateways mit dem KNX-System	30
3.2.2	Verbindung des RS232/KNX Gateways mit dem RS232/TTL Wandler	31
3.2.3	Verbindung des RS232/TTL Wandlers mit dem Mikrocontroller	32
3.2.4	Verbindung des RS232/TTL Wandlers mit der BMZ	34
3.2.5	Verbindung des externen Schwingquarzes mit dem Mikrocontroller	34
4	**Programmierung des Mikrocontrollers**	**35**
4.1	Programmablaufplan	35
4.2	Allgemeiner Aufbau des C-Programms	37
4.3	Empfang und Senden von Textstrings	37
4.3.1	UART Register	39
4.3.2	Initialisierung der UART Schnittstelle	43
4.3.3	Senden mit dem UART	46
4.3.4	Empfangen mit dem UART	47
4.3.5	Umsetzung im Hauptprogramm	49
4.4	Suchablauf der enthaltenen Statusmeldungen	49
4.5	Suchablauf der Meldernummer	52
4.6	Ausgabe der gefilterten Textinformationen	55

4.7 Programmierung mit Atmel Studio 6.2 56
4.7.1 Erstellung eines Projektes 57
4.7.2 Programmierung des externen Oszillators 58
4.7.3 Übertragung des Quellcodes auf den AVR 64

5 KNX-Gate2 65
5.1 Inbetriebnahme 65
5.2 Projektierung 66

6 Visualisierung mit „tebis KNX domovea“ 71
6.1 Das System 71
6.2 Konfigurationstool 72
6.3 Die Visualisierungs- und Bedienoberfläche (Client-Software) 73

7 Zusammenfassung und Ausblick 74

Anhang i
A. Filterprogramm des ATmega324A i

Abbildungsverzeichnis viii

Tabellenverzeichnis x

Literaturverzeichnis xi

Danksagung xiii

1 Einleitung

1.1 Einleitung

Unser Alltag hat sich verändert. Wir ziehen Bargeld aus dem Automaten, kaufen und verkaufen über das Internet, telefonieren rund um die Welt. Im Auto nutzen wir ein Navigationssystem, die Autotüren verriegeln und öffnen sich per Funk und das Innenraumlicht im KFZ schaltet sich an, sobald wir einsteigen. Kurzum, bei der Kommunikation, Unterhaltung oder im Auto heißt es seit geraumer Zeit: „Willkommen in der Zukunft“. Betrachtet man den Stand der Technik bei unseren Gebäuden, erblickt uns allerdings ein anderes Bild. Nicht selten lautet hier das Fazit: „Heimkommen in die Vergangenheit“. Elektroinstallationen haben in den meisten Gebäuden großen Nachholbedarf. In Autos sind vernetzte Sensoren und Aktoren längst Standard. Aufgrund der langen Lebensdauer von Gebäuden findet dieser Wandel allerdings nur verzögert statt. Was für einen großen Nutzen dieser Wandel bringen kann, zeigt sich vor allem in den Bereichen Betriebskosten, Sicherheit und Flexibilität. Dort könnte man durch ein intelligentes Gebäudesystem enorme Verbesserungen erzielen.
Durch intelligente Beleuchtungs- und Heizungssteuerungen, sowie durch Ausschalten von Steckdosen bzw. Schaltaktoren, womit Geräte komplett ausgeschaltet werden können und nicht weiter im Standby Modus verharren, kann eine Menge Energie und damit Kosten gespart werden. Sicherheitsaspekte wie dem einer Alarmanlage steht dem KNX-System ebenfalls kein Problem im Weg. Durch Benutzung eines Tablets besteht zusätzlich die Möglichkeit über visualisierte Darstellungen Informationen aller Teilnehmer im KNX-Netz anzeigen zu lassen und des Weiteren die Ansteuerung dieser aus einer großen Entfernung realisieren zu können. Man sieht allein an diesen kurzen Beispielen, welchen großen Vorteil dieses System mit sich bringen kann.
Dennoch sind gewisse Nachteile vorhanden. Ein grundwiegendes Problem herrscht z.B. in dem Bereich der Sicherheit, denn den wichtigsten Sicherheitsaspekt beherbergt in einem Gebäude der Brandschutz und dieser hat im KNX-System noch keine große Bedeutung gewonnen. Es gibt nur vereinzelte Unternehmen, die ihre Brandmeldesysteme in das KNX-System integriert haben und so eine Kommunikation mit anderen Teilnehmern im KNX-System ermöglichen (z.B. ESPP by Honeywell). Brandmeldeanlagen können selbst keine Brandentstehung verhindern oder einen Brand löschen. Ihre Aufgaben bestehen darin, entstehende Brände möglichst früh zu erkennen und diese dann so schnell wie möglich der zu Hilfe leistende Stelle (z.B. der Feuerwehr) zu informieren. Außerdem sollen Personen, die sich im Gebäude befinden, durch akustische Signale gewarnt werden, um so den Schutz der Betroffenen zu erhöhen. Man stelle sich vor, in einer Küche wurde gekocht und danach versehentlich der Herd angelassen. Durch Anbringung eines Wärmemelders über dem Herd, wird dieser im Falle eines Brandes ab einer bestimmten Temperatur ausgelöst. Dies hat zur Folge, dass die Personen im Haus akustisch informiert werden und des Weiteren die Feuerwehr alarmiert wird. Nach der Alarmierung bis hin zum Eintreffen der Feuerwehr verstreichen wertvolle Minuten, in denen sich das Feuer weiter ausbreiten kann. Durch die Erfahrung der Feuerwehrmänner stellt es kein größeres Problem dar, das Feuer zu löschen. Doch durch den erheblichen Zeitaufwand und durch die Verwendung einer großen Menge von Wasser, entsteht ein gravierender Schaden der Wohnung und damit hohe Kosten für den Bewohner.

1.2 Motivation und Zielsetzung

Es wäre doch vom großen Nutzen, wenn eine Brandmeldeanlage, schon bevor ein Feuer entsteht, diesen verhindern könnte. Durch eine Verknüpfung mit einer intelligenten Gebäudesteuerung wie dem KNX, könnten sich hierbei enorme Vorteile ergeben. Zum Beispiel wäre es durch eine Anbindung möglich, den vorherig genannten Brandfall zu verhindern, bevor dieser erst entsteht. Es besteht die Möglichkeit bei einer Brandmeldeanlage einen Voralarm auszulösen. Bei der Verwendung eines Wärmemelders ist dieser bei Auslösen eines Voralarms zuständig für die Benachrichtigung eines kritischen Temperaturanstieges, ohne gleich die Feuerwehr zu alarmieren. Durch eine mögliche Kommunikation zwischen der Brandmeldeanlage und dem KNX könnte man die Voralarm-Meldung des Wärmemelders nutzen, um den darunter liegenden Herd einfach auszuschalten. Damit verhindert man eine weitere Wärmeentstehung, die womöglich zu einen Brand führen könnte.

Des Weiteren soll durch die Verknüpfung der Brandmeldeanlage mit dem KNX System gewährleistet werden, dass eine Visualisierung aller Informationen der Brandmeldeanlage vorgenommen werden kann, sodass abgesehen von Feueralarm- und Voralarmmeldungen, auch Störungen einzelner Melder angezeigt werden können. Dies führt dadurch zur besseren Übersichtlichkeit aller im Haushalt befindlichen Teilnehmer, welches zum Beispiel für einen Facility Manager vom großen Nutzen sein kann, da dieser ständig einen kompletten Überblick aller sich im Gebäude befindlichen Geräte haben muss. Zur Visualisierung aller Informationen, die über ein Display oder Tablet angezeigt werden können, wird das „tebis KNX domovea“ verwendet, welches eine weitgehend vorkonfigurierte und damit besonders einfach zu installierende Lösung darstellt. Das „tebis KNX domovea“ soll in dieser Arbeit allerdings nur am Rande allgemein erläutert werden. Das Hauptaugenmerk liegt darin, eine Verknüpfung einer Brandmeldeanlage und dem KNX-System zu realisieren, um damit die Grundlage einer Visualisierung zu schaffen.

1.3 Vorgehensweise

Die Entwicklung der Nachrüstlösung einer Brandmeldeanlage der Firma Notifer zur Anbindung und Visualisierung an das KNX System lässt sich in mehreren Phasen einteilen. Zu aller erst soll eine Analyse der zu verwendeten Systeme getroffen werden. Dabei wird betrachtet, welche Systeme benötigt werden, um die Nachrüstlösung der Brandmeldeanlage bewerkstelligen zu können. Anschließend wird geschildert, wie die eigentliche Realisierung erfolgen soll. Dabei soll auf den genauen Aufbau des zu verwendeten Systems und aller ausgewählten Geräte eingegangen werden. Daraufhin erfolgt die eigentlich Umsetzung der Aufgabenstellung, in dem ein Mikrocontroller und die sogenannte KNX Gate2 Software programmiert werden. Im letzten Punkt wird dann noch ein kleiner Einblick verschafft, wie eine Visualisierung von KNX Geräten erfolgen kann.

2 Analyse

In diesem Kapitel geht es darum den Stand der Technik sowie die vorhandene Systeme zu betrachten und zu analysieren. Auf der Grundlage dieser Analyse wird dann ein System zur Lösung des Problems entwickelt. Des Weiteren werden die Anforderungen, die das System erfüllen soll, aufgestellt.

2.1 KNX - System

Der Konnex-Bus (KNX) ist ein Feldbus, der für den Einsatz in der Gebäudeautomatisierung entwickelt wurde. Es ist der einzige Standard für Gebäudesystemtechnik, der weltweite Zertifizierungsprogramme für Produkte, Schulungsstätten (private und öffentliche Einrichtungen) und sogar für Personen (Gebäudeplaner, Elektroinstallateure) anbietet. Das Ziel bestand dabei alle wichtigen Anlagen in der Gebäudetechnik steuern zu können und damit die einzelnen Gewerke miteinander zu verknüpfen. Alle Geräte benutzen das gleiche Übertragungsverfahren und können so über eine gemeinsame Busleitung Daten austauschen. Dies vereinfacht die Planung und Ausführung von Gebäudefunktionen und ermöglicht ohne Zusatzaufwand viel höhere Funktionalität, Flexibilität und mehr Komfort. KNX ist ein offener Standard, d. h. jeder Hersteller bzw. Entwickler hat vollen Zugriff auf alle notwendigen technischen Informationen, die er für die Weiterentwicklung benötigt. Allerdings erfordert dies die beitragspflichtige Mitgliedschaft in der offenen Vereinigung KNX Association. Daher wird kritisiert, dass dies kein wirklich offener Standard sei, da durch die Mitgliedschaft grundsätzlich Kosten entstehen. Erst wenn diese Mitgliedschaft auch kostenfrei ist, könne von einem „offenen Standard“ die Rede sein.

2.1.1 Teilnehmer

Beim KNX System wird neben den Systemgeräten (Spannungsversorgung, Busleitung usw.) generell zwischen Sensoren und Aktoren unterschieden. Sensoren sind Geräte, die im Gebäude Ereignisse erkennen (Über-/Unterschreitung eines Helligkeitswertes, Bewegungen, Temperaturwerte) und in Telegramme umwandeln. Diese Telegramme werden anschließend in das KNX Netz versendet, durch andere Geräte empfangen und die darin enthaltenen Befehle in Aktionen umgewandelt. Diese Befehlsempfänger werden als Aktoren bezeichnet. Durch eine dezentrale Struktur (jedes Gerät enthält einen eigenen Mikrocontroller) kann die Größe des Bussystems genau auf den Bedarf angepasst werden. Theoretisch können bis zu mehreren 10000 Teilnehmer in einer KNX Anlage miteinander kommunizieren.

2.1.2 Programmierung

Die Funktion der einzelnen Busteilnehmer wird durch ihre Programmierung bestimmt, die jederzeit verändert und angepasst werden kann. Die Programmierung der Teilnehmer erfolgt über eine Software, die von der KNX Association bereitgestellt wird. Diese Software ist ein herstellerunabhängiges Installationswerkzeug und lautet auf den Namen ETS (Engineering Tool Software). Ein Projektierer muss bei der Konfiguration der KNX Anlage folgende Schritte durchführen:

- Vergabe von physikalischen Adressen an die einzelnen Geräte (zur eindeutigen Identifikation eines Sensors oder Aktors in einer KNX-Anlage)
- Auswahl und Einstellung der passenden Anwendungssoftware für Sensoren und Aktoren (zugehörige Anwendungssoftware wird in das Bauteil geladen)
- Vergabe von Gruppenadressen (Zur Verwendung der Funktionen von Sensoren und Aktoren)

Ausgenommen von Spannungsversorgungen enthält jedes Busgerät bei der Projektierung eine eindeutige physikalische Adresse. Diese wird bei der Inbetriebnahme mit der ETS in den Teilnehmer geladen und dauerhaft in einem EEPROM („electrically erasable programmable read-only memory“) gespeichert.
Die physikalische Adresse, die einem Teilnehmer eindeutig zugeordnet wurde, gibt gleichzeitig Auskunft über seine topologische Lage im Gesamtsystem, und zwar durch die Angabe:

- Bereich.Linie.Teilnehmer

Nachdem jedem Teilnehmer eine physikalische Adresse zugewiesen wurde, kann jeder eindeutig im System identifiziert werden. Da das KNX System eine dezentrale Struktur aufweist (jedes Gerät enthält einen eigenen Mikrocontroller), muss jedes Gerät einzeln programmiert werden. Dabei muss eine Anwendungssoftware, die vom Hersteller entwickelt und von der KNX Accosiation zertifiziert werden muss, auf das Bauteil geladen. Die Anwendungssoftware wird dafür (in der Regel online im Internet) durch den jeweiligen Hersteller bereitgestellt und dann wiederum durch die ETS Software auf das Gerät übertragen. Nachdem die Übertragung von statten ging, müssen die einzelnen Funktionen eines Gerätes eingestellt (z.B. Helligkeit einer Lampe beim Drücken eines Tasters) und zusätzlich Gruppenadressen vergeben werden. Gruppenadressen werden dabei zur Kommunikation einzelner Teilnehmer benötigt.

2.1.3 Kommunikation

Die Kommunikation zwischen Sensoren und Aktoren erfolgt Hauptsächlich durch sogenannte Gruppenadressen. Damit zwei Teilnehmer im KNX-System miteinander kommunizieren können, müssen beide die gleiche Gruppenadresse zugewiesen bekommen haben. Um die Kommunikation im KNX-System näher erläutern zu können, betrachtet man eine einfache KNX-Anlage, welche einen Tastsensor und zwei Schaltaktoren beinhaltet. In Abbildung 2.1 soll das Beispiel näher erläutert werden. Beim betätigen des Tasters soll der Schaltaktor aktiviert und somit die angeschlossene Lampe zum Leuchten gebracht werden.
Detailliert besteht dieses KNX-System aus folgenden Komponenten:

- Eine Spannungsversorgung (29 V DC)
- Eine Drossel (kann auch in Spannungsversorgung (SV) eingebaut sein)
- Ein Tastsensor
- Zwei Schaltaktoren
- Busleitung (es werden nur zwei Adern benötigt)

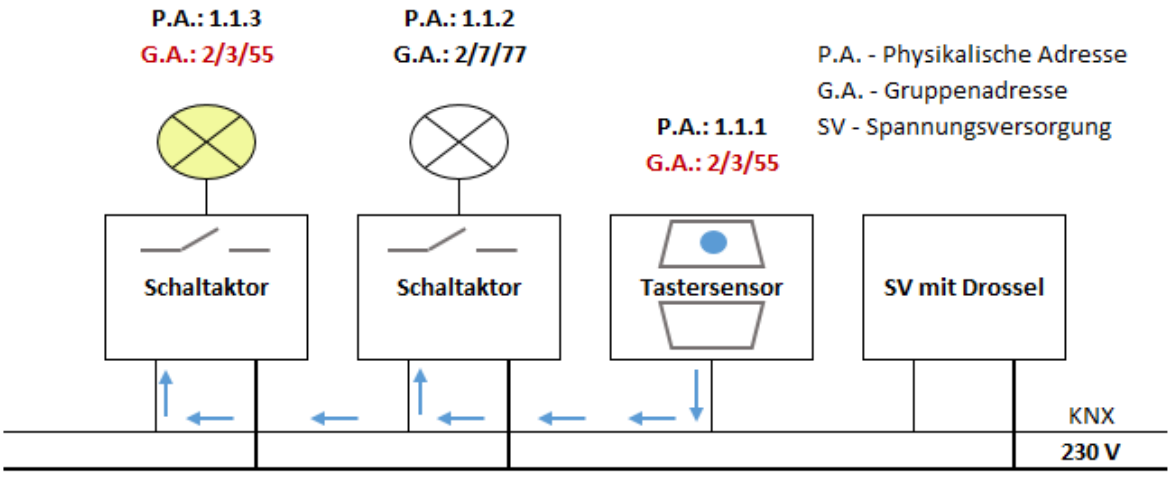

Abbildung 2.1 : Kommunikationablauf im KNX-System

Nach der Konfiguration funktioniert die Anlage wie folgt:

- Wird die obere Wippe des einfach Tastsensors (1.1.1) betätigt (siehe Abbildung 2.1), sendet dieser ein Telegramm aus, dass neben diversen Informationen die Gruppenadresse (2/3/55) und den Wert „1“ enthält.
- Dieses Telegramm wird von allen angeschlossenen Sensoren und Aktoren empfangen und ausgewertet
- Nur die Geräte, die die gleiche Gruppenadresse bei sich feststellen:
  - ❖ Senden ein Quittungstelegramm.
  - ❖ Lesen den Wert und verhalten sich dementsprechend.
    (In dem oben genannten Beispiel wird der Schaltaktor (1.1.3) sein Ausgangsrelais und damit die Lampe einschalten)

Derselbe Vorgang ereignet sich bei Betätigung der unteren Wippe, nur der Wert wird diesmal zu „0“ gesetzt und somit das Ausgangsrelais des Aktors ausgeschaltet.

2.2 Brandmeldeanlage der Firma Notifier

Eine Brandmeldeanlage ist eine Gefahrenmeldeanlage aus dem Bereich des vorbeugenden Brandschutzes, die Ereignisse von verschiedenen Brandmeldern empfängt, auswertet und dann reagiert. Sie können im Grundausbau keine Brandentstehung verhindern oder einen Brand löschen. Ihre Aufgaben bestehen darin, entstehende Brände möglichst früh zu erkennen und diese dann so schnell wie möglich an die zu Hilfe leistende Stelle (z.B. die Feuerwehr) zu informieren. Außerdem sollen Personen, die sich im Gebäude befinden, durch akustische Signale gewarnt werden, um so den Schutz der Betroffenen zu erhöhen.
Des Weiteren können Brandschutzeinrichtungen angesteuert werden, um zum Beispiel durch eine Raum- und Wärmeabzugsanlage die entstandenen giftigen Dämpfe beseitigen zu können oder eine Ausbreitung dieser mit Hilfe von Brandschutztüren zu verhindern. Meistens werden Brandmeldeanlagen in besonders gefährdeten Gebäuden, wie Flughäfen, Bahnhöfen, Schulen, Universitäten, Fabrikhallen, Altenwohnheimen und Krankenhäuser installiert. Die Pflicht zu einem Einbau einer auf die Feuerwehr aufgeschalteten Brandmeldeanlage ist im Bauordnungsrecht im Rahmen von

Sonderbauvorschriften geregelt. Mit einer Baugenehmigung kann die Bauaufsicht gegebenenfalls den Einbau einer Brandmeldeanlage fordern.

2.2.1 Bestandteile einer Brandmeldeanlage

Brandmeldeanlagen bestehen aus zentralen und dezentralen Komponenten. Zur Brandmeldezentrale (BMZ) gehören neben elektronischen Bauteilen zur Informationsverarbeitung die Energieversorgung und die Bedien- und Anzeigeeinheit. Die im Gebäude befindlichen automatischen Brandmelder (z.B. Rauch- und Wärmemelder) und Handfeuermelder dienen der eigentlichen Branderkennung.

Im Außenbereich eines Gebäudes werden folgende Komponenten installiert, um der Feuerwehr den Zugang zum Gebäude zu ermöglichen:

- Feuerwehr-Schlüsseldepot
- Freischaltelement
- Blitzleuchte

Damit der Brandort schnell gefunden werden kann und um die wichtigsten Schalthandlungen vornehmen zu können, werden an der Erstinformationsstelle der Feuerwehr folgende Komponenten angefordert:

- Feuerwehr-Bedienfeld
- Feuerwehr-Anzeigetableau
- Laufkartendepot und Laufkarten
- Sprechstelle für Durchsagen (optional)

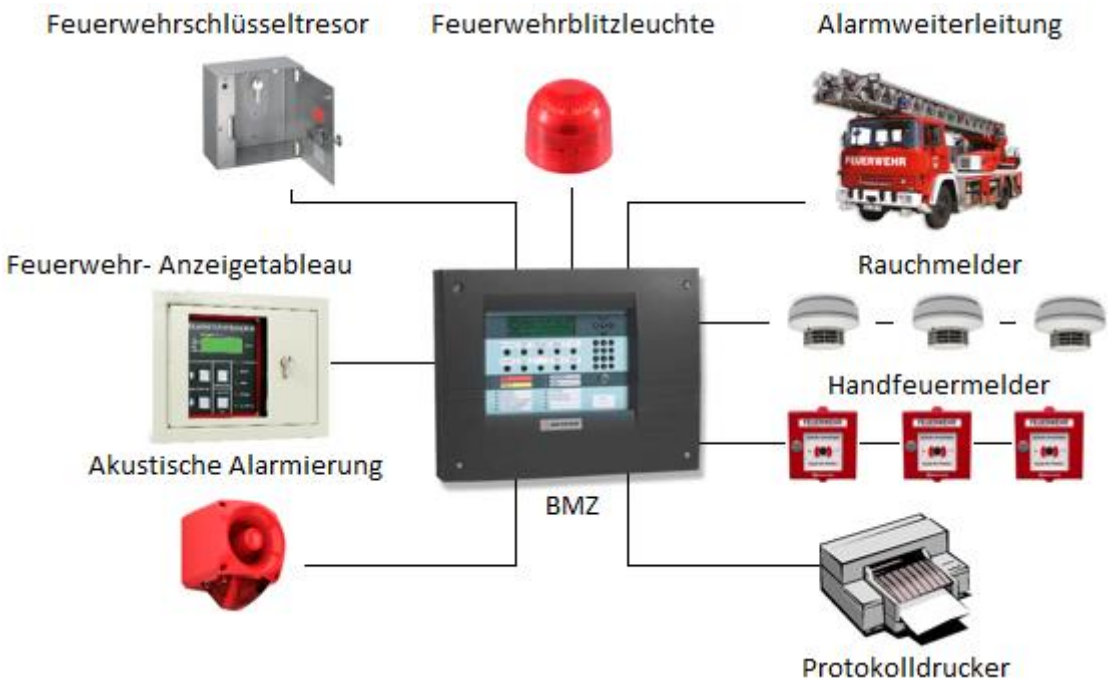

Abbildung 2.2 : Aufbau einer Brandmeldeanlage

Zusätzlich können an der Brandmeldeanlage akustische und optische Signalgeber, sowie Schalter für die Aktivierung von Brandschutzeinrichtungen angeschlossen werden.
All diese Komponenten zusammen mit dem Leitungsnetz bilden die Brandmeldeanlage.

Brandmeldezentrale (BMZ)

Die Brandmeldezentrale ist im Grunde genommen das Herzstück einer Brandmeldeanlage. Hier laufen alle Informationen zusammen, Meldungen der installierten Sensoren laufen dort ein und lösen die vorher einprogrammierte Aktion aus. Die Brandmeldezentrale hat die Aufgabe Meldungen der angeschlossenen Melder aufzunehmen, sie optisch und akustisch anzuzeigen, den Ort der Gefahr zu kennzeichnen, sowie die Meldung gegebenenfalls zu registrieren. Des Weiteren dient sie zur Überwachung möglicher Fehlsituationen, z.B. bei Kurzschluss, Drahtbruch oder sonstigen Störungen. Soweit erforderlich, sind Signale an die Feuerwehr oder über die Steuereinrichtung für automatische Brandschutzeinrichtungen weiterzuleiten, z.B. zu einer automatischen Löschanlage, Aufzüge, Rauch- und Wärmeabzugsanlagen.

2.2.2 Schnittstellen der BMZ

Um Lageplantableaus und Fremdgeräte an einer Brandmeldeanlage anschließen zu können, besitzen diese je nach Art der Brandmeldezentrale eine bestimmte Schnittstelle. Bei Brandmeldezentralen der Firma Notifier gibt es je nach Modell eine ISO RS232, ISO RS422 und/oder eine ISO RS485 Schnittstelle. Da die Verknüpfung der Brandmeldeanlage mit dem KNX allerdings über die RS232-Schnittstelle erfolgen soll, wird in diesem Abschnitt nur explizit auf die RS232-Schnittstelle eingegangen.

RS232-Schnittstelle

RS-232 ist der Name der am meisten verwendeten seriellen asynchronen Schnittstelle, um Daten zwischen zwei elektronischen Geräten aus zu tauschen (im Fachjargon: Datenkommunikation). Der offizielle Name ist EIA RS-232-C, genormt durch die amerikanische Electronic Industries Alliance (EIA). Ursprünglich diente die RS232-Schnittstelle lange Zeit dem Datenaustausch zwischen Computer („Data Terminal Equipment“, kurz DTE) und Modem (Data Communication Equipment, kurz DCE). Die „klassischen“ Einsatzgebiete von RS232 haben sich über die Jahre hinweg etwas geändert. Während ursprünglich die Fernschreiber/Terminal/Computer – Modem Kommunikation im Vordergrund stand, sind es heute vor allem Konfigurationsaufgaben von Peripheriehardware, welche die RS232-Schnittstelle übernimmt.

Die Übertragung von Informationen erfolgt dabei in Wörtern. Ein Wort entspricht dabei je nach Konfiguration fünf bis neun Bits, in dem dann ein einzelnes Zeichen kodiert ist. Meistens erfolgt die Kodierung gemäß ASCII („American Standard Code for Information Interchange“). Durch Verwendung eines Startbits und eines oder mehreren Stopbits signalisiert der Sender dementsprechend den Anfang und das Ende eines übertragenden Frames.

Eine RS232-Verbindung arbeitet (bit-)seriell mit je einer Datenleitung für beide Übertragungsrichtungen. Das heißt, die Bits werden, im Gegensatz zur parallelen Datenübertragung, nacheinander auf einer Leitung übertragen. Die dafür nötige Seriell-Parallel-Wandlung geschieht meistens in sogenannten UARTs („Universal Asynchronous Receiver Transmitter“, entweder als integriertes Modul in einem Mikrocontroller oder als Einzelbaustein).

Die binären Zustände werden durch verschiedene elektrische Spannungspegel realisiert. Für die Datenleitungen wird eine negative Logik verwendet, wobei eine Spannung zwischen −3 V und −15 V eine logische Eins und eine Spannung zwischen +3 V und +15 V eine logische Null darstellt. Signalpegel zwischen −3 V und +3 V gelten als undefiniert. Die oben angegebenen Spannungen beziehen sich auf die Empfänger (Eingänge). Bei den Sendern (Ausgänge) muss die Spannung mindestens +5 V bzw. −5 V an einer Last von 3…7 kΩ betragen, um genügend Störabstand zu gewährleisten. Üblich ist die Verwendung von +12 V und −12 V. Der Standard legt keine Bitraten fest, obwohl erwähnt wird, dass er für Übertragungsraten bis 20.000 bit/s gedacht ist. Übliche UARTs die in Verbindung mit der RS232-Schnittstelle verwendet werden, unterstützen Übertragungsraten von 115,2 kbit/s und mehr. Um ein definiertes Übertragungsverhalten zu erreichen, schreibt die Norm eine maximale Flankensteilheit am Sender und eine (von der Bitrate abhängige) minimale Flankensteilheit im Übergangsbereich −3 V … +3 V am Empfänger vor.

Tabelle 2.1 : Maximale Übertragungsrate und Leitungslänge

max. Baud	max. Länge
2.400	900 m
4.800	300 m
9.600	152 m
19.200	15 m
57.600	5 m
115.200	<2 m

Als Steckverbindung wurden nach der ursprünglichen Norm 25-polige D-Sub-Stecker („D-Subminiature“) für DTE und 25-polige D-Sub-Buchsen für DCE benutzt. Da viele der 25 Leitungen reine Drucker- bzw. Terminal-Steuerleitungen aus der elektromechanischen Ära sind, die für die meisten Verbindungen mit moderneren Peripheriegeräten nicht benötigt werden, haben sich heute 9-polige D-Sub-Stecker und Buchsen etabliert, welche häufig DB-9 genannt werden oder korrekter DE-9.

Abbildung 2.3 : DB-9 Stecker (männlich) und DB-9 Buchse (weiblich)

Tabelle 2.2 : Pinbelegungen der 9-poligen RS232-Schnittstelle

Pin	Kurz	Beschreibung	Richtung beim DTE (PC - DB9 Stecker)	Richtung beim DCE (Modem - DB9 Buchse)
1	DCD	Trägererkennung (Data Carrier Detect)	Eingang	Ausgang
2	RXD	Empfangen von Daten (Receive Data)	Eingang (RXD in)	Ausgang (RXD out)
3	TXD	Senden von Daten (Transmit Data)	Ausgang (TXD out)	Eingang (TXD in)
4	DTR	Computer bereit (Data terminal Ready)	Ausgang	Eingang
5	GND	Masse (Ground)	Ausgang	Ausgang
6	DSR	Betriebsbereit (Data Set Ready)	Eingang	Ausgang
7	RTS	Sendeanforderung (Request To Send)	Ausgang	Eingang
8	CTS	Sendebereit (Clear To Send)	Eingang	Ausgang
9	RI	Ruf ankommend (Ring Indiator)	Eingang	Ausgang

Großrechner und Text-Terminals wurden bis in die frühen 1990er-Jahre unter Zuhilfenahme von Modems durch Punkt-zu-Punkt-Verbindungen über die Telefonleitung zusammengeschlossen. Die Übertragung der Daten bei beiden Systemen erfolgte sequenziell. Durch den ursprünglichen Verwendungszweck bedingt, weist die Schnittstelle, im Vergleich zu heutigen Anwendungen, bei der Definition der Steuerleitungen einige Asymmetrien auf, die bei den später üblich gewordenen Anwendungen in völlig anderen Bereichen zu Verschaltungsproblemen führen können.
Heutzutage kommt es auch vor, dass die Pinbelegung etwas anders bezeichnet wird, da der logische Zusammenhang sich eher erschließt, als die des definierten Standards, siehe Abbildung 2.4.

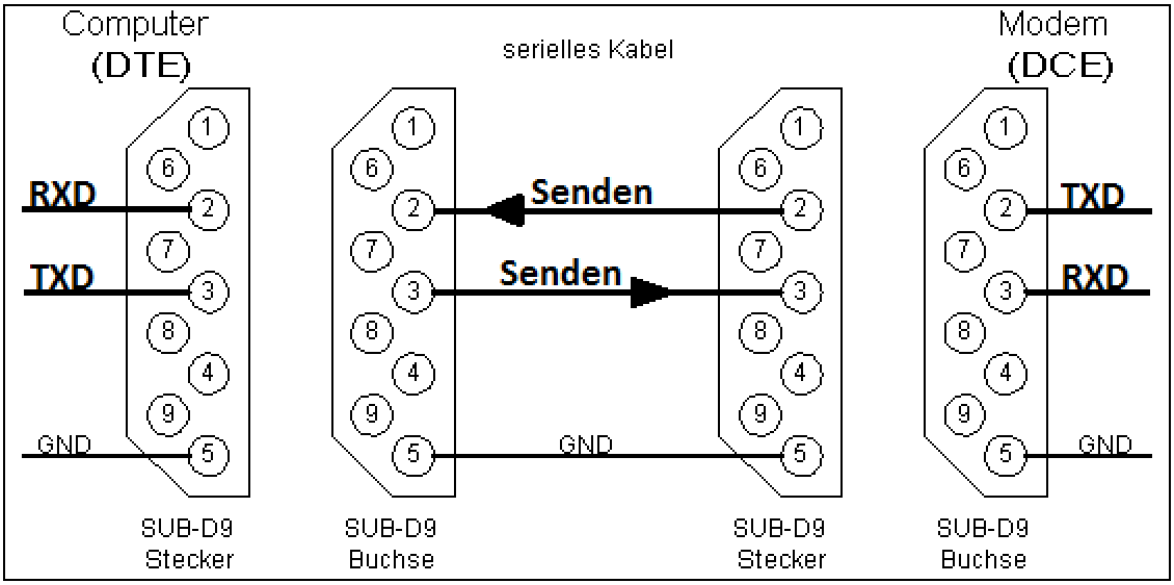

Abbildung 2.4 : Auftauchende Namensänderungen bei der RS232-Schnittstelle

Die Pinbelegung der DB9-Buchse wird dabei wie folgt verändert:

- „TXD in“ wird zu „RXD“
- „RXD out“ wird zu „TXD“

Da man nie genau weiß, welcher Entwickler bzw. Hersteller welche Bezeichnung verwendet, kann dies sehr irreführend sein, was sich im späteren Verlaufe noch bewahrheiten wird.

2.2.3 Informationsdaten

Wie bereits erwähnt, können die beiden Schnittstellentypen RS232 bzw. RS485 dafür genutzt werden, um Fremdgeräte anzuschließen. Beispielsweise ist es möglich einen Protokolldrucker anzuschließen, um einen zeitlichen Verlauf aller eingegangenen Informationen nachvollziehen zu können. Diese Ausgabeinformationen beinhalten neben der Uhrzeit und dem aktuellem Datum, zusätzlich zum Beispiel die Nummer eines Melders, die dazugehörig übertragende Informationsmeldung, die Bauart des Melders und einen Messwert in Prozent.

Beispiele:

```
65437: Fre 18-Jul-14 00:03:40:   * FEUERALARM *\r\n
R1 Mod.   4 - Gruppe 0004/01 - DKM \r\n
Grup.:Mitte\r\n
Ort: Handmelder Messw.:114%\r\n
\r\n
\r\n
65439: Fre 18-Jul-14 00:03:48:   Element wieder unter ALARMWERT\r\n
R1 Mod.   4 - Gruppe 0004/01 - DKM\r\n
Grup.:Mitte\r\n
Ort: Handmelder Messw.: 71%\r\n
\r\n
\r\n
00238: Fre 18-Jul-14 01:20:54:   VORALARM\r\n
R1 Meld.  1 - Gruppe 0001/02 - MULT\r\n
Grup.:Oben\r\n
Ort: Oben Links Messw.:108%\r\n
\r\n
\r\n
00443: Fre 18-Jul-14 02:50:48:   Wieder Vorhanden\r\n
R1 Meld.  3 - Gruppe 0001/03 - MULT\r\n
Grup.:Oben\r\n
Ort: Oben Rechts Messw.: 28%\r\n
```

Abbildung 2.5 : Beispiele ausgegebener Statusmeldungen

Die Übertragung der Informationen erfolgt standardmäßig mit 9600 Baud und ist dabei durch den ASCII-Zeichensatz definiert. Dieser ist eine 7-Bit-Zeichenkodierung und umfasst das lateinische Alphabet in Groß- und Kleinschreibung, die zehnarabischen Ziffern sowie einige Satzzeichen. Der Zeichenvorrat entspricht weitgehend dem einer Tastatur oder Schreibmaschine für die englische Sprache, siehe Tabelle 2.3.

Tabelle 2.3 : Ausschnitt der ASCII Zeichentabelle

Dez	Hex	ASCII	Dez	HEX	ASCII	Dez	Hex	ASCII	Dez	HEX	ASCII
…	…	…	46	2E	.	59	3B	;	72	48	H
…	…	…	47	2F	/	60	3C	<	73	49	I
35	23	#	48	30	0	61	3D	=	74	4A	J
36	24	$	49	31	1	62	3E	>	75	4B	K
37	25	%	50	32	2	63	3F	?	76	4C	L
38	26	&	51	33	3	64	40	@	77	4D	M
39	27	'	52	34	4	65	41	A	78	4E	N
40	28	(	53	35	5	66	42	B	79	4F	O
41	29	)	54	36	6	67	43	C	80	50	P
42	2A	*	55	37	7	68	44	D	81	51	Q
43	2B	+	56	38	8	69	45	E	82	52	R
44	2C	,	57	39	9	70	46	F	…	…	…
45	2D	-	58	3A	:	71	47	G	…	…	…

Bei der Brandmeldeanlage werden die einzelnen ausgegebenen Strings durch Zeilenumbrüche miteinander getrennt. Um den Zeilenumbruch in einer Textdatei explizit zu kodieren, existieren verschiedene Standards:

Tabelle 2.4 : Festgelegte Standards eines Zeilenumbruchs

Betriebssystem	Zeichensatz	Abkürzung	Code Hex	Code Dezimal	Escape-Sequenz
Unix, Linux, Android, Mac OS X, AmigaOS,BSD, weitere	ASCII	LF	0A	10	\n
Windows, DOS, OS/2, CP/M, TOS (Atari)		CR LF	0D 0A	13 10	\r\n
Mac OS bis Version 9, Apple II, C64		CR	0D	13	\r

Bei der Brandmeldeanlage wird der Zeilenumbruch durch ein *Carriage Return* (CR) und ein *Line Feed* (LF) gekennzeichnet, siehe Abbildung2.4. Diese sind neben der Signalisierung eines Zeilenumbruches zusätzlich notwendig um das Ende eines Strings feststellen zu können. Somit weiß das Empfangsgerät, wann ein String zu Ende ist.

2.3 Benötigte Systeme zur Realisierung der Aufgabe

Um die Verknüpfung des KNX-Systems und der Brandmeldeanlage zu realisieren. sind an dem gesuchten System gewissen Anforderungen gestellt. Es muss zu allererst die Textinformationen der Brandmeldeanlage aufnehmen und diese dann in irgendeiner Art und Weise mit Gruppenadressen verknüpfen.

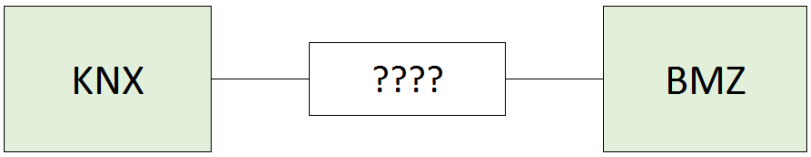

Abbildung 2.6 : Benötigtes System zur Verknüpfung des KNX-System mit der BMZ

Wie schon erwähnt gibt es im KNX-System eine Vielzahl von Bauteilen, die miteinander kommunizieren können. So gibt es auch Bauteile, die es ermöglichen eine Verbindung zwischen verschiedenste Arten von Schnittstellen und dem KNX herzustellen (z.B. KNX/Infarot-, KNX/Ethernet- oder KNX/DALI-Bus). Dadurch können Fremdgeräte mit dem KNX verbunden werden und somit kommunizieren. In diesem Fall wird ein Bauteil benötigt, das eine Verbindung zwischen einer RS232 Schnittstelle und dem KNX herstellt. Solche Geräte können beispielsweise festdefinierte Textzeilen, die von einem Fremdgerät gesendet werden, mit Gruppenadressen verbinden. Im nachfolgenden Beispiel soll dies genauer erläutert werden.

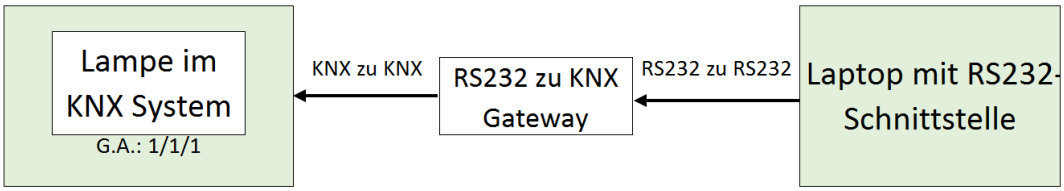

Abbildung 2.7 : Verwendung eines RS232 zu KNX Gateways

Das Ziel in diesem Beispiel (Abbildung 2.7) ist es, über bestimmte Textnachrichten, die vom Laptop aus gesendet werden eine Lampe im KNX-System an- bzw. ausstellen zu können. Man nehme an, die Lampe soll durch den Textstring „Lampe-AN“ angeschaltet und durch den Textstring „Lampe-AUS“ ausgeschaltet werden. Um die Kommunikation des PCs und dem KNX zu ermöglichen, können im Gateway Textstrings softwaretechnisch definiert und zusätzlich mit bestimmten Gruppenadressen und Werten verbunden werden.

Tabelle 2.5 : Kommunikationsbeispiel

Gruppenadresse	Wert	Textstring
1/1/1	1	Lampe-AN
1/1/1	0	Lampe-AUS

Mit Hilfe der Zuordnung in der Tabelle 2.5 ist es nun möglich, die Lampe an- und auszuschalten. Je nach Art des Bauteils wird diese Programmierung unterschiedlich gelöst.
Abfolge:

- Textstring „Lampe-AN“ wird vom PC gesendet
- Gateway empfängt diese Textnachricht und vergleicht diese mit der vordefinierten Filtertabelle
- Textstring „Lampe-AN“ ist enthalten
- Gruppenadresse 1/1/1 wird mit dem Wert „1“ auf den KNX-Bus gesendet
- Aktor der Lampe besitzt die gleiche Gruppenadresse, wird somit reagieren und sich anschalten, da zusätzlich der Wert „1“ versendet wurde.

Wird in diesem Fall der String „Lampe-Aus“ vom PC gesendet, verhält sich das Gateway identisch zum ersten Fall, außer das ein Wert „0“ auf den KNX-Bus geschickt wird und die Lampe somit ausgeht. Werden außer den vordefinierten Textstrings andere Strings vom PC aus gesendet, hat das keinen Einfluss, da das Gateway somit keine Gruppenadressen zum KNX-Bus verschickt.

Diese Art der Verknüpfung von Textstrings mit Gruppenadressen soll bei der Verbindung der BMZ und dem KNX ebenfalls verwendet werden. Beispielsweise soll bei Eintreten eines Feueralarms die Meldung „FEUERALARM“ und die dazugehörige Meldernummer genutzt werden, um diese dann mit bestimmten Gruppenadressen verknüpfen zu können, damit z.B. in einer visuellen Darstellung angezeigt werden kann, dass ein Melder ausgelöst wurde. Dabei tritt aber das Problem auf, dass die BMZ keine konstanten Strings versendet, da diese ein Datum und die Uhrzeit enthalten. Zusätzlich besteht das Problem, dass die benötigten Informationen („FEUERALARM“ und die dazu gehörige Meldernummer) nicht in einem String gesendet werden, sondern nacheinander.
Da ein Gateway nur konstante Textstrings aufnehmen kann, ist es nicht ohne weiteres möglich, die Textinformationen der BMZ im Gateway zu definieren und diese mit Gruppenadressen zu verbinden. Man muss also eine Lösung finden, um die wichtigsten Textinformationen der BMZ entsprechend zu filtern und erst dann zum Gateway zu schicken.

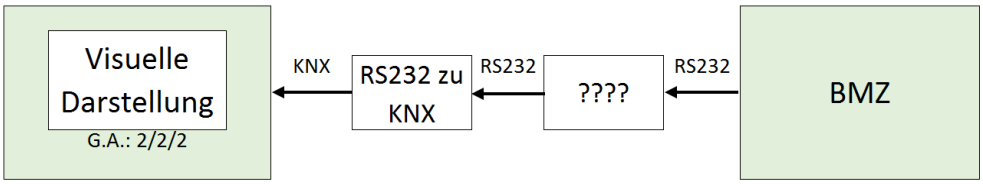

Abbildung 2.8 : System zur Filterung von Textnachrichten

Die wichtigsten Informationen der BMZ sind die Statusmeldungen an sich (z.B. „FEUERALARM“, „VORALARM“ etc.) und die dazugehörigen Meldernummern (z.B. „0001/02“, „0002/14“ etc.).

Im Beispiel eines Feueralarms soll es sich wie folgt verhalten:

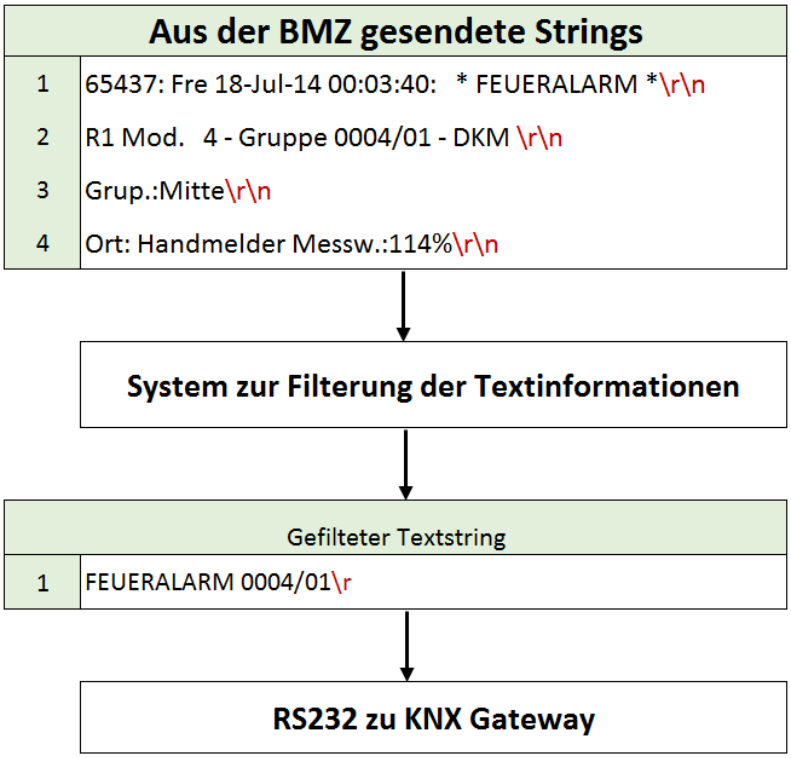

Abbildung 2.9 : Zielsetzung der gewünschten Filterung

- Die BMZ sendet Textinformationen über eine RS232-Schnittstelle aus
- Mit Hilfe eines weiteren Systems sollen diese dann gefiltert werden
- Der gefilterte Textstring enthält dann alle wichtigen Informationen und ist konstant
- Damit kann das RS232/KNX Gateway diesen String ohne weiteres aufnehmen und mit Gruppenadressen verknüpfen

Dabei ist allerdings auch zu achten, dass der gefilterte Text auch mit einem Carriage return („\r“ oder „CR“)beendet wird, damit das RS232 zu KNX Gateway das Ende eines Strings erkennen kann. Nachfolgend wird in der Tabelle 2.6 beschrieben, welche Statusmeldungen aus der BMZ gefiltert werden sollen.

Tabelle 2.6 : Übersicht aller gesuchten Textinformationen

	Statusmeldung	Besitzt zugehörige Meldernummer	Bemerkung
1	FEUERALARM	JA	Signalisiert Feuer
2	VORALARM	JA	Signalisiert kritischen Temperaturwert
3	ELEMENT wieder unter ALARMWERT	JA	Kritischer Temperaturwert wurde wieder unterschritten
4	ELEMENT FEHLT	JA	Störung eines Elementes
5	Wieder Vorhanden	JA	keine Störung mehr vorhanden
6	ZENTRALE ZURÜCKSETZEN	NEIN	setzt die BMZ auf Anfangszustand

Der String 6 (ZENTRALE ZURÜCKSETZEN) ist in diesem Fall dadurch wichtig, da die Brandmeldeanlage auf ihren Anfangszustand gesetzt wird und somit alle Melder ebenfalls zurückgesetzt werden. Dies muss bei einer späteren Visualisierung im KNX Netz berücksichtigt werden, damit auch dort alle enthaltenen Meldungen zurückgesetzt werden können.

Filterung

Eine Filterung lässt sich mit Hilfe eines PCs ohne weiteres realisieren. Durch die Anwendung einer Programmiersprache wie C, C++ etc. können die Strings aufgenommen und gefiltert werden. Da ein PC sich allerdings in der Praxis, durch zu große Abmessungen, als sehr unkomfortabel verhält, ist die Verwendung von Mikrocontrollern eher sinnvoll. Diese ermöglichen ebenfalls eine Anbindung an einer seriellen Schnittstelle wie RS232 (UART-Schnittstelle des Mikrocontrollers) und die Programmierung kann neben Assembler ebenfalls in C realisiert werden, um die eintreffenden Textinformationen erfolgreich zu filtern. Bei der Verwendung der UART-Schnittstelle eines Mikrocontrollers muss beachtet werden, dass der Controller mit einer TTL Spannung („Transistor-Transistor-Logik“, logisch 0 entspricht 0 V und logisch 1 entspricht 3 V) arbeitet. RS232-Schnittstellen haben dagegen einen Spannungsbereich von -15 V bis +15 V. Es muss also realisiert werden, dass die Spannungspegel konvertiert werden, damit die beiden Systeme auch wirklich miteinander kommunizieren können. Dazu können RS232 zu TTL Wandler verwendet werden, die es als fertiges Produkt zu kaufen gibt.

Abbildung 2.10 : Benötigte RS232 zu TTL Schnittstelle

2.4 Seriell zu KNX Gateway

Im diesen Kapitel sollen kurz einige RS232 zu KNX Gateways vorgestellt werden. Alle besitzen die Möglichkeit Textinformationen aufzunehmen und diese mit sogenannten Gruppenadressen zu verknüpfen.

2.4.1 Zennio SKX OPEN

SKX OPEN ist eine bidirektionale Kommunikationsschnittstelle für KNX-RS232. SKX OPEN ermöglicht die KNX-Integration externer Geräte über deren seriellen RS232-Port.

Abbildung 2.11 : Zennio SKX Open

Das Hauptmerkmal dieses Gerätes ist die Fähigkeit herstellerunabhängig mit einer weiten Protokollpalette interagieren zu können und so die ideale Schnittstelle zur Integration verschiedener Gerätemarken in ein KNX-System darzustellen. Dank eines Applikationsprogramms ist es möglich, jedes beliebige Gerät mit einer RS232-Schnittstelle zu integrieren, unter der Voraussetzung, dass die mit jeder Aktion verknüpften Hexadezimalcodes bekannt sind. Der SKX Open verfügt über 48 1-bit Kommunikationsobjekte über die eine Kommunikation zwischen KNX- und RS232-Protokoll durchgeführt werden kann. Außerdem stehen weitere 1-bit Objekte zur Fehlererkennung zur Verfügung. Die Übertragungsgeschwindigkeit ist frei wählbar (1200, 2400, 4800, 9600, 19200 Baud) und zusätzlich besteht die Möglichkeit das Übertragungsende der Frames zu erkennen (Timeout, End-Byte). Die maximale Länge des Protokolls beträgt 10 Byte bzw. 20 HEX-Zeichen (ohne End-Byte Datenrahmen, falls vorhanden).

2.4.2 KNX Gateway RS232/485-IP von „Elka Elektronik“

Das KNX Gateway RS232/485-IP ist ein intelligentes Systemgerät zur Ankopplung von unterschiedlichen Fremdsystemen mit RS232/RS485- oder Ethernet-Schnittstelle an den KNX. Das Gateway arbeitet mit ELKA Standard ASCII-Protokoll oder mit bis zu 500 frei definierbaren Strings und erlaubt so eine variable Protollanbindung an Endgeräte mit serieller Schnittstelle wie z.B. Beamer oder TV-Geräte. Unabhängig von der konfigurierten Kommunikationsstrecke ist der E-Mailversand bei Störungen möglich. Mit Hilfe der Toolsoftware KNX-Gate2 wird das Gateway wahlweise über die Ethernet- oder die RS232-Schnittstelle programmiert. Zu Diagnosezwecken kann über RS232 bzw. Ethernet ein Trace-Modus in der KNX-Gate2 aktiviert werden, über den im Servicefall die übermittelten Daten zwischen KNX und dem Subsystem (z.B. Beamer oder TV-Gerät) aufgelistet werden. Des Weiteren unterstützt die KNX-Gate2 das Auslesen bereits projektierter Gateways (Rekonstruktion) und das Programmieren der physikalischen Adresse des KNX-Projekts. Es werden 2- und 3-stellige Gruppenadressen sowie die freie Gruppenadressstruktur unterstützt. Gruppenadressen, die bereits in der ETS-Software der KNX Association programmiert wurden, können per xml- oder csv-File importiert oder im Konfigurationstool projektiert werden.

Die Baudrate ist frei wählbar und zusätzlich ist es möglich, das Sendeverhalten (z.B. 8-bit mit 1-Stoppbit oder 8-bit mit 2-Stoppbits) zum Subsystem einzustellen. Abhängig der Ressourcen können bis zu 2000 Gruppenadressen zur Konfigurierung verwendet werden.

Abbildung 2.12 : KNX Gateway RS232/485-IP von „Elka Elektronik“

2.4.3 KNX-GW-RS232-RS485 von „arcus-eds“

Das KNX-zu-Seriell Gateway ist ein bidirektionales Interface zwischen dem KNX-Bus und den seriellen Schnittstellen RS232 und RS485. Es empfängt Daten-Telegramme auf dem KNX-Bus, stellt sie dem internen Interface-Programm zur Verfügung, welches daraus die seriellen Telegramme generiert. Ebenso kann bei Eintreffen eines seriellen Telegramms ein KNX-Telegramm erzeugt werden. Die Programmierung erfolgt in Abhängigkeit von der Anwendung. Applikationen können über die eingebaute USB-Schnittstelle programmiert, aktualisiert oder ausgetauscht werden. Das Gateway verfügt über einen Programmspeicher von 8KByte. Die verwendeten Gruppenadressen werden zusammen

mit den zu sendenden Zeichenketten in einer Projektdatei definiert und mittels USB-Kabels übertragen. Eine eindeutige physikalische Adresse kann über eine Dummyapplikation in der ETS vergeben werden. Des Weiteren ist das Gateway sofort betriebsbereit, es muss nicht über die ETS Software projektiert werden. Für das Gateway existieren Programmbeispiele und API´s für verschiedene Anwendungen. Aktuelle Versionen sind auf der Internetseite „http://www.arcus-eds.de“ zu finden.

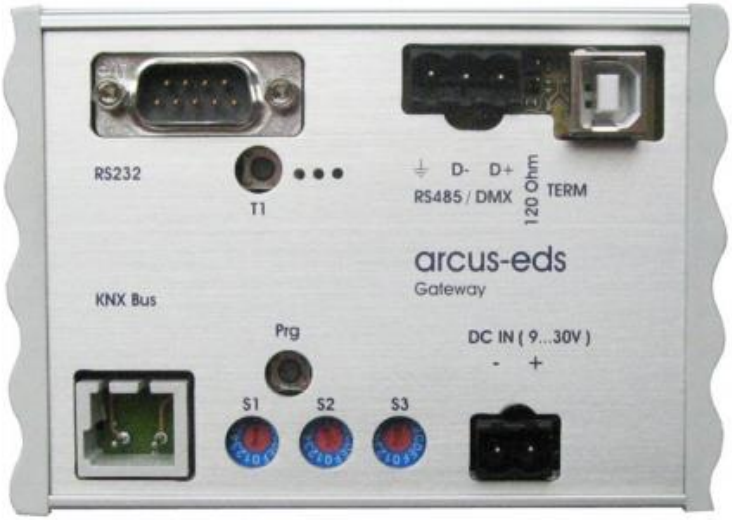

Abbildung 2.13 : KNX-GW-RS232-RS485 von „arcus-eds“

Programmbeispiel Beamersteuerung NEC

```
#define baudrate      baud_38400
#define serialmode    mode_8N1

#define addrtablelen 16
#include __initserialgatewayIII.code

## No Input from RS232

## No Standard-strings to RS232

## Hexstrings to RS232
## ON
1/2/0 2 0 0 0 0 2 6 hexstring h1
## OFF
1/2/1 2 1 0 0 0 3 6 hexstring h2
## Input1
1/2/2 2 3 0 0 2 1 0x01 0x09 8 hexstring h3
## Input2
1/2/3 2 3 0 0 2 1 0x02 0x0A 8 hexstring h4

fsave
```

Die Beispiele stellen nur beispielhaft die Programmierung dar, je nach Beamermodell steht eine mehr oder weniger große Anzahl von RS232-Befehlen zur Verfügung. Im Gateway können bis zu 200 Strings problemlos verarbeitet werden. Mit der vordefinierten Datei __initserialgatewayIII.code werden Zeichenketten beim Empfangen einer "1" gesendet. Mit der Datei __initserialgatewayIV.code können 1'en und 0'en verschiedene Zeichenketten zugeordnet werden.

2.5 Mikrocontroller

Grundsätzlich stellt ein Mikrocontroller nix anderes dar, als einen sehr kleinen Computer auf einem einzelnen Halbleiter-Chip. Ein Mikrocontroller arbeitet ebenfalls nach dem Funktionsprinzip Eingabe-Verarbeitung-Ausgabe. Er kann also Eingaben entgegennehmen, diese Daten dann verarbeiten und anschließend in irgendeiner Art und Weise reagieren und etwas ausgeben. Es lassen sich mit ihnen komplexe Aufgaben lösen, für die sonst ein aufwändiger Schaltungsaufwand notwendig wäre. Eingabemöglichkeiten können sich beispielsweise als Tastendrücke, Messwerte oder (Funk-) Signale wiederfinden. Je nachdem, welche Aufgaben das Programm vorsieht, können dann nach der Verarbeitung Reaktionen generiert werden, wie zum Beispiel: Leuchtanzeigen (LEDs, Textdisplays etc.), Kontrolle von Stellgliedern oder gesendete Signalpegel. In immer mehr Geräten des Alltags werden die Aufgaben von analogen Schaltungen durch Mikrocontroller realisiert. Damit lassen sich vor allem die Produktionskosten der Hardware drastisch senken. Mikrocontroller findet man häufig in eingebetteten Systemen (Embedded Systems). Unter Embedded Systems versteht man Datenverarbeitungssysteme, die in ein technisches Umfeld eingebettet sind und spezielle Anwendungen abarbeiten. Ein Beispiel für ein eingebettetes System wäre etwa die mit einem Mikrocontroller realisierte Steuerung einer Kaffeemaschine. Hier dient die Datenverarbeitungsleistung dazu, die umgebenen Komponenten wie Wasserbehälter, Heizelemente und Ventile zu koordinieren, um einen guten Kaffee zu bereiten. Die Anwendungsbereiche von Mikrocontrollern sind schier unendlich. In allen Bereichen unseres Lebens lassen sich heute „versteckte“ Mikrocontroller finden. Nicht nur die geringen Kosten und der geringe Leistungsverbrauch machen einen Mikrocontroller so wertvoll. Durch die einfache Programmierbarkeit lässt sich ein System im Nachhinein weitaus leichter warten und optimieren, ohne dass man die ganze Schaltung erneuern muss.

2.5.1 Auswahl des Mikrocontrollers/Atmega324A

Es gibt Dutzende Hersteller, und jeder davon hat unzählige Varianten im Angebot. Um sich bei der Auswahl eines Mikrocontrollers richtig zu entscheiden muss man bei der Entscheidung u.a. bestimmte Kriterien beachten. Je nach Aufgabe eines Mikrocontrollers muss die Taktgeschwindigkeit, die Bitbreite und zum Beispiel die Bauform betrachtet werden. Zusätzlich spielen natürlich auch der Preis, der Stromverbrauch, der Speicher und die Onboard-Peripherie eine wichtige Rolle.
Neben klassischen (bedrahteten) Bauformen setzt sich heutzutage SMD immer mehr durch. Surface Mountable Device (Oberflächenmontiertes Bauteil), kurz SMD, steht für ein elektronisches Bauteil, welches ohne Bohrungen direkt auf die Oberfläche einer Platine gelötet wird. Für SMD benutzt man üblicherweise geätzte Platinen oder Adapter (die aber wieder extra kosten). Will man mit Lochrasterplatinen

oder Breadboards arbeiten, dann braucht man die klassischen Bauformen, z. B. PDIP (Plastic Dual In-line Package). Zu beachten ist dabei, dass es PDIP oft nur bis DIP40 (also mit 40 Pins) gibt, d.h. einen Mikrocontroller mit 50 I/O-Pins kann es dann nur als SMD geben. Viele Mikrocontroller sind in verschiedenen Bauformen verfügbar.

Betrachtet man die Onboard-Peripherie haben Mikrocontroller meist eine ganze Menge Funktionen integriert, z. B. Analog-Digital-Wandler, I²C-Bus (Inter-Integrated Circuit), SPI („Serial Peripheral Interface“), PWM („Pulse-width modulation“), RS-232 usw. Der Vorteil liegt darin, dass der Controller damit mehrere Dinge gleichzeitig machen kann. Wenn man zehn zeitkritische Dinge gleichzeitig erledigen muss, ist die Programmierung dadurch deutlich einfacher. Je nach Anforderung eines Systems, sollten vor allem die Peripherieeigenschaften eines Mikrocontrollers betrachtet werden.

Wie gerade angesprochen ist auch die Bitbreite (8, 16 oder 32 Bit) mitentscheidend für die Auswahl, da diese einen großen Einfluss auf die Verarbeitungsgeschwindigkeit hat. Da der Preis in etwa linear mit der Bitbreite steigt, sind 8-Bit-Controller nach wie vor die am verbreitetsten. Für Anwendungen, die eine höhere Verarbeitungsleistung benötigen, setzen die Entwickler mittlerweile meist direkt auf 32-Bit-Mikrocontroller. Rechenintensive Programme, insbesondere mit 32-Bit-Zahlen oder -Zwischenergebnissen oder gar Gleitkommazahlen, gehören in einen 32-Bit-Controller. Das betrifft insbesondere digitale Filteralgorithmen, die zumeist schnell ablaufen müssen. In der Regel haben 32-Bit-Controller auch höhere Taktraten, etwa 60 MHz (ARM7-Mikrocontroller) statt 16 MHz (AVR-Mikrocontroller). Operiert man mit vielen 16-Bit-Zahlen, typischerweise bei der Messwerterfassung von einer Datenquelle mit einer Abtastrate im kHz-Bereich oder mit 16-Bit-Timern, erspart man sich eine Reihe Probleme durch die Auswahl eines 16-Bit-Controllers. Auch 32-Bit-Adressen lassen sich noch halbwegs vernünftig berechnen. Alles andere passt in einen 8-Bit-Controller.
Wenn man allein die Taktgeschwindigkeit betrachtet wünscht man sich oft einen möglichst schnellen Controller, insbesondere als Anfänger, wenn man effiziente Lösungen noch nicht so kennt, andererseits schlägt sich ein hoher Takt auch im Stromverbrauch und im Preis nieder. Man sollte sich dabei nicht von den hohen Taktraten der PC-Prozessoren irritieren lassen. Für viele Anwendungen reicht ein 1-MHz-Takt völlig aus.
Im Vergleich zu PC-Prozessoren (Pentium, Athlon usw.) brauchen Mikrocontroller relativ wenig Strom. Will man sie allerdings mit Batterien betreiben, dann wird der Stromverbrauch plötzlich doch wichtig. Die meisten Mikrocontroller besitzen hierfür Stromsparmodi, mit denen man den Controller teilweise abschalten kann. Für einen extrem geringen Stromverbrauch sind z. B. die MSP430-Controller oder EFM32-Controller optimiert. Der Stromverbrauch hängt auch stark vom Takt und der Versorgungsspannung ab.

Atmega324A

Wie bereits erwähnt, gibt es verschiedene Kriterien die man bei der Auswahl eines Mikrocontrollers beachten sollte. Um die gestellten Anforderungen der Filterung von Textnachrichten realisieren zu können, sind abgesehen von der Peripherie keine großen Anforderungen an den Mikrocontroller gestellt. Es ist dabei völlig ausreichend einen 8-Bit-Mikrocontroller zu verwenden.

Das Vorhandensein einer UART-Schnittstelle spielt in diesem Fall die größte Rolle. Da man dadurch nicht auf allzu viele Kriterien achten muss, bieten sich eine Vielzahl von Mikrocontrollern an (z.B. ATmega168, ATmega324A, ATmega48, AT90PWM216 etc.).
Die Entscheidung traf letztendlich auf den ATmega324A, weil dieser sogar zwei UART Schnittstellen besitzt und sich dadurch noch weitere Möglichkeiten ergeben könnten. Des Weiteren wird dieser auch von den entsprechenden Programmiergeräten unterstützt, die im nachfolgenden Kapitel noch näher erläutert werden.

Der ATmega324A von Atmel ist wie bereits erwähnt, ein 8-Bit AVR RISC-basierter Mikrocontroller. Dieser kombiniert z.B. einen 32KB ISP Flash-Speicher, 1KB EEPROM, 2KB SRAM, 32 universelle I/O Ports, 32 mehrzweck Register, einen Echtzeitzähler, drei flexible Timer/Counter und zwei UARTs. Um einen gesamten Überblick der Peripherie zu erhalten, sollte ein Blick in das Datenblatt bzw. in das Blockschaltbild geworfen werden.

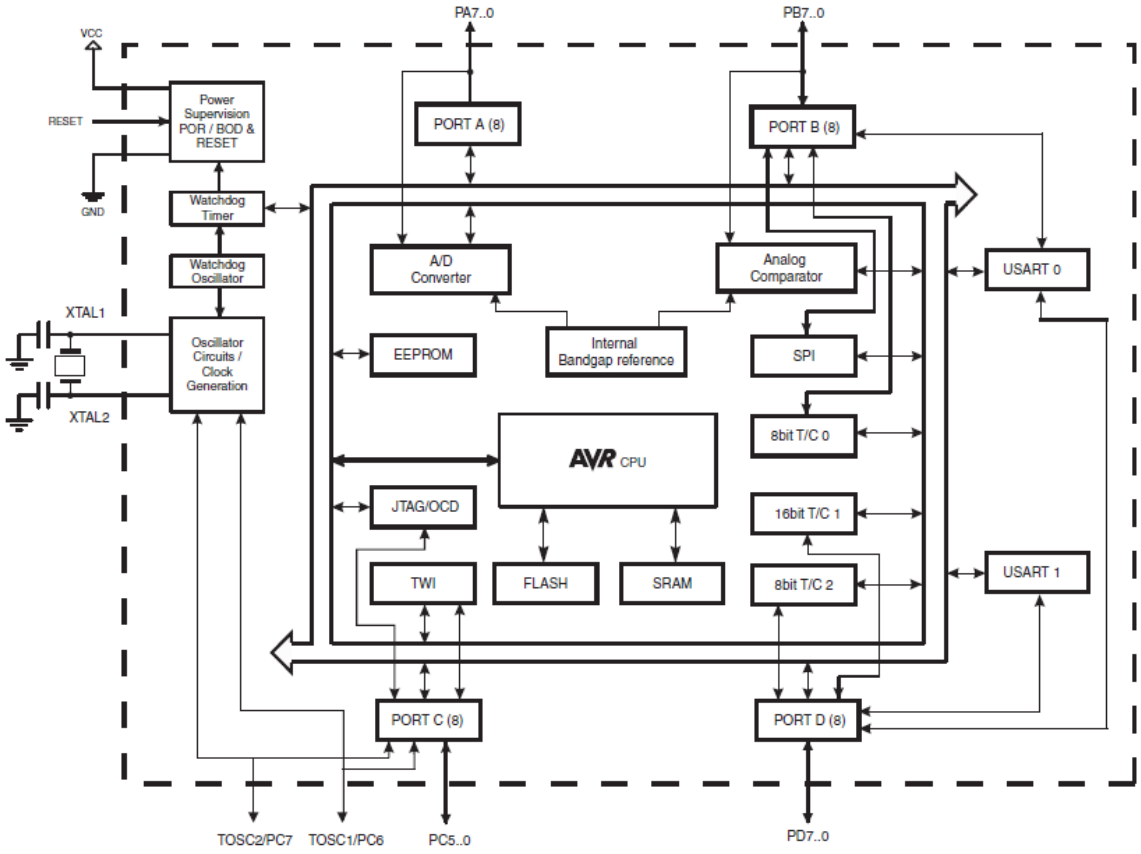

Abbildung 2.14 : ATmega324A Blockschaltbild

2.5.2 Analyse des zu benutzenden Oszillators

Die standardmäßige Datenübertragung der Brandmeldeanlage erfolgt über 9600 Baud. Daher ist zu beachten, ob der Mikrocontroller mit dieser Baudrate so umgehen kann, dass in der Übertragung keine Fehler entstehen. Im Datenblatt des Mikrocontrollers ATmega324A kann nachvollzogen werden, welchen internen Oszillator dieser besitzt, also mit welcher Geschwindigkeit dieser Arbeit, und wie groß die Toleranzen der angewandten Baudraten in Abhängigkeit des internen Oszillators sind (siehe Tabelle 2.7).

Tabelle 2.7 : Fehlertoleranzen in Abhängigkeit der Taktfrequenz

Baud rate [bps]	f = 4.0000MHz				f = 7.3728MHz				f = 8.0000MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBBR	Error	UBRR	Error	UBBR	Error	UBRR	Error	UBBR	Error
2400	103	0,2%	207	0,20%	191	0,0%	383	0,0%	207	0,2%	416	-0,1%
4800	51	0,2%	103	0,20%	95	0,0%	191	0,0%	103	0,2%	207	0,2%
9600	**25**	**0,2%**	**51**	**0,20%**	**47**	**0,0%**	**95**	**0,0%**	**51**	**0,2%**	**103**	**0,2%**
14.4k	16	2,1%	34	-0,80%	31	0,0%	63	0,0%	34	-0,8%	68	0,6%
19.2k	12	0,2%	25	0,2%	23	0,0%	47	0,0%	25	0,2%	51	0,2%
28.8k	8	-3,5%	16	2,1%	15	0,0%	31	0,0%	16	2,1%	34	-0,8%
38.4k	6	-7,0%	12	0,2%	11	0,0%	23	0,0%	12	0,2%	25	0,2%
57.6k	3	8,5%	8	-3,50%	7	0,0%	15	0,0%	8	-3,5%	16	2,1%
76.8k	2	8,5%	6	-7,00%	5	0,0%	11	0,0%	6	-7,0%	12	0,2%
115.2k	1	8,5%	3	8,50%	3	0,0%	7	0,0%	3	8,5%	8	-3,5%
230.4k	0	8,5%	1	8,50%	1	0,0%	3	0,0%	1	8,5%	3	8,5%
250k	0	0,0%	1	0,00%	1	-7,8%	3	-7,8%	1	0,0%	3	0,0%
0.5M	-	-	0	0,00%	0	-7,8%	1	-7,8%	0	0,0%	1	0,0%
1M	-	-	-	-	-	-	0	-7,8%	-	-	0	0,0%

Wie klar zu erkennen ist, hat der interne Oszillator (8MHz) Toleranzen von 0,2% aufzuweisen. Dies scheint auf den ersten Blick nicht viel zu sein, dennoch soll erreicht werden, dass die Fehler möglichst minimal gehalten werden. Daher soll ein externer Oszillator (Schwingquarz) verwendet werden, der bei einer Übertragungsrate von 9600 Baud, eine 0,0% Toleranz mit sich bringt. Bei der Lösung der Aufgabe wird auf ein Oszillator mit einer Frequenz von 7,3728 MHz zurückgegriffen, wobei es auch andere Möglichkeiten zur Lösung des Zieles gibt (z.B. die Verwendung von 14,7456 MHz oder 11,0592 MHz).

Abbildung 2.15 : Schwingquarz

2.5.3 Programmierung

Mikrocontroller werden meistens in den Programmiersprachen Assembler oder C programmiert. Andere Sprachen wie BASIC, Pascal, Forth, Ada oder C++ sind allerdings ebenfalls möglich. Die eingesetzte Programmiersprache hängt mit von Architektur, Anwendungsziel und Anwendungsanforderungen ab. Für 4-Bit-Architekturen wird praktisch ausschließlich Assembler verwendet, da nur die wenigsten Compiler die knappen Ressourcen dieser kleinsten Mikrocontroller effektiv nutzen können. Assembler wird auch häufig bei 8-Bit-Architekturen eingesetzt, um möglichst hohe Effizienz und Code-Dichte zu erreichen. Es ist allerdings zunehmend üblich, bei Projekten, die vergleichsweise wenig der zur Verfügung stehenden Mikrocontroller-Ressourcen in Anspruch nehmen, Hochsprachen zu verwenden, um den resultierenden höheren Entwicklungsaufwand einzusparen.

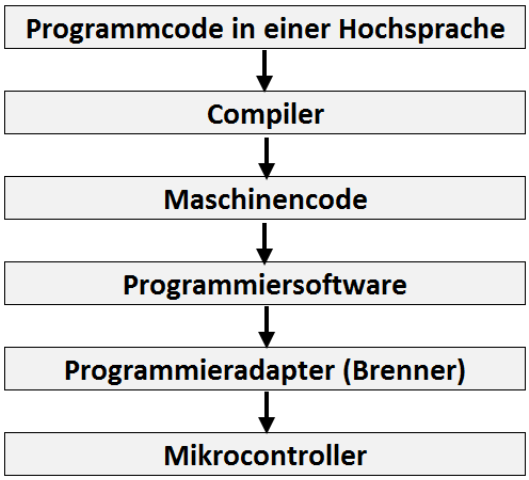

Abbildung 2.16 : Programmierungsabfolge

Der Programmcode wird am Computer durch den Entwickler erstellt. Bei Verwendung einer Hochsprache wie C werden die Anweisungen vom Compiler in für den Mikrocontroller verständlichen Maschinencode übersetzt. Dieser Bytecode wird dann mit Hilfe einer Programmiersoftware an den externen ISP-Programmieradapter (In System Programmer) übertragen, welcher die Daten im Flash-Speicher des Mikrocontrollers abspeichert.

Tabelle 2.8 : Programmiersoftare für Mikrocontroller

	Unterstützte Sprachen						
Hersteller	**Compiler**	**Ada**	**Assembler**	**Basic**	**C**	**C++**	**Pascal**
GNU	avr-as	Nein	Ja	Nein	Nein	Nein	Nein
GNU	avr-gcc	Nein	Ja	Nein	Ja	Ja	Nein
GNU	AVR-Ada	Ja	Nein	Nein	Nein	Nein	Nein
Atmel	Atmel Studio	Nein	Ja	Nein	Ja	Ja	Nein
cadManiac.org	KontrollerLab	Nein	Nein	Ja	Ja	Ja	Nein
Conrad Electronic	C-Control Pro	Nein	Ja	Ja	Ja	Nein	Nein
E-LAB Computers	AVRco	Nein	Nein	Nein	Nein	Nein	Ja
AdaCore	GNAT Pro	Ja	Nein	Nein	Ja	Ja	Nein
MCS Electronics	BASCOM	Nein	Nein	Ja(Dialekt)	Nein	Nein	Nein
nettypes.de	mBasic	Nein	Nein	Ja	Nein	Nein	Nein

Die AVR-Mikrocontroller von Atmel sind wegen ihrer übersichtlichen internen Struktur, der In-System-Programmierbarkeit, und der Vielzahl von kostenlosen Programmen zur Softwareentwicklung beliebt. In der Tabelle 2.8 sind einige Softwareprogramme zusammengefasst. Die sicherlich am meisten verbreitetste Software für Mikrocontroller ist Atmel Studio. Diese ist ebenfalls kostenlos und die hauseigene Software von Atmel. Diese Software enthält bereits einen Compiler, ermöglicht eine Programmierung in Assembler, C und C++, und zusätzlich kann diese auch Daten zu einem externen ISP-Programmieradapter übertragen.

2.5.4 Programmiergeräte und Evaluationsboards

Wie genauer erläutert wurde, benötigt man einen ISP-Programmieradapter um die Daten des Datencodes auf den Flashspeicher des Mikrocontrollers speichern zu können. Dabei gibt es ebenfalls eine große Auswahl. Je nach Wahl des Mikrocontrollers muss dementsprechend die Wahl des Programmiergerätes getroffen werden, denn nicht alle Programmiergeräte unterstützen auch alle Mikrocontroller.
In-System-Programming (ISP) bedeutet, einen Mikrocontroller oder anderen programmierbaren Bausteine im eingebauten Zustand zu programmieren. Dazu muss der Mikrocontroller entsprechend beschaltet sein. Das bedeutet, die benötigten Anschlüsse am Mikrocontroller müssen zugänglich und nicht ohne weitere Vorkehrungen anderweitig benutzt sein. Atmel verwendet für ihre 8-Bit RISC Mikrocontroller zum Teil unterschiedliche ISP-Protokolle. Das bekannteste davon wird einfach als ISP bezeichnet. Insgesamt findet man:

- ISP - Der Normalfall. Bei vielen, aber nicht allen AVRs teilen sich SPI- und ISP-Schnittstelle die Pins. Je nach AVR gibt es leichte Unterschiede im Protokoll. Das Protokoll für einen Typ ist im Datenblatt unter „Memory Programming“ -> „Serial Downloading“ beschrieben.
- TPI - Tiny Programming Interface. Einige AVRs der Tiny-Serie, besonders die 6-Pin Tinys benutzen dieses Protokoll.
- PDI - Programming and Debugging Interface. Dient zur Programmierung der XMEGAs und zum Debugging (dient zum Austesten der Hardware).
- JTAG – Joint Test Action Group. AVRs mit JTAG Debugging-Schnittstelle lassen sich auch über JTAG programmieren.
- Bootloader - Einige wenige AVRs kommen bereits mit einem einprogrammierten Bootloader. Bei diesen kann man ein zum Bootloader passendes Programm nutzen, um den AVR über eine im Bootloader definierte Schnittstelle programmieren.

Um einen kleinen Einblick zu bekommen, werden nachfolgend einige Programmiergeräte näher erläutert.

AVRISP mkII

Der AVRISP mkII ist ein In-System Programmer von Atmel und dient dem Upgrade von AVR Flash-Mikrocontrollern. Es ist möglich, durch die mitgelieferte Software Atmel Studio, AVR

Abbildung 2.17 : AVRISP mkII

Mikrocontroller über die ISP, PDI und TPI-Schnittstelle zu programmieren.
Die Spannungsversorgung erfolgt über den verfügbaren USB-Port, somit ist kein Netzteil nötig. Geeignet ist er zur Flash- und EEPROM-Programmierung. Er unterstützt eine Ziel-CPU-Spannung von 1,8…5,5 V und es ist möglich, die Programmiergeschwindigkeit (50 Hz … 8 MHz) einzustellen.

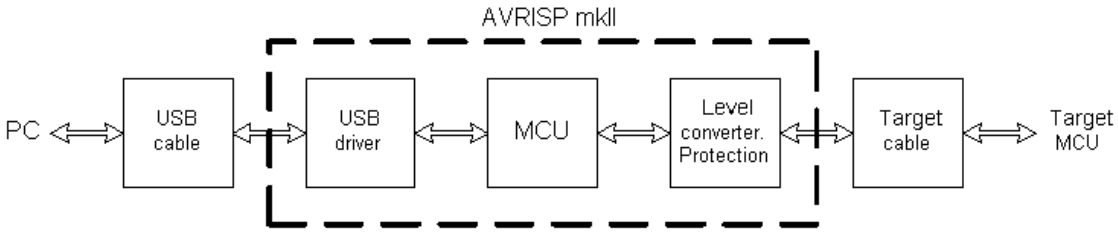

Abbildung 2.18 : Struktur des AVRISP mkII

DIAMEX PROG-S

Dieser ISP-Programmer (nur ISP; kein PDI und TPI unterstützt) ermöglich den Einsatz unterschiedliche Einsatzbereiche. Nicht nur die bekannten und beliebten AVR-Microcontroller können programmiert werden, sondern auch neue Cortex-M3 Controller. Mit dem eingebauten DIP-Schalter stellt man ein, ob ein AVR, STM32 oder NXP/LPC

Abbildung 2.19 : DIAMEX PROG-S

Controller programmiert werden soll. Des Weiteren ist es möglich, den DIAMEX PROG-S als USB-Seriell Wandler zu verwenden (bis zu 125.000 Baud). Zwei eingebaute Status-LEDs signalisieren den momentanen Zustand des Programmers und die Spannungsversorgung erfolgt wie bei dem AVRISP mkII direkt vom PC aus. Die Geschwindigkeit der Datenübertragung ist per Software einstellbar (bis maximal 12 Megabit). Die Verbindung zu einem Mikrocontroller erfolgt über ein 10poliges Programmer-Anschlusskabel. Neben der Software von Atmel (Atmel Studio) werden außerdem AVRDUDE und Basecom unterstützt. Neben herkömmlichen Programmiergeräten können auch sogenannte Evaluationsboards verwendet werden. Der Unterschied besteht darin, dass diese Boards neben einem funktionstüchtigen Programmer, zusätzlich mehrere Elemente besitzen, um die Funktion eines Mikrocontrollers zu testen. Dies ist besonders für Anfänger geeignet, da man zum Testen keine eigene Schaltung entwickeln muss.

ATMEL Evaluations-Board Ver. 2.0.1 von Pollin

Die Programmierung der Microcontroller kann direkt auf dem Atmel-Evaluations-Board erfolgen. Dazu bietet dieses Board drei verschiedene Möglichkeiten: Die Microcontroller ATmega8535, ATmega16, ATmega32, ATmega 644, Atmega8, Attiny 2313, Attiny12 und Attiny15 können über die ISP-Schnittstelle programmiert werden, indem sie über die serielle Schnittstelle mit dem PC verbunden werden, oder über einen ISP-Schnittstellenadapter, der über die 10-poligen Pfostenleisten mit dem Atmel-Evaluations-Board verbunden wird. Für die Programmierung der Microcontroller über die serielle ISP-Schnitstelle eignet sich das Programm PonyProg von Claudio Lanconelli, das kostenlos heruntergeladen werden kann. Der Mikrocontroller ATmega16, ATmega32 und ATmega 644 kann zusätzlich noch über eine JTAG-Schnittstelle, welche ein Debuggen der entwickelten Software ermöglicht, programmiert werden. Für den JTAG-Schnittstellenadapter ist der 10-polige Wannenstecker mit der Bezeichnung JTAG vorgesehen. Zusätzlich besteht es aus einer Vielzahl von Bauelementen wie Widerständen, Elkos, Kondensatoren, Gleichrichtern, RS232 zu TTL Wandler, LEDs, Dioden, ICs, Tastern, AC-Summer und Anschlussbuchsen (z.B. zur Anwendung von Quarzen), die eine einfache und zügige Anwendungsentwicklung ermöglichen. Durch einen 40-poligen Wannenstecker besteht die Möglichkeit, freien Zugriff auf jeden Pin eines Mikrocontrollers zu besitzen. Um das Evaluationsboard betreiben zu können, benötigt man eine Betriebsspannung von 9 V.

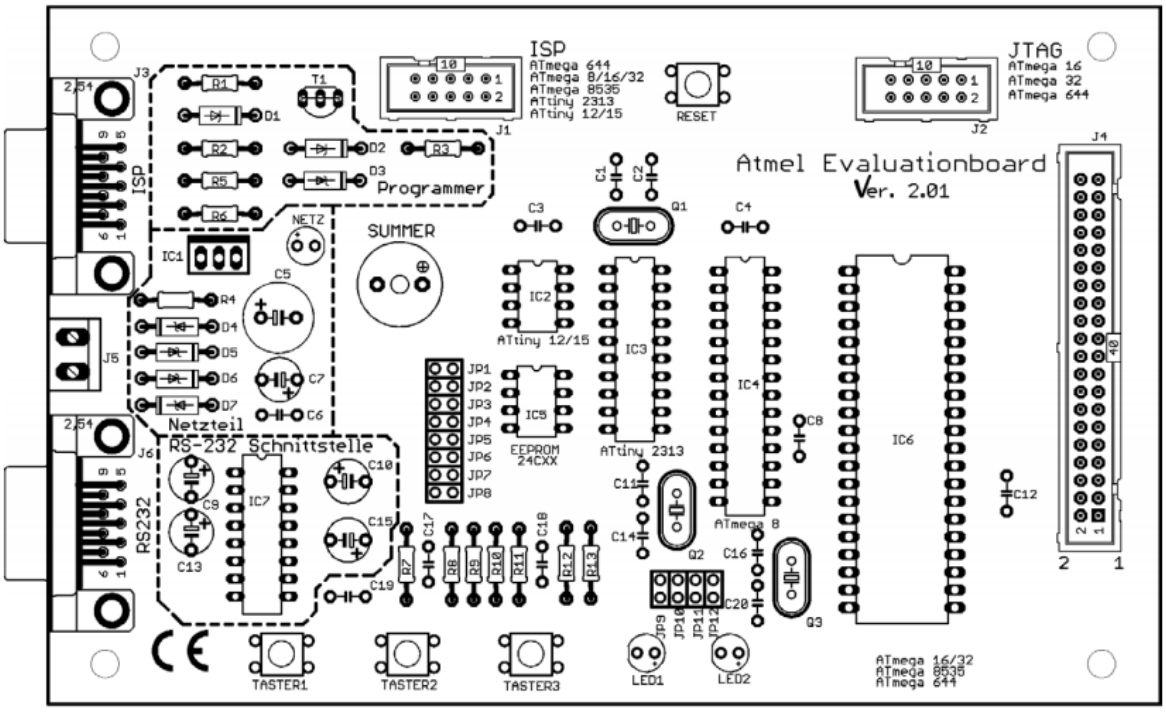

Abbildung 2.20 : ATMEL Evaluations-Board Ver. 2.0.1 von Pollin

aTeVaL (Atmel Evaluationsboard)

Das aTeVaL Evaluationsboard ist im Vergleich zum Evaluationsboard von Pollin ähnlich ausgestattet. Es verfügt zum Beispiel über zwei LEDs, drei Taster, zwei Potentiometer, einen Summer, ICs, einen Sockel für externe EEPROMs und drei Sockelpins für Schwingquarze. Zusätzlich verfügt es ebenfalls über einen 40-poligen Wannenstecker um auf alle Pins der Mikrocontroller zugreifen zu können. Des Weiteren werden drei Arten von Quarzen mitgeliefert (7,3728 MHz, 8 MHz, 16 MHz Quarze). Die Programmierung der Mikrocontroller kann direkt auf dem Board erfolgen. Über die ISP Schnittstelle können kompatible Controller programmiert werden, indem sie über eine vorhandene USB Schnittstelle mit dem PC verbunden werden. Zusätzlich kann die Programmierung auch über einen ISP-Schnittstellenadapter und einer JTAG-Schnittstelle erfolgen. Der Programmer an sich basiert auf den Atmel AVR-ISP MkII und ist damit mit allen Atmel- und AVR-Studios kompatibel. Den integrierten USB zu TTL Wandler kann durch Setzen eines Lötjumpers ebenfalls genutzt werden. Die komplette Versorgung des Evaluationsboards erfolgt über die USB Schnittstelle (5 V).

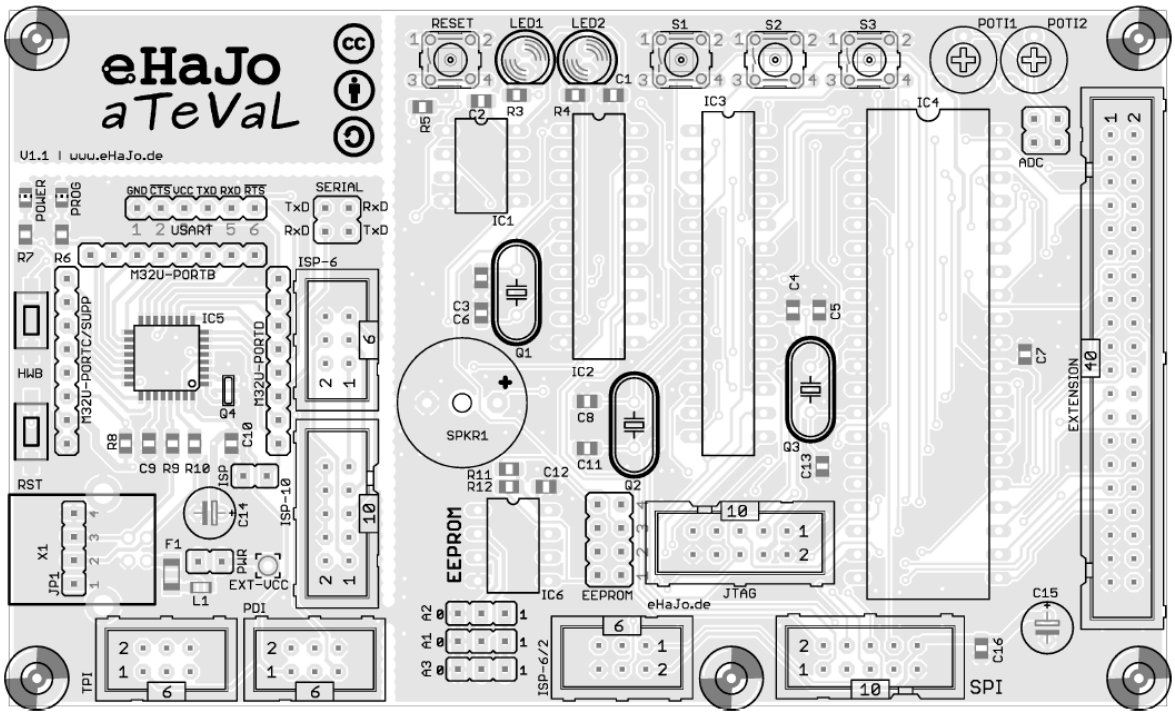

Abbildung 2.21 : aTeVaL (Atmel Evaluationsboard)

2.6 RS232 zu TTL Wandler

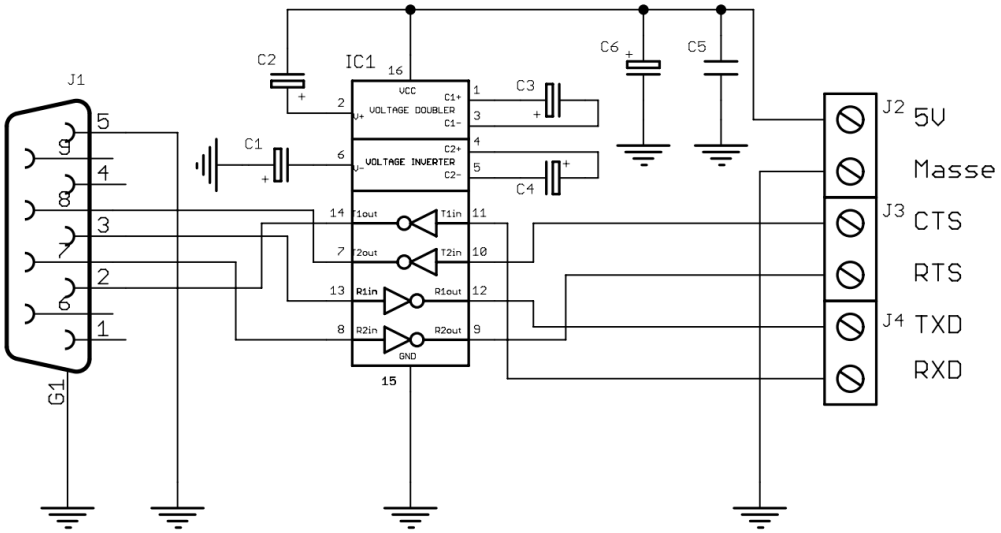

Abbildung 2.22 : Schaltbild eines RS232-TTL-Converters von Pollin

Ein RS232-TTL Wandler dient zur Pegelanpassung einer einfachen seriellen Schnittstelle (UART) im TTL-Standard (Pegelspannung 5 V), die sehr häufig bei Mikrokontroller eingesetzt wird und der standardisierten RS232-Schnittstelle, wie sie bei PCs verwendet wird. Durch den Einsatz des integrierten Schaltkreises MAX232 benötigt RS232/TTL-Wandler lediglich ein paar Elkos (Elektrolytkondensatoren), einen Kondensator und Anschlussklemmen für seine vollständige Funktion. In Abbildung 2.22 ist das Schaltbild eines RS232-TTL-Converters von Pollin zu sehen, der als Bausatz im Handel zu erwerben ist. Der für diese Aufgabe verwendete RS232/TTL-Wandler wurde bei der Firma *„Konsolen-Ersatzteile“* erworben, da dieser schon fertig bestückt war und man keine weitere Arbeit für das Löten investieren musste. Des Weiteren ist darauf zu achten, dass nicht alle RS232/TTL Wandler die gleiche TTL Pinbelegung besitzen. Wie in Abbildung 2.23 zu sehen ist, gibt es auch RS232/TTL Wandler, die lediglich 4 Anschlüsse aufweisen (VCC, GND, TXD und RXD). Dies ist für eine Datenkommunikation aber völlig ausreichend.

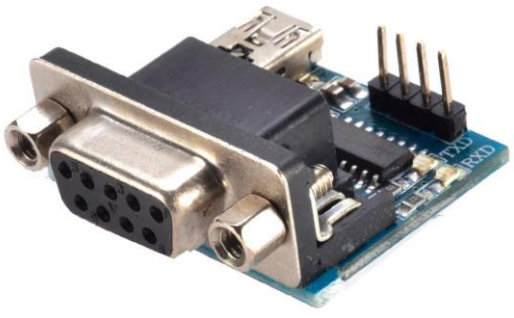

Abbildung 2.23 : RS232 zu TTL Wandler

3 Realisierung der Aufgabenstellung

In diesem Kapitel wird die Auswahl aller verwendeten Geräte näher erläutert. Des Weiteren wird beschrieben, wie alle Systeme miteinander verbunden werden müssen, damit eine spätere Kommunikation möglich ist.

3.1 Auswahl geeigneter Bauteile

3.1.1 Auswahl des RS232 zu KNX Gateways

Betrachtet man sich eine Brandmeldeanlage mit etwa 300 Teilnehmern (Brandmelder etc.) muss man kurzer Hand feststellen, dass man ein Gateway benötigt, das eine Vielzahl von Meldungen aufnehmen muss. Da man bei dem Zennio SKX OPEN allerdings nur 48 Kommunikationsobjekte integrieren kann, scheidet dieses bei der Erfüllung der Aufgabe schon mal aus. Am geeignetsten ist das Gateway von „Elka Elektroniks“, da dieses bis zu 500 unterschiedliche Datentelegramme aufnehmen kann und zusätzlich eine bessere Programmierbarkeit im Vergleich zum Gateway von „arcus-eds“ hervorbringt. Beim dem Gateway von „arcus-eds“ besteht zusätzlich ein größerer Aufwand durch das Erlernen der Programmierung, dass bei dem Gateway Elka entfällt, da es eine innovative Programmsoftware mitliefert. Des Weiteren kann das Gateway von „arcus-eds“ nur etwa 200 Datentelegramme verarbeiten, das bei einer großen Brandmeldeanlage von Nachteil sein könnte.

3.1.2 Auswahl des Programmiergerätes

Die Auswahl des Programmiergerätes ist wie bereits erwähnt wurde relativ groß. Somit resultieren eine Menge Geräte, die für die Lösung der Aufgabe gut geeignet sind. Am sinnvollsten erschließt sich aber die Verwendung eines Evaluationsboards, da man dadurch die Möglichkeit besitzt, durch zusätzliche Hardware einfache programmtechnische Sachen zu testen. Besonders achten muss man auf die Verfügbarkeit heutiger Schnittstellen. Zum Beispiel ist es heutzutage immer weniger üblich, dass ein PC-System bzw. Laptop über eine RS232 Schnittstelle verfügt. Daher wäre eine Verwendung einer USB-Schnittstelle zur Programmierung des Mikrocontrollers sehr ratsam. Aufgrund der Tatsache, dass das aTeVaL Evaluationsboard eine USB-ISP-Schnittstelle verfügt, ist die Verwendung dieser gut möglich. Vergleicht man das Atmel Evaluationsboard von Pollin, ist dieses für die meisten heutigen PC-Systeme nicht geeignet, da dieses nur über eine RS232-ISP-Schnittstelle verfügt. Man würde dafür ein weiteres externes Programmiergerät verwenden müssen, das dadurch natürlich auch höhere Kosten verursachen würde. Aufgrund dieser Tatsachen wird für die Programmierung des ATmega324A das aTeVaL Evaluationsboard verwendet.

3.2 Anschluss aller Systeme

In diesem Kapitel soll nochmals zusammengefasst werden welche Geräte zur Realisierung der Aufgabe benötigt und wie diese verbunden werden.

3.2.1 Anschlussverbindung des RS232/KNX Gateways mit dem KNX-System

Zu allererst muss die Verbindung zwischen dem KNX System und dem RS232-KNX Gateway hergestellt werden. Dies erfolgt über die KNX Twisted Pair (KNX TP) Busleitung, die sowohl alle Busteilnehmer mit Daten, als auch mit der nötigen Betriebsspannung versorgt. Die Nennspannung des Bussystems beträgt 24 V. Die Spannungsversorgungen speisen 29 V in das System ein. Die Busteilnehmer arbeiten bei Spannungen bis zu 21 V fehlerfrei. Es steht also ein Toleranzbereich von 8 V zur Verfügung, um eventuelle Spannungsabfälle auf der Leitung und an Kontaktwiederständen abzupuffern.

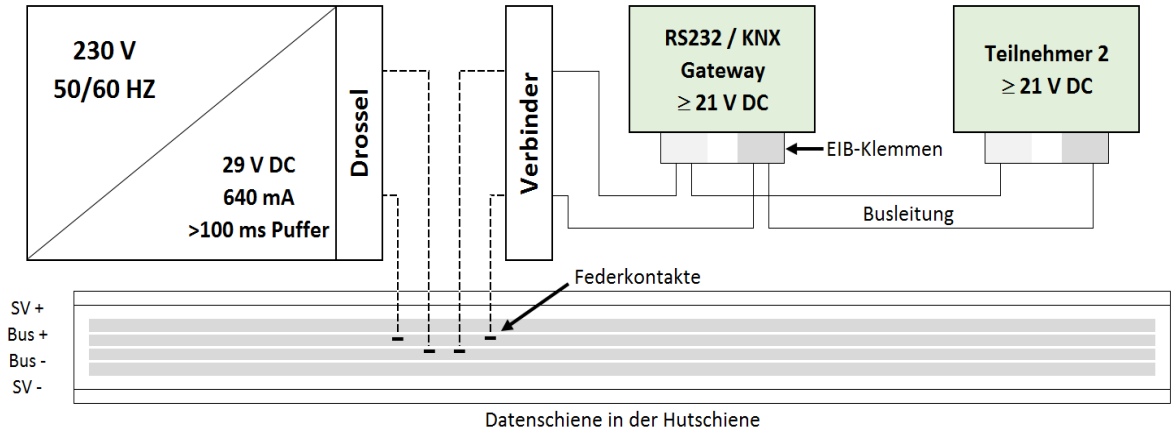

Abbildung 3.1 : Anschluss des RS232/KNX Gateways

Die Spannungsversorgung wird auf eine sogenannte Hutschiene, welche die Datenschiene integriert, mit allen Teilnehmern einer Linie verbunden. Siehe Abbildung 3.1. Die genauen Bedingungen in der Topologie eines KNX-Netzes sollen hier nicht weiter berücksichtigt werden. Diese können auf der Internetseite (www.knx.org) der KNX Association näher recherchiert werden. Es muss allerdings darauf geachtet werden, dass das RS232/KNX-Gateway nach dem Anschluss an der Hutschiene noch nicht betriebsbereit ist.
Wie in Abschnitt 2.1.2 näher erläutert wurde, muss zunächst noch die jeweilige Anwendungssoftware geladen werden, die physikalische Adresse und die verwendeten Gruppenadressen vergeben werden. In der Regel werden alle Abfolgen durch die ETS Software realisiert. Es besteht aber auch die Möglichkeit, dass Geräte über eine eigene vom Hersteller entwickelte Software programmiert werden können und nicht mit Hilfe der ETS Software. Bei dem RS232/KNX Gateway von Elka wird dabei die Software „KNX-Gate2“ verwendet. Die ETS Software muss jedoch in irgendeiner Art und Weise erkennen, dass ein RS232/KNX Gateway am KNX System angeschlossen ist. Deshalb wird dem Gateway in der ETS Software eine sogenannte Dummy-Applikation zugewiesen. Diese Applikation sorgt dafür, dass das Gateway eine physikalische Adresse erhält und somit vom System erkannt wird. Die physikalische Adresse, die mit Hilfe der Dummy Applikation in der ETS Software eingestellt wurde, muss aber zusätzlich auch in der KNX-Gate2 zugewiesen werden. Dieser Ablauf wird in Abschnitt 5.1 näher erläutert.

3.2.2 Verbindung des RS232/KNX Gateways mit dem RS232/TTL Wandler

Nachdem das RS232/KNX Gateway von „Elka Elektroniks“ mit dem KNX-System verbunden wurde, muss dieses über die RS232 Schnittstelle mit dem Mikrocontroller verbunden werden. Dabei wird ein RS232 zu TTL Wandler zu Hilfe genommen. Beide RS232-Schnittstellen besitzen jeweils einen RS232 Kabelstecker, mit einer weiblichen 9-Pin Pinbelegung.

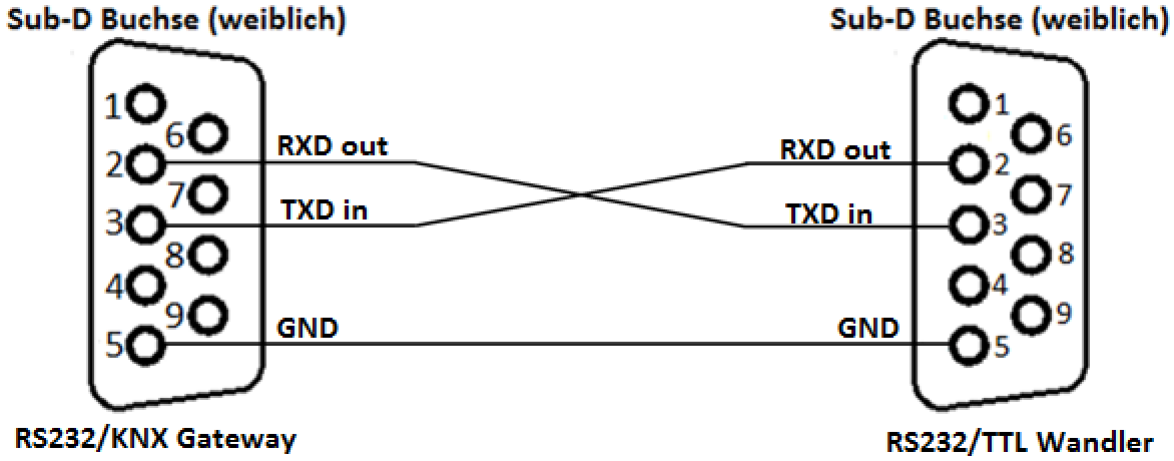

Abbildung 3.2 : Kreuzung der Anschlüsse

Bei der Verbindung von zwei RS232-Schnittstellen müssen zwei Arten von Verkabelungen beachtet werden. Handelt es sich um eine Verbindung von Rechner (DTE, meistens Stecker) zu einem Modem (DCE, meistens Buchse) benötigt man ein 1 zu 1 Kabel. Handelt es sich dagegen um eine Verbindung zweier Geräte gleichen Typs (z.B. zweier PCs), so sind die Leitungen zu kreuzen (siehe Abbildung 3.2). Ein solches Kabel nennt man Nullmodem-kabel (siehe Abbildung 3.3), da kein Modem (also „0 Modems“) eingesetzt wird. Da bei der Verwendung des RS232/KNX Gateways und des RS232/TTL Wandlers ebenfalls zwei Geräte des gleichen Typs genutzt werden, muss in diesem Fall ein gekreuztes Kabel verwendet werden. Da ein Nullmodem-Kabel allerdings ebenfalls über zwei Buchsen verfügt, ist die Verwendung nicht ohne weiteres möglich. Um die Verbindung dennoch realisieren zu können, werden zwei sogenannte Gender Changer (zu Deutsch, Geschlechtsumwandler) verwendet. Diese ermöglichen das Wandeln einer weiblichen, zu einer männlichen Pinbelegung. Damit verfügt das Nullmodem-Kabel über zwei Stecker (Kabelstecker mit männlicher Pinbelegung) und kann so für die Verknüpfung des RS232/KNX Gateways und des RS232/TTL Wandlers genutzt werden.

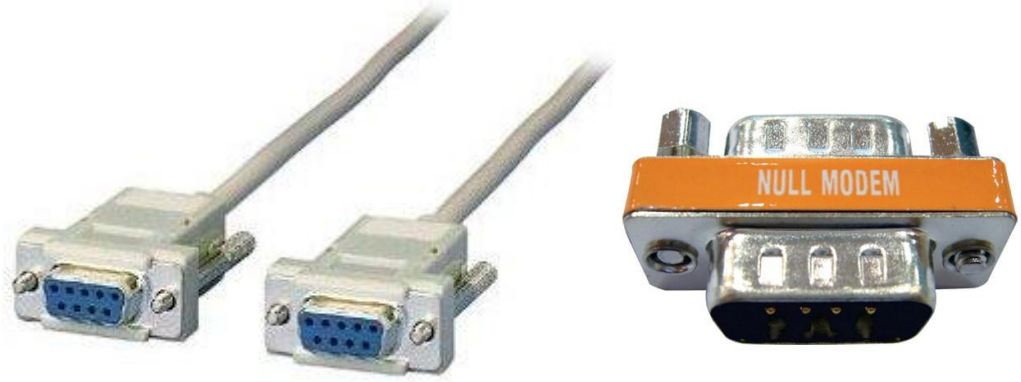

Abbildung 3.3 : Nullmodem-Kabel in Verbindung eines Gender Changer

3.2.3 Verbindung des RS232/TTL Wandlers mit dem Mikrocontroller

Nun muss realisiert werden, dass die zwei RS232-TTL-Wandler richtig mit dem Mikrocontroller verbunden werden. Dies soll anhand der Abbildung 3.4 näher erläutert werden. Wie bereits im Abschnitt 2.2.2 erläutert wurde, ist die heutige Pinbezeichnung der RS232-Schnittstelle sehr irreführend, vor allem bei der Verwendung mit Mikrocontrollern. Daher sollten die hier erklärten Verbindungen nicht allgemein geltend gemacht werden. Laut Definition der Pinbelegung einer RS232-Schnittstelle, verhalten

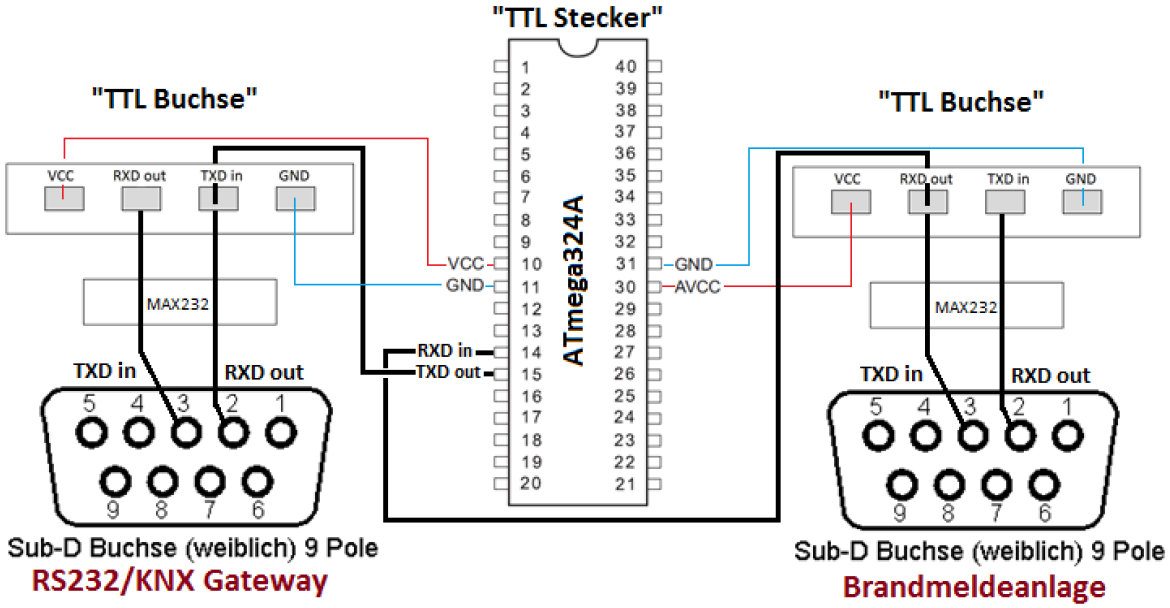

Abbildung 3.4 : TTL - Anschlussverbindungen

sich die Bauteile in diesem Fall jedoch richtig. Da es bei der Verbindung von RS232 Schnittstellen Buchsen und Stecker gibt, und deren Pinbelegung daher leicht zu erkennen sind, kann man die Anschlüsse „RXD in“ und „RXD out“ bzw. „TXD out“ und „TXD in“ leicht voneinander unterscheiden. Nach der Konvertierung von RS232 zu TTL ist dies dann allerdings nicht mehr ganz so leicht nachzuvollziehen, da viele Hersteller unterschiedliche Bezeichnungen der Ein- bzw. Ausgänge (statt „TXD in“ wird zum Beispiel „RXD“ als Beschriftung beschrieben) verwenden. Daher ist eine allgemeine Erläuterung der Beschaltung nicht möglich. Bei dem hier verwendeten RS232/TTL Wandler ist die Beschaltung der jeweiligen Ein- bzw. Ausgänge der RS232 Schnittstelle und der TTL

Schnittstelle gekreuzt vorgenommen wurden (korrekte Anwendung der definierten Pinbelegung der Buchse). Am einfachsten ist es, sich die TTL Pinbelegung des RS232/TTL Wandlers als Buchse („TTL Buchse“) und die Pinbelegung des Mikrocontrollers (da dieser eine Art PC darstellt) als Stecker („TTL Stecker“) vorzustellen. Da dadurch zwei Geräte unterschiedlichen Typs aufeinander treffen, benötigt man eine 1 zu 1 Verkabelung („RXD out“ auf „RXD in“ und „TXD in“ auf „TXD out“). Wie bereits erwähnt, muss im inneren des RS232/TTL Wandlers eine Kreuzung der Pins vorgenommen werden, da es sich um die Verwendung zweier Geräte „gleichen“ Typs handelt (RS232 Buchse und „TTL Buchse“).

Tabelle 3.1 : Pinbelegungen des 40-poligen Wannensteckers – Teil 1

Pin	Atmega 16/32/8535	Funktion Mega16/32	Atmega8	Funktion	Attiny2313	Funktion	Attiny12/15	Funktion Attiny12/13	Funktion Attiny15	EEPROM 24Cxx	Funktion
1	40	PA0	23	PC0	n.c.	-	n.c.	-	-	n.c.	-
2	39	PA1	24	PC1	n.c.	-	n.c.	-	-	n.c.	-
3	38	PA2	25	PC2	n.c.	-	n.c.	-	-	n.c.	-
4	37	PA3	26	PC3	n.c.	-	n.c.	-	-	n.c.	-
5	36	PA4	27	PC4	n.c.	-	n.c.	-	-	5	SDA
6	35	PA5	28	PC5	n.c.	-	n.c.	-	-	6	SCL
7	34	PA6	n.c.	-	n.c.	-	n.c.	-	-	n.c.	-
8	33	PA7	n.c.	-	n.c.	-	n.c.	-	-	n.c.	-
9	1	PB0	n.c.	-	12	PB0	n.c.	-	-	n.c.	-
10	2	PB1	n.c.	-	13	PB1	n.c.	-	-	n.c.	-
11	3	PB2	14	PB0	14	PB2	n.c.	-	-	n.c.	-
12	4	PB3	15	PB1	15	PB3	3	PB4	PB3	n.c.	-
13	5	PB4	16	PB2	16	PB4	2	PB3	PB4	n.c.	-
14	6	PB5	17	PB3	17	PB5	5	PB0	PB0	n.c.	-
15	7	PB6	18	PB4	18	PB6	6	PB1	PB1	n.c.	-
16	8	PB7	19	PB5	19	PB7	7	PB2	PB2	n.c.	-
17	22	PC0	n.c.	-	n.c.	-	n.c.	-	-	6	SCL
18	23	PC1	n.c.	-	n.c.	-	n.c.	-	-	5	SDA
19	24	PC2	n.c.	-	n.c.	-	n.c.	-	-	n.c.	-
20	n.c.	-	n.c.	-	n.c.	-	n.c.	-	-	n.c.	-
21	25	PC3	n.c.	-	n.c.	-	n.c.	-	-	n.c.	-
22	26	PC4	n.c.	-	n.c.	-	n.c.	-	-	n.c.	-
23	27	PC5	n.c.	-	n.c.	-	n.c.	-	-	n.c.	-
24	28	PC6	n.c.	-	n.c.	-	n.c.	-	-	n.c.	-
25	29	PC7	n.c.	-	n.c.	-	n.c.	-	-	n.c.	-
26	14	PD0	2	PD0	2	PD0	n.c.	-	-	n.c.	-
27	15	PD1	3	PD1	3	PD1	n.c.	-	-	n.c.	-
28	16	PD2	4	PD2	6	PD2	n.c.	-	-	n.c.	-

Tabelle 3.2 : Pinbelegungen des 40-poligen Wannensteckers – Teil 2

Pin	Atmega 16/32/8535	Funktion Mega16/32	Atmega8	Funktion	Attiny2313	Funktion	Attiny12/15	Funktion Attiny12/13	Funktion Attiny15	EEPROM 24Cxx	Funktion
29	17	PD3	5	PD3	7	PD3	n.c.	-	-	n.c.	-
30	18	PD4	6	PD4	8	PD4	n.c.	-	-	n.c.	-
31	19	PD5	11	PD5	9	PD5	n.c.	-	-	n.c.	-
32	20	PD6	12	PD6	11	PD6	n.c.	-	-	n.c.	-
33	21	PD7	13	PD7	n.c.	-	n.c.	-	-	n.c.	-
34	32	AREF	21	AREF	n.c.	-	n.c.	-	-	n.c.	-
35	11,31	GND	8,22	GND	10	GND	4	GND	GND	4	GND
36	10,30	VCC	7,20	VCC	20	VCC	8	VCC	VCC	8	VCC
37	11,31	GND	8,22	GND	10	GND	4	GND	GND	4	GND
38	10,30	VCC	7,20	VCC	20	VCC	8	VCC	VCC	8	VCC
39	11,31	GND	8,22	GND	10	GND	4	GND	GND	4	GND
40	10,30	VCC	7,20	VCC	20	VCC	8	VCC	VCC	8	VCC

Da für die Spannungsversorgung des Mikrocontrollers das aTeVaL Evaluationsboard benutzt wird, um die Spannungsversorgung bereitzustellen, wird zur Verbindung der Pins der 40-polige Wannenstecker verwendet. Um herauszufinden, welcher Pin des Wannensteckers zu welchem Pin des Mikrocontrollers gehört, muss zusätzlich in das Datenblatt des Evaluationsboard geschaut werden. Dort findet man eine entsprechende Tabelle, in der alle Pinbelegungen zusammengefasst sind. Dabei sind des Weiteren auch die Pins des „TXD out“ und des „RXD in“ zu entnehmen, da diese ebenfalls über die 40-poligen Wannenstecker an den Mikrocontroller angeschlossen werden.

3.2.4 Verbindung des RS232/TTL Wandlers mit der BMZ

Schließlich muss nur noch der Mikrocontroller mit der Brandmeldeanlage verbunden werden. Da die Brandmeldeanlage ebenfalls einen RS232 Kabelstecker besitzt, der über eine weiblichen 9-Pin Pinbelegung verfügt, erfolgt der Aufbau nach dem gleichen Schema wie bei der Verbindung des RS232/KNX Gateways mit dem ersten RS232/TTL Wandlers, durch Verwendung eines Nullmodem-Kabels und zweier Gender Changer.

3.2.5 Verbindung des externen Schwingquarzes mit dem Mikrocontroller

Die Verbindung des Schwingquarzes mit dem ATmega324A lässt sich sehr leicht realisieren, da der Mikrocontroller zwei separate Pinanschlüsse (XTAL1 und XTAL2) mit sich bringt (siehe Abbildung 2.13). Mit diesen Anschlüssen lassen sich externe Oszillatoren anschließen. XTAL bedeutet so viel wie Crystal (in Deutsch, Schwingquarz). Das aTeVaL Evaluationsboard besitzt dafür eigene Sockelanschlüsse, über die man Schwingquarze an eingelegte Mikrocontroller anschließen kann (siehe Abbildung 2.20). In diesem Fall muss man den Sockel Q3 verwenden, da der ATmega324A auf den IC4 aufgebracht ist.

4 Programmierung des Mikrocontrollers

4.1 Programmablaufplan

Nachdem man die Schaltung realisiert hat, erfolgt nun die Programmierung des Mikrocontrollers. Die Hauptaufgabe des Mikrocontrollers ist es darauf zu warten, dass ein Datentelegramm eintrifft. Dieses soll dann im Anschluss bearbeitet werden, sodass die erhaltene Information entsprechend gefiltert und anschließend zum Gateway versendet werden kann. Wie schon erwähnt, gilt es die Statusmeldung an sich und die dazugehörige Meldernummer zu erkennen und zu filtern (siehe Tabelle 2.6). Das Problem besteht darin aber, dass die einzelnen Informationen in getrennten Strings übertragen werden. Das heißt, dass der Mikrocontroller die eingetretene Statusmeldung (siehe Abbildung 4.1, String 1) zunächst filtern und abspeichern muss. Danach muss er den nächsten eingetroffenen String analysieren, die Meldernummer (siehe Abbildung 4.1, String 2) ermitteln und diese zusammen mit der abgespeicherten Statusmeldung ausgeben. Lediglich bei der Meldung „ZENTRALE ZURÜCKSETZEN“ kann der gefilterte Text sofort ausgegeben werden, da dieser in keinem Zusammenhang mit einer Meldernummer steht.

```
1  65437: Fre 18-Jul-14 00:03:40:   * FEUERALARM *\r\n
2  R1 Mod.   4 - Gruppe 0004/01 - DKM \r\n
3  Grup.:Mitte\r\n
4   Ort: Handmelder Messw.:114%\r\n
```

Abbildung 4.1 : Beispiel einer Textausgabe der BMZ

Der genaue Ablauf des Programms soll nun an dem Programmablaufplan in Abbildung 4.2 näher erläutert werden.
Zu aller erst wartet der Mikrocontroller auf das Eintreffen eines Datentelegramms, welches über die UART Schnittstelle empfangen wird. Nachdem ein Textstring über die UART Schnittstelle eingetroffen ist, nimmt er diesen auf, in dem er ihn in eine Zeichenkette abspeichert. Diese Zeichenkette soll entsprechend nach einer Statusmeldung durchsucht werden und diese in eine weitere Zeichenkette als Zwischenspeicher hinterlegen. Nach Eintreffen des zweiten Strings, wird die darin enthaltene Meldernummer gefiltert. Da es eine Vielzahl an Meldernummern geben kann, wäre es extrem aufwendig jede Nummer einzelnen zu suchen. Daher wird jeder eintreffende String auch nach dem Wort „Gruppe“ durchsucht, um festzustellen, ob der String eine Meldernummer enthält. Wenn festgestellt wird, dass das Wort „Gruppe“ enthalten und somit auch eine Meldernummer vorhanden ist, wird durch ein weiteres Verfahren die Meldernummer gefiltert (wird in Abschnitt 4.5 erläutert). Es ist darauf zu achten, dass die Meldernummer erst dann gefiltert wird, wenn vorher auch eine Statusmeldung abgespeichert wurde und somit der Zwischenspeicher „voll“ ist. Zusätzlich sei gesagt, dass der Mikrocontroller nach jedem Eintreffen eines Strings, diesen auch immer nach einer Statusmeldung bzw. nach dem Wort „Gruppe“ durchsucht, auch wenn der Zwischenspeicher bereits voll sein sollte (auch bei Eintreffen des zweiten Strings). Dies hat aufgrund von bestimmten Bedingungen aber keinen Einfluss auf den darauf folgenden Ablauf des Programms.
Nachdem eine Statusmeldung entsprechend abgespeichert und eine dazugehörige Meldernummer ermittelt wurde, sollen diese beiden Informationen zusammengefasst

und als String zum RS232/KNX Gateway gesendet werden. Eine Ausnahme spielt dabei, wie bereits erläutert wurde, die Statusmeldung „ZENTRALE ZURÜCKSETZEN“, die bei einem Auftreten ohne weiteres direkt zum RS232/KNX Gateway weitergeleitet werden soll. Nach dem Ausgeben der gefilterten Textnachricht soll er anschließend wieder auf ein neues Eintreffen eines Strings warten und somit die gleiche Abfolge erneut ausführen.

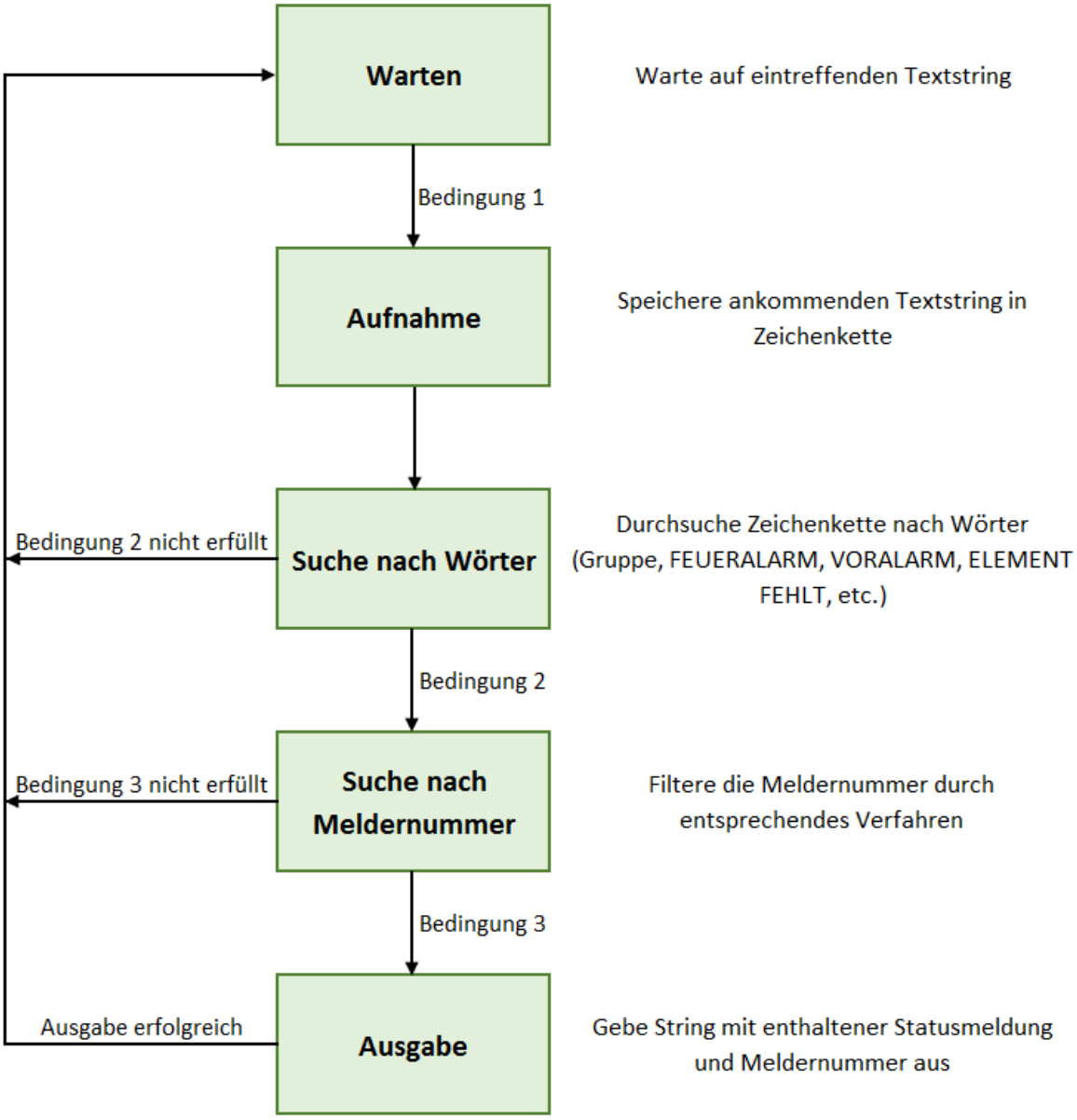

Abbildung 4.2 : Programmablaufplan

Alle Ablaufbedingungen noch einmal zusammengefasst:

- Bedingung 1 – Textstring wurde von der BMZ gesendet
- Bedingung 2 – Statusmeldung ist im Zwischenspeicher hinterlegt
- Bedingung 3 – Zwischenspeicher enthält eine Statusmeldung und Meldernummer wurde erfolgreich gefiltert

4.2 Allgemeiner Aufbau des C-Programms

Damit sich der Leser nicht im Quellcode verliert, soll anhand von Tabelle 4.1 der Aufbau der Datei dargestellt werden. Der Aufbau der Grafik erinnert grob an ein Struktogramm. Vorn in der Datei befinden sich die grundsätzlichen Dinge, wie include-Anweisungen zum Einfügen von Dateien, Deklarationen, Makros, Variablen und Konstanten. Danach folgen drei Unterprogramme, die für das Empfangen und Senden von Zeichenketten notwendig sind. Abschließend kommt das Hauptprogramm „main“, das den hauptsächlichen Ablauf des Programms beinhaltet. Dieses ist, mit Hilfe einer while-Schleife, in einer Endlosschleife eingebunden. Die Abfolge des Hauptprogramms gliedert sich in mehreren Schritten. Zuerst müssen noch weitere Variablen und Zeichenketten definiert werden. Daraufhin werden die Textinformationen der Brandmeldeanlage empfangen, die dann nach den gesuchten Statusmeldungen und Meldernummern durchsucht werden. Daraufhin folgt die Übertragung zum RS232/KNX Gateway.

Tabelle 4.1 : Programmübersicht

Include-Anweisungen, Makros, Definitionen von Konstanten
Unterprogramme Initialisierung Empfangen von Zeichenketten Senden von Zeichenketten
Hauptprogramm Definition von Variablen und Logiken Empfangen eines Textstrings Suche nach Wörter Filterung der Meldernummer Ausgabe zum RS232/KNX Gateway

Die einzelnen Abfolgen des Programms werden nun in den nachfolgenden Kapiteln näher erläutert.

4.3 Empfang und Senden von Textstrings

Zu aller erst soll realisiert werden, dass der Mikrocontroller einfache Textstrings aufnehmen bzw. versenden kann. Mit Hilfe eines Laptops werden Nachrichten zum Mikrocontroller gesendet, die anschließend von diesem aufgenommen und daraufhin wieder zum Laptop zurück geschickt werden. Es müssen logischerweise dieselben Textinformationen im Laptop eintreffen, die vorher auch versendet wurden.

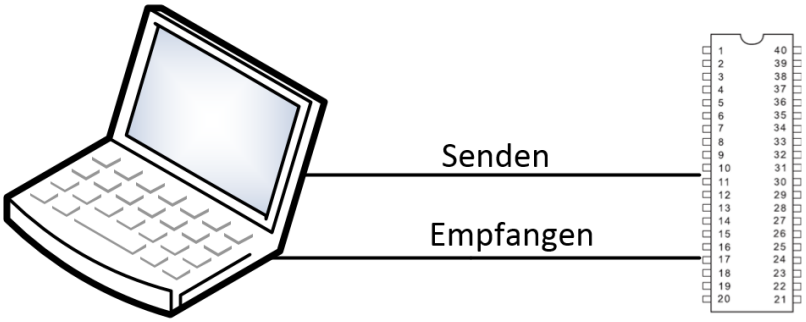

Abbildung 4.3 : Senden und Empfangen mit Hilfe eines Laptops

Bei der Übertragung von Informationen über eine RS232 Schnittstelle ist es wichtig, dass beide Teilnehmer die gleiche Baudrate, sowie die gleiche Länge der Übertragenen Daten- und Stoppbits eingestellt haben. Die Einstellung des Laptops erfolgt über das Programm *HTERM,* welches zur Übertragung verwendet wird. Unter Windows können dabei auch andere Programme, wie *HyperTerminal* oder *Serial Port Monitor,* verwendet werden.
Da die Brandmeldeanlage standardmäßig mit einer Baudrate von 9600 Baud, 8 Datenbits und 1 Stoppbit arbeitet, werden diese Werte dementsprechend berücksichtigt und im HTERM Programm eingestellt.

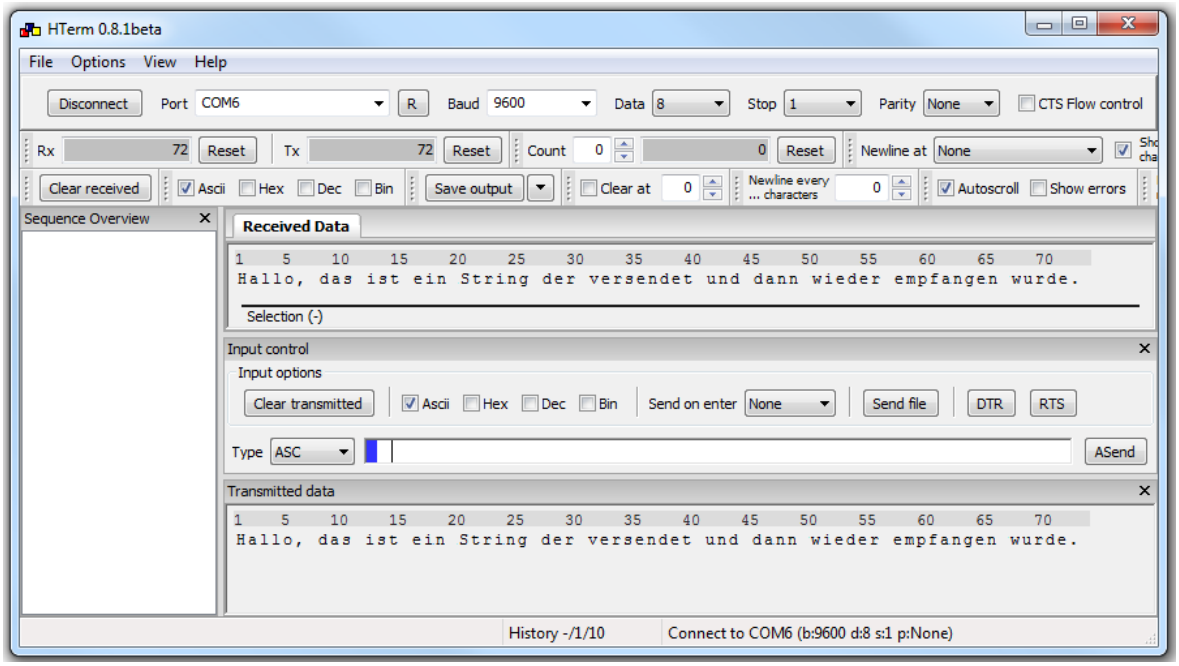

Abbildung 4.4 : Übertragung von Textinformationen mit Hilfe von HTerm 0.8.1beta

Wie in Abbildung 4.4 zu sehen ist, werden im HTERM Programm einerseits die Empfangenen und andererseits die gesendeten Daten angezeigt. Bei der Verbindung zwischen Laptop und Mikrocontroller wird das Evaluationsbaord aTeVaL verwendet, da es ohne weiteres möglich ist eine Verknüpfung der USB-Schnittstelle und der UART Schnittstelle des Mikrocontrollers zu realisieren. Da die Baudrate, Datenbit- und Stoppbitlänge am Laptop bereits eingestellt wurden, muss man anschließend dasselbe am

Mikrocontroller vornehmen. Mikrocontroller haben verschiedene Register die standardmäßig, je nach Modell und Hersteller, vordefiniert sind. Der UART wird dabei über vier separate Register angesprochen. Die USARTs der ATMEGAs verfügen über mehrere zusätzliche Konfigurationsregister, die man aus dem Datenblatt entnehmen kann. Bevor man eine UART Schnittstelle verwendet, muss man diese in der C Programmierung dementsprechend Initialisieren. Zunächst soll allerdings erst ein kleiner Einblick in den entsprechenden Registern verschafft werden.

4.3.1 UART Register

UCSRA Register

Im „UART Control and Status Register A“ teilt der UART mit, was er gerade so macht.

Tabelle 4.2 : UCSRA Resgister

Bit	7	6	5	4	3	2	1	0
Name	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

RXC (UART Receive Complete)

Dieses Bit wird vom AVR gesetzt, wenn ein empfangenes Zeichen vom Empfangs-Schieberegister in das Empfangs-Datenregister transferiert wurde. Das Zeichen muss nun schnellstmöglich aus dem Datenregister ausgelesen werden. Falls dies nicht erfolgt, bevor ein weiteres Zeichen komplett empfangen wurde, wird eine Überlauf-Fehlersituation eintreten. Mit dem Auslesen des Datenregisters wird das Bit automatisch gelöscht.

TXC (UART Transmit Complete)

Dieses Bit wird vom AVR gesetzt, wenn das im Sende-Schieberegister befindliche Zeichen vollständig ausgegeben wurde und kein weiteres Zeichen im Sendedatenregister ansteht. Dies bedeutet also, dass die Kommunikation vollumfänglich abgeschlossen ist. Dieses Bit ist wichtig bei Halbduplex-Verbindungen, wenn das Programm nach dem Senden von Daten auf Empfang schalten muss. Im Vollduplexbetrieb braucht man dieses Bit nicht zu beachten. Das Bit wird nur dann automatisch gelöscht, wenn der entsprechende Interrupthandler aufgerufen wird, ansonsten muss man das Bit selber löschen. Um das Bit zu löschen, muss eine 1 an die entsprechende Position geschrieben werden.

UDRE (UART Data Register Empty)

Dieses Bit zeigt an, ob der Sendepuffer bereit ist, um ein zu sendendes Zeichen aufzunehmen. Das Bit wird vom AVR gesetzt (1), wenn der Sendepuffer leer ist. Es wird gelöscht (0), wenn ein Zeichen im Sendedatenregister vorhanden ist und noch nicht in das Sende-Schieberegister übernommen wurde.

FE (Framing Error)

Dieses Bit wird vom AVR gesetzt, wenn der UART einen Zeichenrahmenfehler detektiert, d.h. wenn das Stopbit eines empfangenen Zeichens 0 ist. Das Bit wird automatisch gelöscht, wenn das Stopbit des empfangenen Zeichens 1 ist.

DOR (Data Over Run)

Dieses Bit wird vom AVR gesetzt, wenn das Programm das im Empfangsdatenregister bereit liegende Zeichen nicht abholt, bevor das nachfolgende Zeichen komplett empfangen wurde. Das nachfolgende Zeichen wird verworfen. Das Bit wird automatisch gelöscht, wenn das empfangene Zeichen in das Empfangsdatenregister transferiert werden konnte.

PE (Parity Error)

Dieses Bit wird vom AVR gesetzt, wenn das im Empfangsdatenregister bereit liegende Zeichen einen Paritätsfehler aufweist. Das Bit wird automatisch gelöscht, wenn das empfangene Zeichen in das Empfangsdatenregister transferiert werden konnte.

U2X (Double the Transmission Speed)

Dieses Bit wird lediglich im asynchronen Modus genutzt. Im synchronen Modus ist es „0“ zu setzen. Wird das Bit gesetzt, so wird der Baudraten Divisor von 16 auf 8 reduziert, was einer Verdopplung der Transferrate gleich kommt.

MPCM (Multi Prozessor Communication Mode)

Dieses Bit aktiviert die Multi-Prozessor-Kommunikation. Jeder eintreffende Frame der keine Adressinformation enthält wird dadurch ignoriert.

UCSRB Register

Im „UART Control and Status Register B“ stellt man ein, wie man den UART verwenden möchte.

Tabelle 4.3 : UCSRB Register

Bit	7	6	5	4	3	2	1	0
Name	**RXCIE**	**TXCIE**	**UDRIE**	**RXEN**	**TXEN**	**UCSZ2**	**RXB8**	**TXB8**

RXCIE (RX Complete Interrupt Enable)

Wenn dieses Bit gesetzt ist, wird ein UART RX Complete Interrupt ausgelöst, wenn ein Zeichen vom UART empfangen wurde. Das Global Enable Interrupt Flag muss selbstverständlich auch gesetzt sein.

TXCIE (TX Complete Interrupt Enable)

Wenn dieses Bit gesetzt ist, wird ein UART TX Complete Interrupt ausgelöst, wenn ein Zeichen vom UART gesendet wurde. Das Global Enable Interrupt Flag muss selbstverständlich auch gesetzt sein.

UDRIE (**U**ART **D**ata **R**egister **E**mpty **I**nterrupt **E**nable)
Wenn dieses Bit gesetzt ist, wird ein UART Datenregister Empty Interrupt ausgelöst, wenn der UART wieder bereit ist um ein neues zu sendendes Zeichen zu übernehmen. Das Global Enable Interrupt Flag muss selbstverständlich auch gesetzt sein.

RXEN (**R**eceiver **E**nable)
Nur wenn dieses Bit gesetzt ist, arbeitet der Empfänger des UART überhaupt. Wenn das Bit nicht gesetzt ist, kann der entsprechende Pin des AVR als normaler I/O-Pin verwendet werden.

TXEN (**T**ransmitter **E**nable)
Nur wenn dieses Bit gesetzt ist, arbeitet der Sender des UART überhaupt. Wenn das Bit nicht gesetzt ist, kann der entsprechende Pin des AVR als normaler I/O-Pin verwendet werden.

UCSZ2 (Characters Size)
Dieses Bit setzt in Verbindung mit den UCSZ1:0 Bits im UCSRC Register die Anzahl von Datenbits eines Frames beim Empfang oder Senden.

RXB8 (Receive Data Bit 8)
Wenn das vorher erwähnte UCSZ2-Bit gesetzt ist, dann enthält dieses Bit das 9. Datenbit eines empfangenen Zeichens.

TXB8 (Transmit Data Bit 8)
Wenn das vorher erwähnte UCSZ2-Bit gesetzt ist, dann muss in dieses Bit das 9. Bit des zu sendenden Zeichens eingeschrieben werden bevor das eigentliche Datenbyte in das Datenregister geschrieben wird.

UCSRC Register (UART Control and Status Register C)

Tabelle 4.4 : UCSRC Register

Bit	7	6	5	4	3	2	1	0
Name	**URSEL**	**UMSEL**	**UPM1**	**UPM0**	**USBS**	**UCSZ1**	**UCSZ0**	**UCPOL**

URSEL (USART Register Select)

Dieses Bit selektiert die Auswahl des UCSRC- bzw. des UBRRH Registers. Beim Lesen von UCSRC wird es als 1 gelesen. Beim Schreiben auf UCSRC muss es auf 1 gesetzt werden.
Um Zugriffsadressen einzusparen, wurde von Atmel ein etwas seltsamer Weg gewählt. Das UCSRC Register und das High-Byte des Baudratenregisters teilen sich dieselbe Registeradresse, um der Hardware mitzuteilen, welche Bedeutung ein zugewiesener Wert haben soll, entweder neue Belegung des Baudratenregisters oder eben zur Konfiguration des UCSRC Registers. Ist es nicht gesetzt, dann wird eine Zuweisung immer als Zuweisung an das High-Byte des Baudratenregisters angesehen, selbst wenn das so nicht beabsichtigt war. Nur dann wenn dieses Bit gesetzt ist, dann wird eine Zuweisung auch tatsächlich als eine Zuweisung an das UCSRC Register gewertet und die Konfiguration

verändert. Lässt man das Bit irrtümlich weg, dann verursacht eine Zuweisung an UCSRC eine Veränderung der Baudrateneinstellung.

UMSEL (USART Mode Select)

Durch dieses Bit kann eine asynchrone oder synchrone Übertragung eingestellt werden. Durch Setzen des Bits wird eine synchrone Übertragung eingestellt.

UPM1:0 (Parity Mode)

Durch diese Bits wird die Art der Parität festgelegt.

Tabelle 4.5 : UPM1:0 Definierung

UPM1	UPM0	Parity Mode
0	0	keine Parität
0	1	entfällt
1	0	1 Paritätsbit
1	1	2 Paritätsbits

USBS (USART Stop Bit Select)

Dieses Bit setzt die Anzahl der zu sendenden Stopbits eines Frames. Beim Setzen werden 2 Stopbits übertragen, andernfalls nur 1 Stopbit.

Tabelle 4.6 : USBS Bit

USBS	Anzahl der Stop Bits
0	1
1	2

UCSZ1:0 (Character Size)

Diese Bits setzen in Verbindung mit UCSZ2 aus dem UCSRB Register die Anzahl der Datenbits eines Frames.
Diese Bits setzen den entsprechenden Paritätsmodus.

Tabelle 4.7 : UCSZ2:0 Bits

UCSZ2	UCSZ1	UCSZ0	Datenlänge
0	0	0	5-Bit
0	0	1	6-Bit
0	1	0	7-Bit
0	1	1	8-Bit
1	0	0	entfällt
1	0	1	entfällt
1	1	0	entfällt
1	1	1	9-Bit

UCPOL (Clock Polarity)

Dieses Bit bestimmt nur im Synchronbetrieb die Phasenlage der gesendeten bzw. empfangenen Datenbits in Bezug auf den Übertragungstakt. Im Asynchronbetrieb sollte das Bit „0“ sein.

UDR Register:

Im „UART Data Register“ werden Daten zwischen UART und CPU übertragen. Da der UART im Vollduplexbetrieb gleichzeitig empfangen und senden kann, handelt es sich hier physikalisch um 2 Register, die aber über die gleiche I/O-Adresse angesprochen werden. Je nachdem, ob ein Lese- oder ein Schreibzugriff auf den UART erfolgt, wird automatisch das richtige UDR angesprochen.

UBRR Register:

In dem „UART Baud Rate Register“ muss dem UART mitgeteilt werden, wie schnell man gerne kommunizieren möchten. Der Wert, der in dieses Register geschrieben werden muss, errechnet sich nach folgender Formel (wenn U2X Bit 0 gesetzt ist):

$$\mathrm{UBRR} = \frac{\mathrm{Taktfrequenz}}{\mathrm{Baudrate} \cdot 16} - 1$$

4.3.2 Initialisierung der UART Schnittstelle

Bevor man Daten mit dem UART auf die serielle Schnittstelle ausgeben möchte, muss man wie bereits erwähnt, diese zuerst initialisieren. Diese Initialisierung wird in Form eines Unterprogramms realisiert. Dazu setzt man je nach gewünschter Funktionsweise die benötigten Bits im UART Control Register. Da man lediglich nur Senden und Empfangen, und keine Interrupts auswerten muss, gestaltet sich die Initialisierung sehr einfach. Es muss dabei nur das Transmitter Enable Bit (TXEN) und das Receiver Enable Bit (RXEN) gesetzt werden.

```
UCSR0B |= (1<<TXEN0);      // TXEN setzen/einschalten
UCSR0B |= (1<<RXEN0);      // RXEN setzen/einschalten
```

Dabei ist allerdings zu achten, dass der ATmega324A zwei UARTs besitzt und die einzelnen Register dementsprechend nummeriert sind. Es wird nur der UART0 verwendet, wobei der UART1 unberücksichtigt bleibt.
Wie bereits erläutert wurde, muss bei dem Mikrocontroller zusätzlich die Baudrate, sowie die gleiche Länge der übertragenen Daten- und Stoppbits eingestellt werden. Um die Anzahl der Daten- und Stoppbits einzustellen, muss das UCSRC Register entsprechend gesetzt werden. Durch die Übertragung von 8 Daten- und einem Stoppbit muss nach Tabelle 4.7 das erste und zweite Bit auf „1“, sowie das dritte Bit auf „0“ gesetzt sein. Des Weiteren ist darauf zu achten, dass das URSEL Bit auf „1“ und durch eine asynchrone Übertragung das UMSEL Bit auf „0“ gesetzt ist. Die Paritäts-Bits werden ebenfalls auf „0“ gesetzt, da keine Parität vorliegt.

```
UCSR0C  = 0b10000110;      // Asynchron 8 Daten- und 1 Stoppbit
```

Nun ist noch das Baudratenregister UBRR der verwendeten UARTs einzustellen, bzw. die beiden Register UBRRL und UBRRH. Die Berechnung wird während des Compilerlaufs ausgeführt, beansprucht also in der gezeigten Form weder Speicher noch Rechenzeit des Controllers. Das Ergebnis wird jedoch als ganzzahliger Wert eingesetzt, d.h. Nachkommastellen werden einfach abgeschnitten und es erfolgt keine Rundung. Aus diesem Grund kann man sich eines kleinen Tricks bedienen, indem vor der eigentlichen Division bei der Zuweisung die Hälfte des Wertes dazu addiert wird. Allgemein formuliert bedeutet das: `int i = ( a + b/2 ) / b;`. Dies wird in der unten angegebenen Berechnung von UBRR_VAL ausgenutzt um den Fehler zu minimieren.

```
  /*
  UART-Init:
  Berechnung des Wertes für das Baudratenregister
  aus Taktrate und gewünschter Baudrate
  */

#ifndef F_CPU
/*In neueren Versionrn der WinAVR/Mfile Makefile-Vorlage kann
  F_CPU im Makefile definiert werden, eine nochmalige Definition
  würde hier zu einer Compilerwarnung führen. Daher "Schutz" durch
  #ifndef/#endif

  Dieser "Schutz" kann zu Debugsessions führen, wenn
  dort eine andere, nicht zur Hardware passende
  Taktrate eingestellt ist. Dann wird die folgende Definition
  nicht verwendet, sondern stattdessen der Standardwert (1 MHz)
  Daher Ausgabe einer Warnung falls F_CPU noch nicht definiert: */

#warning "F_CPU war noch nicht definiert, wird nun nachgeholt mit 7372800"
#define F_CPU 7372800UL  // Systemtakt in Hz - Definition als unsigned long
beachten
// Ohne ergeben sich unten Fehler in der Berechnung
#endif

#define BAUD 9600      // Baudrate

// Berechnungen
#define UBRR_VAL ((F_CPU+BAUD*8)/(BAUD*16)-1)   // clever runden
#define BAUD_REAL (F_CPU/(16*(UBRR_VAL+1)))     // Reale Baudrate
#define BAUD_ERROR ((BAUD_REAL*1000)/BAUD) // Fehler in Promille, 1000 = kein
Fehler.

#if ((BAUD_ERROR<990) || (BAUD_ERROR>1010))
#error Systematischer Fehler der Baudrate grösser 1% und damit zu hoch!
#endif
```

Die Makros sind sehr praktisch, da damit sowohl automatisch der Wert für UBRR, als auch die Abweichung in der generierten (möglichen) von der gewünschten Baudrate berechnet wird. Im Falle einer zu hohen Abweichung (+/-1%) wird eine Fehlermeldung ausgegeben und der Compilerablauf abgebrochen. Damit können viele Probleme vermieden werden. Außerdem kann man mühelos die Einstellung an eine neue Taktfrequenz bzw. Baudrate anpassen, ohne selber rechnen oder in Tabellen nachschlagen zu müssen.

Inzwischen gibt es in der *avr-libc* Makros für obige Berechnung der UBRR Registerwerte aus Taktrate F_CPU und Baudrate BAUD. Dazu wird die Include-Datei <util/setbaud.h> eingebunden, nachdem F_CPU und die gewünschte Baudrate

definiert wurden. Einige Beispiele zur Anwendung finden sich in der Dokumentation der avr-libc. Im Quellcode kann dann analog zur oben gezeigten Vorgehensweise einfach das Makro UBRR_VALUE (bzw. UBRRH_VALUE und UBRRL_VALUE) an der entsprechenden Stelle eingesetzt werden. Es wird auch automatisch ermittelt, ob der U2X-Modus (vgl. Datenblatt) zu geringeren Abweichungen führt und dann dem Makro USE_U2X ein Wert ungleich null zuweist. Ein Beispiel (angelehnt an die avr-libc-Dokumentation):

```
#include <avr/io.h>
#define F_CPU 7372800UL /* evtl. bereits via Compilerparameter definiert */
#define BAUD 9600
#include <util/setbaud.h>

void uart_init(void)
{
UBRRH = UBRRH_VALUE;
UBRRL = UBRRL_VALUE;
 /* evtl. verkürzt falls Register aufeinanderfolgen (vgl. Datenblatt)
UBRR = UBRR_VALUE;   */
#if USE_2X
/* U2X-Modus erforderlich */
UCSRA |= (1 << U2X);
#else
/* U2X-Modus nicht erforderlich */
UCSRA &= ~(1 << U2X);
#endif
}
```

Die vollständige Initialisierung in Bezug zur Aufgabenstellung ist nachfolgend dargestellt. Dies zeigt nur das entsprechende Unterprogramm, welches letztendlich im Hauptprogramm eingebunden werden muss.

```
.                                   .
.                                   .
19  #define BAUD 9600 //geforderte Baudrate
20  #define F_CPU 7372800UL//Geschwindigkeit des Quarzes
21
22  #include <avr/io.h> // enthält Namen der verwendeten Register
23  #include <util/setbaud.h> //definiert Baudrate im UART Register
.                                   .
.                                   .
35  /*Uart-Schnittstelle initialisieren
36  -----------------------------------*/
37  void uart_init(void)
38  {
39   UBRR0= UBRR_VALUE;
40
41   #if USE_2X
42   /* U2X-Modus erforderlich */
43   UCSR0A |= (1 << U2X0);
44   #else
45   /* U2X-Modus nicht erforderlich */
```

Abbildung 4.5 : Initialisierung der UART-Schnittstelle Teil A

```
46   UCSR0A &= ~(1 << U2X0);
47   #endif
48
49   UCSR0C  = 0b10000110; // Asynchron 8 Daten- und 1 Stoppbit
50   UCSR0B |= (1<<TXEN0); // TXEN setzen/einschalten
51   UCSR0B |= (1<<RXEN0); // RXEN setzen/einschalten
52  }
.                          .
.                          .
```

Abbildung 4.6 : Initialisierung der UART-Schnittstelle Teil B

4.3.3 Senden mit dem UART

Senden einzelner Zeichen

Um nun ein Zeichen auf die Schnittstelle auszugeben, müssen man dasselbe lediglich in das UART Data Register schreiben. Vorher ist zu prüfen, ob das UART-Modul bereit ist, das zu sendende Zeichen entgegenzunehmen. Dabei wird verglichen, welchen Wert das UDR0E Bit aufweist (siehe Tabelle 4.2). Wenn das UDR0E-Bit vom Mikrocontroller auf „1“ gesetzt ist, kann anschließend ein Zeichen gesendet werden.

```
while (!(UCSR0A & (1<<UDR0E)))  /* warten bis Senden möglich  */
{
}
UDR0 = 'z';    /* schreibt das Zeichen z auf die Schnittstelle */
```

Senden einer Zeichenkette

Die Aufgabe "String senden" wird durch zwei Funktionen abgearbeitet. Die universelle/controllerunabhängige Funktion uart_puts übergibt jeweils ein Zeichen der Zeichenkette an eine Funktion uart_putc, die abhängig von der vorhandenen Hardware implementiert werden muss. In der Funktion zum Senden eines Zeichens ist darauf zu achten, dass vor dem Senden geprüft wird, ob der UART bereit ist den "Sendeauftrag" entgegenzunehmen.

```
19 #define BAUD 9600 //geforderte Baudrate
20 #define F_CPU 7372800UL//Geschwindigkeit des Quarzes
21
22 #include <avr/io.h> // enhält Namen der verwendeten Register
23 #include <util/setbaud.h> //definiert Baudrate im UART Register
 .                              .
 .                              .
85 /*Senden von Daten
86 --------------------*/
87 int uart_putc(char data)
88 {
89  while (!(UCSR0A & (1<<UDRE0))) // warten bis Senden möglich
90  ;
```

Abbildung 4.7 : Senden einer Zeichenkette Teil A

```
 91   UDR0 = data; // sende Zeichen
 92   return 0;
 93  }
 94
 95  /* puts ist unabhängig vom Controllertyp */
 96  void uart_puts (char *s)
 97  {
 98   while (*s)
 99   {/* so lange *s != '\0' also ungleich dem
100    "String-Endezeichen(Terminator)" */
101    uart_putc(*s);
102    s++;
103  }
104  }
   .                              .
   .                              .
```

Abbildung 4.8 : Senden einer Zeichenkette Teil B

Im Hauptprogramm wird einfach das Unterprogramm uart_puts implementiert. Soll zum Beispiel eine Stringvariable, z.B. mit dem Inhalt „Hallo Welt“ ausgegeben werden, wird mit Hilfe des Zeigers „*s“ jedes einzelne Zeichen im String abgetastet und separat ausgegeben. Dies erfolgt so lange, bis das Zeichen „\0“ erscheint, welches das Ende einer Zeichenkette signalisiert.

4.3.4 Empfangen mit dem UART

Empfangen einzelner Zeichen

Beim Empfangen einzelner Zeichen verhält es sich ähnlich wie bei dem Senden einzelner Zeichen. Es ist wieder zu prüfen, ob das UART Modul bereit ist, damit das Empfangen von Zeichen gelingen kann. Dabei muss das RXC0 Register auf „1“ gesetzt sein.

```
uint8_t uart_getc(void)
{
        while (!(UCSR0A & (1<<RXC0)))     // warten bis Zeichen verfügbar
        ;
        return UDR0;                // Zeichen aus UDR0 an Aufrufer zurückgeben
}
```

Dieses Unterprogramm wird wiederum im Hauptprogramm eingebunden.

Empfang von Zeichenketten

Beim Empfang von Zeichenketten, muss man sich zunächst darüber im Klaren sein, dass es ein Kriterium geben muss, an dem der Mikrocontroller erkennen kann, wann ein Text zu Ende ist. Bei der Brandmeldeanlage wird dies durch ein *Carriage Return* (CR) und ein *Line Feed* (LF) gekennzeichnet. Dem Mikrocontroller muss also in der Programmierung deutlich gemacht werden, dass das Carriage Return oder das Line Feed, das Ende eines Strings verdeutlichen. Das Problem der Übertragung eines Strings reduziert sich damit auf

die Aufgabenstellung: Empfange und sammle Zeichen in einem char Array, bis entweder das Array voll ist oder z.B. das Textzeichen Line feed empfangen wurde. Danach wird der empfangene Text noch mit einem '\0' Zeichen abgeschlossen um einen Standard C-String daraus zu machen, mit dem dann weiter gearbeitet werden kann.

```
.                                                .
.                                                .
#define BAUD 9600 //geforderte Baudrate
#define F_CPU 7372800UL//Geschwindigkeit des Quarzes

#include <avr/io.h> // enthält Namen der verwendeten Register
#include <util/setbaud.h> //definiert Baudrate im UART Register
.                                                .
.                                                .
/*Empfangen von Daten
----------------------*/
uint8_t uart_getc(void)
{
 // warten bis Zeichen verfügbar
 while (!(UCSR0A & (1<<RXC0)))
 ;
 return UDR0; // Zeichen aus UDR an Aufrufer zurückgeben
 }

void uart_gets( char* Buffer, uint8_t MaxLen )
{
 uint8_t NextChar;
 uint8_t StringLen = 0;

 NextChar = uart_getc(); // Warte auf und empfange das nächste Zeichen

 // Sammle solange Zeichen, bis:
 // * entweder das String Ende Zeichen kam
 // * oder das aufnehmende Array voll ist
 while( NextChar != '\n' && StringLen < MaxLen - 1 )
 {
  *Buffer++ = NextChar;
  StringLen++;
  NextChar = uart_getc();
 }
 // Noch ein '\0' anhängen um einen Standard
 // C-String daraus zu machen
 *Buffer = '\0';
}
.                                                .
.                                                .
```

Abbildung 4.9 : Empfangen von Zeichenketten mit dem UART

Beim Aufruf ist darauf zu achten, dass das empfangende Array auch mit einer ausreichend großen Größe definiert wird. Dabei ist eine Zeichenlänge von 100 Zeichen völlig ausreichend.

```
char zeichenkette[100];
uart_gets(zeichenkette, sizeof(zeichenkette));
```

Im Hauptprogramm muss dann die Zeichenkette und die dazugehörige Länge an das Unterprogramm uart_gets übertragen werden.

4.3.5 Umsetzung im Hauptprogramm

Wie bereits näher erläutert wurde, müssen die jeweiligen Unterprogramme zur Initialisierung und zum Senden bzw. Empfangen von Zeichenketten im Hauptprogramm implementiert werden.

```
int main (void)
{
char zeichenkette[100];

uart_init(); // Initialisierung
uart_gets(zeichenkette, sizeof(zeichenkette)); // Empfangen der eingehenden
Textinformationen, die in der „zeichenkette“ gespeichert werden

// hier könnte man die zeichenkette noch bearbeiten und dann erst senden
uart_puts(zeichenkette); // ausgeben der Variable „zeichenkette“
}
```

Diese Programmierung ermöglicht das einfache Aufnehmen von Zeichenketten und das darauffolgende Senden dieser.

4.4 Suchablauf der enthaltenen Statusmeldungen

Nachdem der Mikrocontroller erfolgreich Textnachrichten Empfangen und Senden kann, muss dieser entsprechend der Aufgabenstellung die gesuchten Statusmeldungen und das Wort „Gruppe“ filtern.
Für die Bearbeitung von Strings stellt C eine Reihe von Bibliotheksfunktionen zur Verfügung. Diese sind in der Bibliothek *<string.h>* integriert. So ist es auch möglich das erste Vorkommen einer Zeichenfolge s2 in einer anderen Zeichenfolge s1 zu suchen. Als Ergebnis wird ein Zeiger auf die gefundene Zeichenfolge (innerhalb s1) geliefert bzw. der NULL-Zeiger, falls die Suche erfolglos war. Ist die Zeichenfolge s2 Null, so wird der Zeiger auf s1 geliefert. In einem Beispiel sieht dies wie folgt aus:

```
char* suchErg; // Zeiger Definition
char s1[]={"Ein Mikrocontroller ist eine tolle Erfindung"}
char s2[]={"ist"}
suchErg=strstr(s1,s2);
```

Der Zeiger „suchErg“ enthält nach dem Suchvorgang „ist eine tolle Erfindung“. Es wird also nicht nur das entsprechende Wort bzw. die Zeichenkette gesucht und in einen Zeiger abgelegt, sondern auch alle darauffolgenden Zeichen.

Um auf das Suchen der Wörter zurück zukommen, soll am nachfolgenden Beispiel der Ablauf der Filterung genauer erläutert werden.

```
1  65437: Fre 18-Jul-14 00:03:40:   * FEUERALARM *\r\n
2  R1 Mod.   4 - Gruppe 0004/01 - DKM \r\n
3  Grup.:Mitte\r\n
4  Ort: Handmelder Messw.:114%\r\n
```

Abbildung 4.10 : Stringbeispiel zur Erläuterung des Suchverlaufes

Es soll zunächst jede empfangene Zeichenkette in die Variable „*zeichenkette*“ gespeichert werden. Diese Variable wird daraufhin durch die Funktion *strstr()* durchsucht. Wenn, in diesem Beispiel das Wort „FEUERALARM“ in der Zeichenkette enthalten ist, wird ein Zeiger (suchErg) auf die gefundene Zeichenfolge geliefert. Der Inhalt des Zeigers wäre in diesem Fall dann „FEUERALARM *“. Anschließend kann man zum Beispiel den Zeiger mit Hilfe der Funktion *strcpy()*, die ebenfalls in der Bibliothek *<string.h>* enthalten ist, in einer weiteren Zeichenkette (in dem Fall „elkatxt“) abspeichern. Dieser String kann dann für weitere Anwendungen genutzt werden, z.B. wird dies für das spätere Senden der gefilterten Textinformationen benötigt (siehe Kapitel 4.6).

```
28  #define laenge 200 //Längenangabe der Zeichenketten
29  #define laenge2 200
30  #define laengeelka 200
 .                              .
 .                              .
117  int ZZ_logik,Gr_logik,zsp=0;
 .                              .
 .                              .
127   char zeichenkette[laenge];
128   // Definitionen Gruppennummer
129   char suchGr[]={"Gruppe"}, suchFA[]={"FEUERALARM"};
130   char suchVA[]={"VORALARM"}, suchEF[]={"ELEMENT FEHLT"};
131   char suchwuA[]={"wieder unter ALARMWERT"};
132   char suchwv[]={"Wieder Vorhanden"}, suchZZ[]={"CKSETZEN"};
133   char zeichenkette2[laenge2]="\0"; //muss als erstes leer sein
134   char* suchErg;
 .                              .
 .                              .
153   //Suche Wort 'FEUERALARM' in der Zeichenkette
154   suchErg=strstr(zeichenkette,suchFA);
155   //schreibe Zeiger in Speicher von elkatxt
156   if(suchErg!=0)
157   {strcpy(elkatxt,suchErg);
158    endtxt=10; // 10 entspricht die Länge des Wortes Feueralarm
159    suchErg=0; // muss auf "0", wegen nächsten Durchlauf des Programms
160    zsp = 1; //signalisiert, dass ein Wort abgespeichert ist
161   }
```

Abbildung 4.11 : Definitionen der gesuchten Zeichenketten und Suchablauf einer Statusmeldung

Damit im späteren Verlauf Bedingungen definiert werden können, müssen zusätzlich gewisse Logiken verwendet werden. Diese Logiken sollen zeigen, ob ein Ereignis eingetroffen ist oder nicht. Im Fall der Statusmeldungen kennzeichnet der Zwischenspeicher, dass eine Statusmeldung aufgetreten ist. Daher soll der Zwischenspeicher eine gewisse Logik erhalten, die bei Eintreffen einer Statusmeldung „1“ gesetzt wird. Diese Zwischenspeicher-Logik wird im Programm als „zsp“ definiert. Des Weiteren müssen auch Logiken für das Auftreten einer Meldernummer und der Statusmeldung „ZENTRALE ZURÜCKSETZEN“ definiert werden, da diese ebenfalls Bedingungsparameter sind. Die Logik der Statusmeldung „ZENTRALE ZURÜCKSETZEN“ wird als „ZZ_logik“ und die Logik der aufgetretenen Meldernummer als „Gr_logik“ bezeichnet.

Um auf das Filtern der gesuchten Wörter zurück zu kommen, ist der Suchmechanismus bei allen Meldungen gleich. Lediglich das Abspeichern der aufgetretenen Zeichenketten unterscheidet sich hierbei ein wenig. Bei allen Statusmeldungen, außer „ZENTRALE ZURÜCKSETZEN“, wird der Inhalt, auf den der Zeiger „suchErg“ zeigt, direkt in die Zeichenkette „elkatxt“ kopiert (z.B. „FEUERALARM *“). Das hat die Ursache, dass dieser String im späteren Verlauf ebenfalls dafür genutzt werden soll, um die eintreffende Meldernummer darin abzuspeichern. Deshalb spielt es keine Rolle, dass noch weitere Zeichen nach der enthaltenen Statusmeldung mit abgespeichert werden. Bei Eintreffen der Nachricht, dass die Zentrale zurückgesetzt wurde, ist es nicht notwendig, zusätzlich eine Meldernummer in den String „elkatxt“ abzuspeichern. Daher muss realisiert werden, dass nur der gewünschte Text, der zum RS232/KNX Gateway in „elkatxt“ gesendet wird, enthalten ist. Ein Problem dabei besteht jedoch, dass der Mikrocontroller keine Sonder-ASCII-Zeichen, wie einem „ü“, „ö“ oder „ä“, aufnehmen und senden kann. Daher wird beim Suchen des Textstring „ZENTRALE ZURÜCKSETZEN“ nur ein Teil des Textinhaltes gesucht (in diesem Fall „CKSETZEN“). Nach Auftreten dieser Zeichenkette wird daraufhin der Textstring „ZENTRALE ZURUECKSETZEN“ in die Variable „elkatxt“ geschrieben. Dadurch wird das Senden eines „Ü“ vermieden und der Textinhalt ist dennoch derselbe (siehe Abbildung 4.12, Zeile 211).

```
  .                                .
207  //Suche Wort 'CKSETZEN' in der Zeichenkette
208  suchErg=strstr(zeichenkette,suchZZ);
209  //schreibe Zeiger in Speicher von elkatxt
210  if(suchErg!=0)
211  {suchErg="ZENTRALE ZURUECKSETZEN";
212   strcpy(elkatxt,suchErg);
213   endtxt=22; // 22 entspricht "ZENTRALE ZURUECKSETZEN"
214   suchErg=0; // muss auf "0", wegen nächsten Durchlauf des Programms
215   ZZ_logik=1; // Logik erfüllt, da Wort enthalten
216  }//Zwischenspeicher nicht erforderlich,
217   //da keine Gruppe mit ausgegeben wird
```

Abbildung 4.12 : Suchablauf der Statusmeldung "ZENTRALE ZURÜCKSETZEN"

Bei Eintreffen des Wortes „Gruppe“ verhält es sich ähnlich zu der Statusmeldung „FEUERALARM“. Der Unterschied besteht dabei, dass man bei dem Abspeichern der Zeichenkette nicht „elkatxt“, sondern die char-Variable „zeichenkette2“ verwendet, da diese für die Filterung der Meldernummer benötigt wird.

```
220   //Suche Wort 'Gruppe' in der Zeichenkette
221   suchErg=strstr(zeichenkette,suchGr);
222   //schreibe Zeiger in Speicher von "zeichenkette2"
223   if(suchErg!=0)
224   {strcpy(zeichenkette2,suchErg);
225    // muss abgespeichert werden, um damit weiterzuarbeiten
226    //damit die Meldernummer ermittelt werden kann
227   }
```

Abbildung 4.13 : Suchablauf der Zeichenkette "Gruppe"

Es sei zusätzlich darauf hingewiesen, dass die Variable „endtxt“ die Länge der in „elkatxt“ enthaltenen Statusmeldung angibt. Dies muss definiert werden, um später die gefilterte Meldernummer in die Zeichenkette „elkatxt“ schreiben zu können (siehe Abschnitt 4.6). Zusätzlich sei angemerkt, dass der Zwischenspeicher „zsp“ außerhalb der while-Schleife, welche so zu sagen das Hauptprogramm enthält, mit dem Wert „0“ definiert werden muss, damit beim ersten Durchlauf des Programms der Zwischenspeicher mit einer Logik definiert ist, und somit beim Vergleichen der Bedingungen kein Fehler auftreten kann.

4.5 Suchablauf der Meldernummer

Die Suche der erwähnten Zeichenketten (FEUERALARM, VORALARM, etc.) wurde im vorherigen Kapitel bereits näher erläutert. Nun stellt sich die Frage, in wie weit die Nummern der Melder aus den empfangenen Zeichenketten gefiltert werden. Da man in der Brandmeldeanlage die Meldernummern frei programmieren kann, sind diese also nie konstant. Des Weiteren muss man darauf achten, dass eine Brandmeldeanlage eine sehr große Anzahl an Teilnehmer beinhalten kann. Somit wäre die Verwendung der Funktion *strstr()* auszuschließen, um die genauen Nummern der Melder in der Zeichenkette einzeln zu suchen, da diese einen enormen Programmierungsaufwand hervorrufen würden. Wie bereits erwähnt besteht allerdings die Möglichkeit, durch die Funktion *strstr()* das Wort bzw. die Zeichenkette „Gruppe“ und die darauffolgenden Zeichen zu suchen und diese in die char-Variable „zeichenkette2“ zu speichern. Da die gespeicherte Variable dann die gesuchte Nummer eines Melders beinhaltet, kann die Suche schon ein wenig eingegrenzt werden.

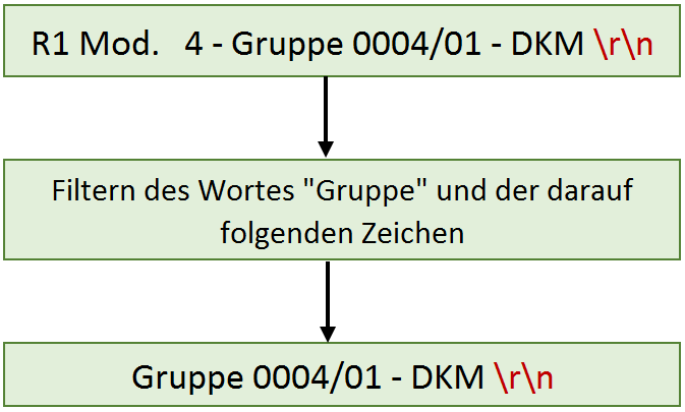

Abbildung 4.14 : Filterung der Zeichenkette "Gruppe"

Nun ist leicht zu erkennen, dass die Nummer eines Melders durch zwei Leerzeichen eingeschlossen ist. Es muss lediglich herausgefunden werden, an welcher Stelle sich das erste Leerzeichen vor bzw. das zweite Leerzeichen nach der gesuchten Nummer befindet.

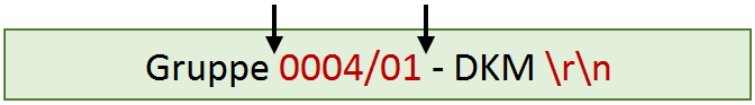

Abbildung 4.15 : Filterung der Meldernummer

Dies kann man durch die Verwendung einer einfachen For-Schleife realisieren, in dem nach und nach jede Stelle der Zeichenkette abgetastet wird, bis man die einzelnen Stellen der Leerzeichen ermittelt hat. Das Abtasten erfolgt mit Hilfe eines Zeigers, der jede Stelle der Zeichenkette abtastet, bis das erste Leerzeichen gefunden wurde. Das eigentliche Suchen des Leerzeichens erfolgt dabei jedoch durch eine If-Anweisung, die jedes durchsuchte Zeichen mit einem Leerzeichen vergleicht (siehe Abbildung 4.16, Zeile 235). Wenn das durchsuchte Zeichen ein Leerzeichen ist, wird der entsprechende Index der Zeichenkette in die Variable „leerz1“ gesichert (siehe Abbildung 4.16, Zeile 239).

```
136     int leerz1=0;
137     int leerz2=0;
  .                                 .
  .                                 .
232     //soll nur durchlaufen werden, wenn Zwischenspeicher voll ist
233     if(zsp==1)
234     {
235      for(i=0;*(zeichenkette2+i)!=' ';i++)
236       {//gibt an, an welcher Stelle das erste "Leerzeichen" ist
237        if(*(zeichenkette2+(i+2))==' ')
238        // +2, Stelle nach " " wird benötigt
239        leerz1=i+2;
240        else
241        if(i==laenge2-1)
242        {Gr_logik=0;break;} //Vermeidung eines Überlaufes,
243         //falls kein Leerzeichen vorhanden sein sollte
244         // und Logik 0, da keine Meledernummer vorhanden
245       }
```

Abbildung 4.16 : Suchablauf des ersten Leerzeichens

Falls es sein sollte, dass der Textstring „Gruppe“ gefunden wurde und die darauffolgenden Zeichen kein Leerzeichen aufweisen, wurde hierbei eine Sicherung eingebunden (siehe Abbildung 4.16, Zeile 242). Dies verhindert somit einen möglichen Überlauf, dass in der Praxis jedoch wohl kaum passieren dürfte, da die Brandmeldeanlage eine definierte Textprogrammierung besitzt und das Wort „Gruppe“ niemals ohne eine Meldernummer vorkommen wird.
Zusätzlich würde bei diesen theoretisch möglichen Überlauf die Logik der auftretenden Meldernummer auf „0“ gesetzt werden, da keine erfolgreiche Filterung der Meldernummer zustande kam. Nachdem das erste Leerzeichen ermittelt wurde, muss schlussfolgernd die Zeichenkette noch nach dem zweiten Leerzeichen durchsucht werden, indem eine weitere If-Anweisung verwendet wird. Diese If-Anweisung vergleicht zunächst den Variableninhalt von „leerz1“. Dieser wurde bei der Definition mit „0“ gesetzt, d.h. die If-Anweisung vergleicht den Inhalt der Variable um herauszufinden, ob diese im Programmverlauf neu definiert wurde oder nicht (die Variable wird mit „1“ verglichen und nicht mit „0“, da in Abbildung 4.17 (Zeile 247) der Wert um eins erhöht werden musste). Ist die Bedingung erfüllt, erfolgt das Durchsuchen der Zeichenkette nach dem zweiten Leerzeichen, nach dem gleichen Schema wie beim ersten, mit Ausnahme, dass der ermittelte Wert in die Variable „leerz2“ gesichert wird.

```
246     //Stelle nach dem ersten Zeichen
247     leerz1=leerz1+1;
248     if(leerz1>1)
249     {
250      for(i=leerz1;*(zeichenkette2+i)!=' ';i++)
251      {if(*(zeichenkette2+(i+2))==' ')
252       {//Stelle von 2.Leerzeichen
253        leerz2=i+1;
254        Gr_logik=1; //erst dann ist Gruppennachricht enthalten
255       }
256       else
257       if(i==laenge2-1)
258       {Gr_logik=0;break;} //Vermeidung eines Überlaufes,
259       // falls kein Leerzeichen vorhanden sein sollte
260       // und Logik 0, da keine Meledernummer vorhanden
261      }
262     }
263   }
    .                                .
    .                                .
```

Abbildung 4.17 : Suchablauf des zweiten Leerzeichens

Da man nun weiß, an welchen Stellen die Meldernummer durch die beiden Leerzeichen eingegrenzt ist, kann diese daraus leicht ermittelt werden. Da die Nummer in der Variable „Zeichenkette2“ bereits gespeichert ist und man den Speicherort in der Zeichenkette, durch die Eingrenzung der beiden Leerzeichen weiß, muss diese nicht noch einmal separat abgespeichert werden. Schlussendlich wurde die Logik der Meldernummer auf „1“ gesetzt, da die Ermittlung der Meldernummer erfolgreich von statten ging (siehe

Abbildung 4.17, Zeile 254). Wie man nun die bereits ermittelte Statusmeldung und die dazu gehörige Meldernummer verbindet und anschließend an das RS23/KNX-Gateway sendet, soll in den nachfolgenden Kapitel näher erläutert werden.

4.6 Ausgabe der gefilterten Textinformationen

Wie bereits in Kapitel 2.3 näher erläutert wurde, soll die ermittelte Statusmeldung und die dazu gehörige Meldernummer zusammen ausgegeben werden, damit das RS232/KNX-Gateway diesen String mit einer entsprechenden Gruppenadresse verknüpfen kann. Im Kapitel 4.3.3 wurde das Senden von Zeichenketten schon näher erläutert. Es muss also nur noch realisiert werden, dass die gefundene Statusmeldung und die dazugehörige Meldernummer in einen gemeinsamen String gespeichert werden, damit dieser dann zum RS232/KNX Gateway versendet werden kann.
Fassen wir noch einmal zusammen:

- Die gefilterte Statusmeldung ist nun in der char-Variable „elkatxt“ abgespeichert
- Die Meldernummer wurde in der Zeichenkette „zeichenkette2“ gesichert und durch die eingeschlossenen Leerzeichen ermittelt (die Indizes in der char-Variable „zeichenkette2“)

Wie in Abbildung 4.18 zu sehen ist, muss nun jede einzelne Zahl der Meldernummer, in den richtigen Bereich der char-Variable „elkatxt“ geschrieben werden.

char-Variable	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
elkatxt	F	E	U	E	R	A	L	A	R	M		*	\r	\0									
zeichenkette2	G	r	u	p	p	e		0	0	0	4	/	0	1		-		D	K	M	\r	\0	
elkatxt =	F	E	U	E	R	A	L	A	R	M		0	0	0	4	/	0	1					

Abbildung 4.18 : Verknüpfung der Meldernummer mit der Ausgabe-Variable

Nachdem im Programm die Bedingung der Ausgabe gesetzt wurde, soll zunächst sichergestellt sein, dass nach der Statusmeldung auch ein Leerzeichen erfolgt, um die Überschaubarkeit des auszugebenen Textes sicherzustellen. Durch die Variable „endtxt“ wurde bereits die Stelle nach der Statusmeldung festgelegt. Dadurch kann diese Stelle ohne weiteres mit einem Leerzeichen belegt werden (siehe Abbildung 4.19, Zeile 273).

```
268   //Bedingung muss efüllt sein
269   if((Gr_logik==1 && zsp==1)||ZZ_logik==1)
270   {
271    if(Gr_logik==1 && zsp==1)
272    {
273        elkatxt[endtxt]=' '; //sorgt für Leerzeichen nach der Statusmeldung
274        endtxt=endtxt+1; //bei "ZENTRALE ZURUECKSETZEN" kein “ “ gefordert
```

Abbildung 4.19 : Setzen der Bedingung und eines Leerzeichens

Im nächsten Schritt wird jede Zahl der Meldernummer einzeln in die jeweilige Stelle der char-Variable „elkatxt“ geschrieben. Dafür muss vorher jedoch der Wert von „endtxt“ um eins erhöht werden, um die erste Stelle nach dem Leerzeichen erhalten zu können. Beim Übertragen der Zahlen wird dabei eine For-Schleife verwendet, die jede einzelne Zahl zwischen „leerz1“ und „leerz2“ abtastet und daraufhin in die Zeichenkette „elkatxt“ überträgt (siehe Abbildung 2.20, Zeile 278). Schlussendlich müssen nur noch ein Carriage Return und ein „\0“-Zeichen an den String gehangen werden. Daraufhin wird der String ausgegeben und die Logik des Zwischenspeichers wieder auf „0“ gesetzt. Nach Ausgabe der Textnachricht beginnt der Mikrocontroller wieder von neuen auf eingehende Informationsmeldungen zu warten.

```
275     //Speichert die Meldernummer in elkatxt
276     for(i=leerz1;i<=leerz2;i++)
277     {
278      elkatxt[endtxt]=*(zeichenkette2+i);
279      endtxt=endtxt+1;
280     }
281    }
282    //An Elkatxt soll ein \r gehangen werden
283    elkatxt[endtxt]='\r';
284    endtxt=endtxt+1;
285    //und eine "\0" gehangen werden
286    elkatxt[endtxt]='\0';
287
288    uart_puts( elkatxt );
289    zsp = 0; // Da Ausgabe erfolgte, kann Speicher "gelöscht werden"
290
291   }
   .                          .
   .                          .
```

Abbildung 4.20 : Ausgabe aller gefilterten Texte

4.7 Programmierung mit Atmel Studio 6.2

In diesem Kapitel soll es darum gehen, wie das entwickelte C-Programm, mit Hilfe von Atmel Studio, auf den Mikrocontroller geladen wird und wie die Anbindung des externen Oszillator (Frequenz von 7.3728MHz) realisiert werden kann. Wie bereits erwähnt wurde, ist Atmel Studio eine Software zur Programmierung von AVR Mikrocontrollern, die kostenlos zur Verfügung gestellt wird. Im nachfolgenden wird geschildert, welche Schritte zur Programmierung des Mikrocontrollers nötig sind.

4.7.1 Erstellung eines Projektes

Zu allererst muss im Atmel Studio ein neues Projekt angelegt werden. Dabei beschreibt man die zu verwendende Programmiersprache und den entsprechenden Mikrocontroller.

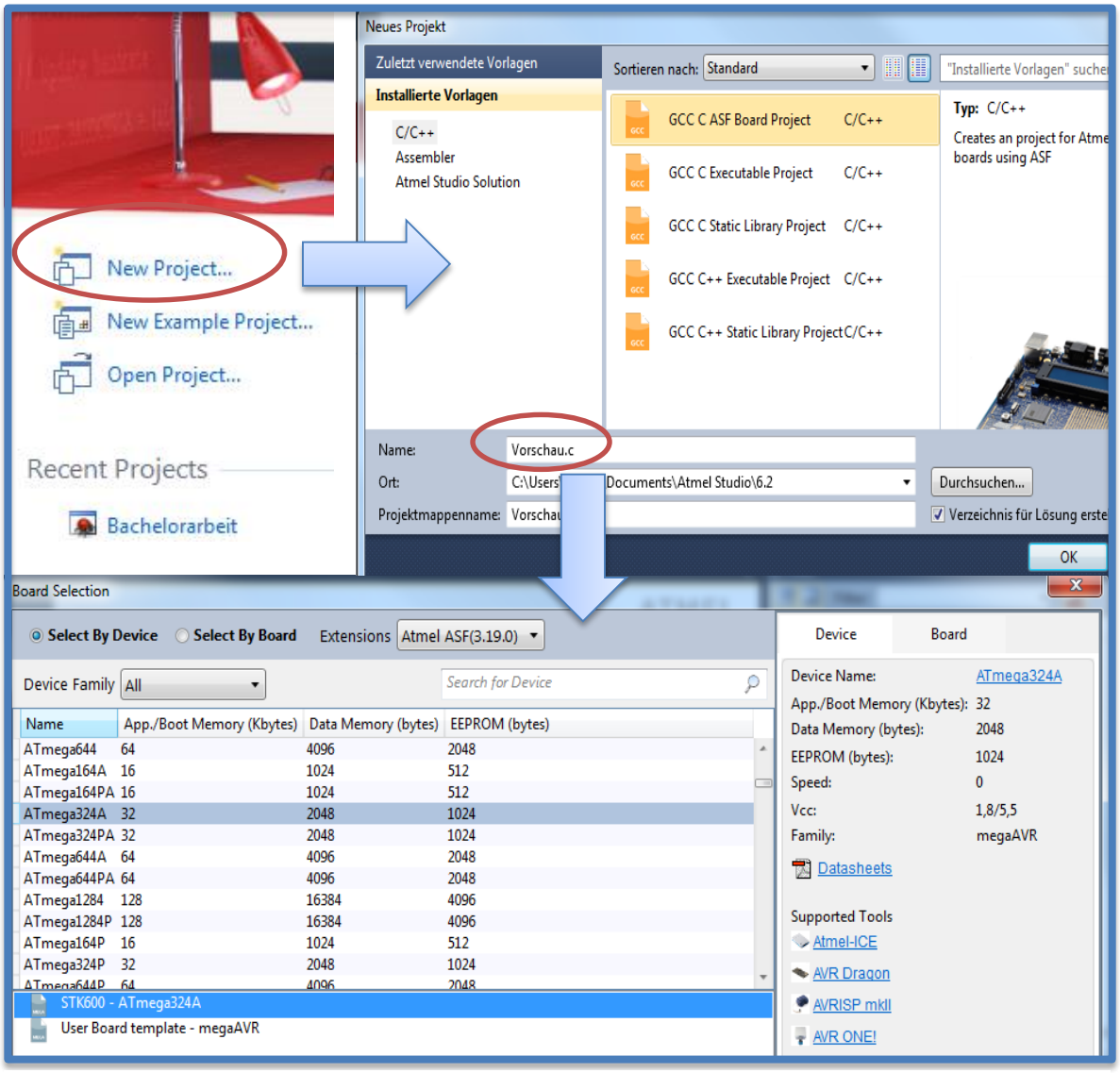

Abbildung 4.21 : Erstellung eines neuen Projektes

Damit das Atmel Studio weiß, ob in diesem Fall C oder C++ verwendet werden soll, muss darauf geachtet werden, dass man bei der Verwendung von C, den Dateinamen mit einer Endung „.c“ versehen muss. Nach erfolgreicher Konfigurierung öffnet sich ein Fenster, in dem man die eigentliche Programmierung realisieren kann. Bevor man jedoch das C-Programm auf den Mikrocontroller bringt, muss zuerst der externe Oszillator implementiert werden.

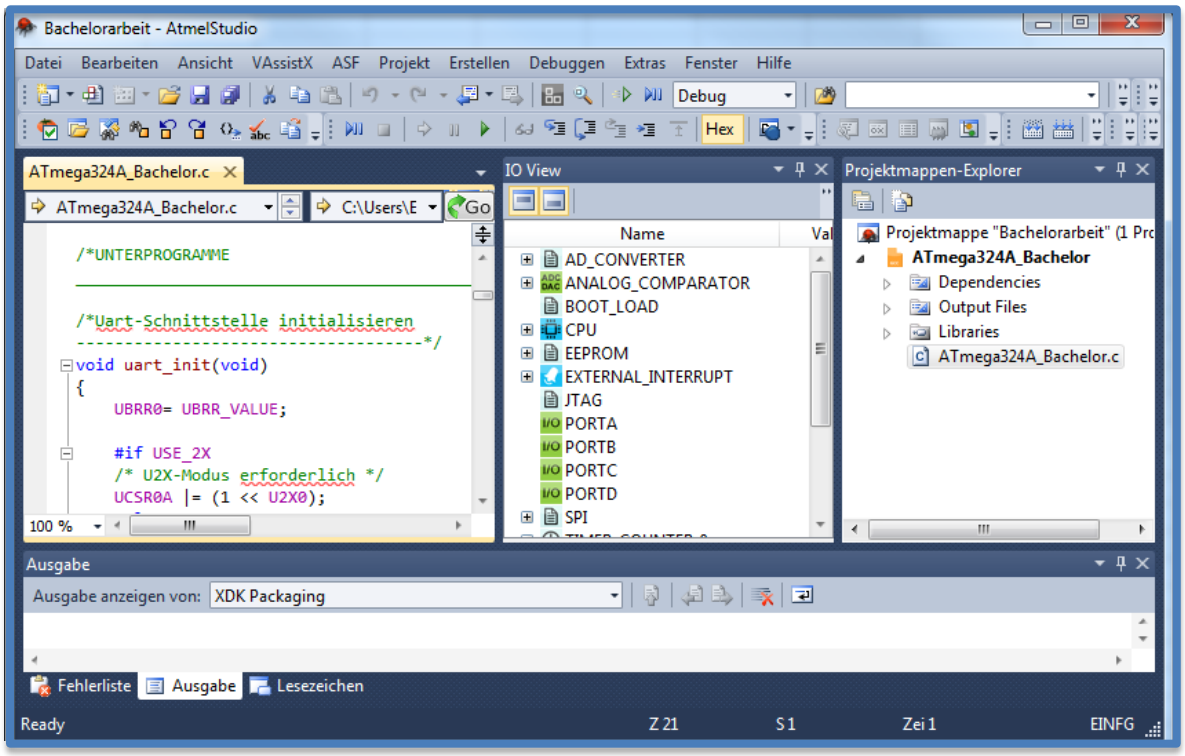

Abbildung 4.22 : Programmübersicht des Atmel Studios 6.2

4.7.2 Programmierung des externen Oszillators

Die Programmierung des externen Oszillators (Schwingquarz) erfolgt ebenfalls durch das Atmel Studio. Dabei müssen sogenannte Fuse-Bits des Mikrocontrollers, über die ISP-Schnittstelle des aTeVaL Evaluationsboardes programmiert werden. Fuse kommt aus dem englischen und bedeutet übersetzt Sicherung oder Schmelzsicherung. Die Bits dienen zur Manipulation der Chip-Einstellungen, man sollte sie daher nur anfassen, wenn man wirklich weiß was man tut.
Zunächst soll näher auf die Funktion dieser Bits und deren Handhabung eingegangen werden. Der ATmega324A verfügt über 24 Fuse-Bits – also 3 Fuse-Bytes. Es ist darauf zu achten, dass aktivierte Fuse-Bits den Wert 0 haben, und deaktivierte den Wert 1.

Tabelle 4.8 : Fuse High Byte

Fuse High Byte	Bit	Standard-Einstellung
OCDEN	7	1 (unprogrammiert, OCD ausgeschaltet)
JTAGEN	6	0 (programmiert, JTAG angeschaltet)
SPIEN	5	0 (programmiert, SPI Programmierung an)
WDTON	4	1 (unprogrammiert)
EESAVE	3	1 (unprogrammiert)
BOOTSZ1	2	0 (programmiert)
BOOTSZ0	1	0 (programmiert)
BOOTRST	0	1 (unprogrammiert)

OCDEN: On Chip Debugging aktivieren

In der Voreinstellung ist dieses Fusebit nicht gesetzt, also 1. Es muss aktiviert werden, wenn über die JTAG Schnittstelle debugged werden soll

JTAGEN: JTAG aktiviert

In der Voreinstellung ist dieses Fusebit gesetzt, also 0.

Die JTAG-Schnittstelle belegt 4 Bits des Mikrocontrollers (TDI/PC5, TDO/PC4, TMS/PC3, TCK/PC2). Benötigt man keine JTAG-Schnittstelle, da man zum Beispiel den Mikrocontroller nicht debuggen will, deaktiviert man JTAGEN und hat dann am Chip 4 zusätzliche I/O-Pins zur Verfügung.

SPIEN: Programmierung über SPI aktiviert

In der Voreinstellung ist dieses Fusebit gesetzt. Wird dieses Fusebit auf 1 gesetzt (deaktiviert), dann lässt sich der Chip nicht mehr über den ISP programmieren sondern nur noch über einen Adapter der den Mikrocontroller nicht seriell sondern parallel programmiert (z.B. ATMEL STK500). Dieses Bit kann unter AVR Studio nicht verändert werden, wenn man den Chip seriell mit einem ISP programmiert.

CKOPT: Verstärkung der Oszillator-Frequenz

Wird der Quarz nicht direkt neben dem Mikrocontroller platziert sondern etwas weiter entfernt, dann kann es sein, dass das Signal der Frequenz zu schwach am Chip ankommt. Tritt dieser Fall auf, kann man das Signal über dieses Fusebit verstärken, wodurch allerdings der Stromverbrauch erhöht wird.

EESAVE: EEPROM wird beim Chip-Erase nicht beeinflusst

In der Voreinstellung ist dieses Fuse-Bit nicht gesetzt. Das heißt, dass bei jedem Chip-Erase das EEPROM auch mit gelöscht wird. Hat man Daten darin gespeichert, die auch über einen Chip-Erase hinweg erhalten werden sollen, oder greift man gar nicht auf das EEPROM zu, sollte dieses Bit gesetzt ("0") werden.

BOOTSZ1 und BOOTSZ0: Größe des Bootloader-Bereichs

Die Größe des Bootloaderbereichs liegt zwischen 128 und 1024 Words und kann über diese beiden Fuse-Bits festgelegt werden. Da in der Voreinstellung diese beiden Bits gesetzt sind, liegt die Größe des Bootloaderbereichs bei 1024 Words (2048 Bytes).

BOOTRST: Auswahl des Resetvektors

In der Voreinstellung ist dieses Fusebit nicht gesetzt. Führt man am Chip einen Reset durch, dann liegt die Wiedereinsprungadresse normalerweise bei 0x00. Wird dieses Fuse-Bit gesetzt, dann startet die Ausführung nicht an dieser Adresse sondern im Bootloader-Bereich.

Tabelle 4.9 : Extended Fuse Byte

Fuse Low Byte	Bit	Standard-Einstellung
-	7	-
-	6	-
-	5	-
-	4	-
-	3	-
BODLEVEL2	2	1 (unprogrammiert)
BODLEVEL1	1	1 (unprogrammiert)
BODLEVEL0	0	1 (unprogrammiert)

BODLEVEL0 bis 2: auslösendes Spannungslevel für den Brown-Out-Detector

In der Voreinstellung sind dieses Fuse-Bits nicht gesetzt. Im Datenblatt kann nachgeschaut werden, welche Einstellungen nötig sind, um einen Reset bei einer bestimmt Spannung auslösen zu lassen. Beispielsweise wird bei der Einstellung „100“ der Reset bei einer Spannung von 4,3 V ausgelöst

Tabelle 4.10 : Fuse Low Byte

Fuse Low Byte	Bit	Standard-Einstellung
CKDIV8	7	0 (programmiert, SPI Programmierung an)
CKOUT	6	1 (unprogrammiert)
SUT1	5	1 (unprogrammiert)
SUT0	4	0 (programmiert, SPI Programmierung an)
CKSEL3	3	0 (programmiert, SPI Programmierung an)
CKSEL2	2	0 (programmiert)
CKSEL1	1	1 (unprogrammiert)
CKSEL0	0	0 (programmiert, SPI Programmierung an)

CKDIV8: Divide clock by 8

Dieses Fuse-Bit ist standardmäßig gesetzt, das bedeutet, dass die Quarzfrequenz intern durch 8 geteilt wird. Da der interne Oszillator 8MHz besitzt, wird eine Frequenz von 1 MHz verwendet (wenn das Bit programmiert ist).

CKOUT: Clock Output

Erlaubt das System, die innere Taktfrequenz an den Ausgang von PORTB1 zu legen.

SUT1 und SUT0: Vorlaufzeit des Mikroprozessors

In der Voreinstellung ist SUT1 nicht gesetzt und SUT0 gesetzt, wodurch eine maximale Vorlaufzeit von 65ms gewährleistet ist.

CKSEL0 bis CKSEL3: Clock Select

Über diese 4 Fuse-Bits wird die Taktquelle bestimmt.

Damit man nun den Quarz mit einer Frequenz von 7,3728 MHz verwenden kann, wird lediglich das Fuse Low Byte angepasst. Dabei muss ein Blick in das Datenblatt des ATmega324A geworfen werden, um herauszufinden, welche Bits verändert werden müssen. Lediglich das CKDIV8 und das CKOUT können vorher bestimmt werden, da diese bei der Anwendung keine Rolle spielen. Somit werden beide Bits „1“ gesetzt.

Zunächst muss dem Mikrocontroller sozusagen mitgeteilt werden, welche externe Taktfrequenz man verwenden möchte. Bevor man dies jedoch tut, muss man sich im Klaren sein, dass der ATmega324A mehrere Optionen besitzt, die Art der externen Quelle anzugeben.

Tabelle 4.11 : Device Clocking Option

Device Clocking Option	CKSEL3..0
Low Power Crystal Oscillator	1111 - 1000
Full Swing Crystal Oscillator	0111 - 0110
Low Frequency Crystal Oscillator	0101 - 0100
Internal 128kHz RC Oscillator	0011
Calibrated Internal RC Oscillator	0010
External Clock	0000
Reserved	0001

Es bestünde dabei die Möglichkeit, die Optionen „Low Power Crystal Oscillator“ (0,4 – 16 MHz) und „Full Swing Crystal Oszillator“ (0,4 – 20 MHz) zu wählen, da 7,3728 MHz in beiden Freuquenzbereichen Platz findet. Rein intuitiv wurde sich hier für die Low Power Crystal Operating Option entschieden. Um nun den Mikrocontroller die externe Taktquelle mitteilen zu können, muss im Datenblatt, unter den Low Power Crystal Operating Einstellungen, die entsprechenden CKSEL-Bits nachgelesen werden.

Tabelle 4.12 : Low Power Crystal Oscillator Operating Modes

Frequency Range [MHz]	CKSEL3..1	Recommended Range for Capacitors C1 and C2 [pF]
0,4 - 0,9	100	-
0,9 - 3,0	101	11 - 22
3,0 - 8,0	110	11 - 22
8,0 - 16,0	111	12 - 22

Daraus folgt, dass für die Einstellung der Bits CKSEL3..1 eine Einstellung mit „110“ vorgenommen werden muss, da sich der verwendete Quarz, mit der Taktfrequenz von 7,3728 MHz, in dem Bereich von 3,0 – 8,0 MHz befindet.
Nun müssen lediglich noch die Bits CKSEL0, SUT1 und SUT0 definiert werden.

Tabelle 4.13 : Start-up Times for the Low Power Crystal Oscillator Clock Selection

Oscillator source / power conditions	Start-up time from power-down and power-save	Additional delay from reset (VCC = 5.0V)	CKSEL0	SUT1..0
….	….	….	….	….
Crystal Oscillator, BOD enabled	16K CK	14CK	1	01
Crystal Oscillator, fast rising power	16K CK	14CK + 4.1ms	1	10
Crystal Oscillator, slowly rising power	16K CK	14CK + 65ms	1	11

Der Mikrocontroller soll wie in den Standard Einstellungen eine maximale Vorlaufzeit (65 ms) erhalten. Daher wird die Auswahl „Crystal Oscillator, slowly rising power“ getroffen. Schlussfolgernd muss das CKSEL0-Bit und die beiden SUT-Bits „1“ gesetzt werden.
Zusammenfassend werden noch einmal alle Änderungen des Fuse Low Bytes in Tablle 4.14 zusammengetragen.

Tabelle 4.14 : Fuse Low Byte Veränderungen

	7	6	5	4	3	2	1	0
	CKDIV8	**CKOUT**	**SUT1**	**SUT0**	**CKSEL3**	**CKSEL2**	**CKSEL1**	**CKSEL0**
Standard	0	1	1	0	0	0	1	0
Änderung	1	1	1	1	1	1	0	1

Nun können diese Änderungen mit Hilfe von Atmel Studio zum Mikrocontroller übertragen werden. Dabei bindet man zunächst, unter dem Punkt „Device Programming“, das aTeVal Evaluationsboard ein.

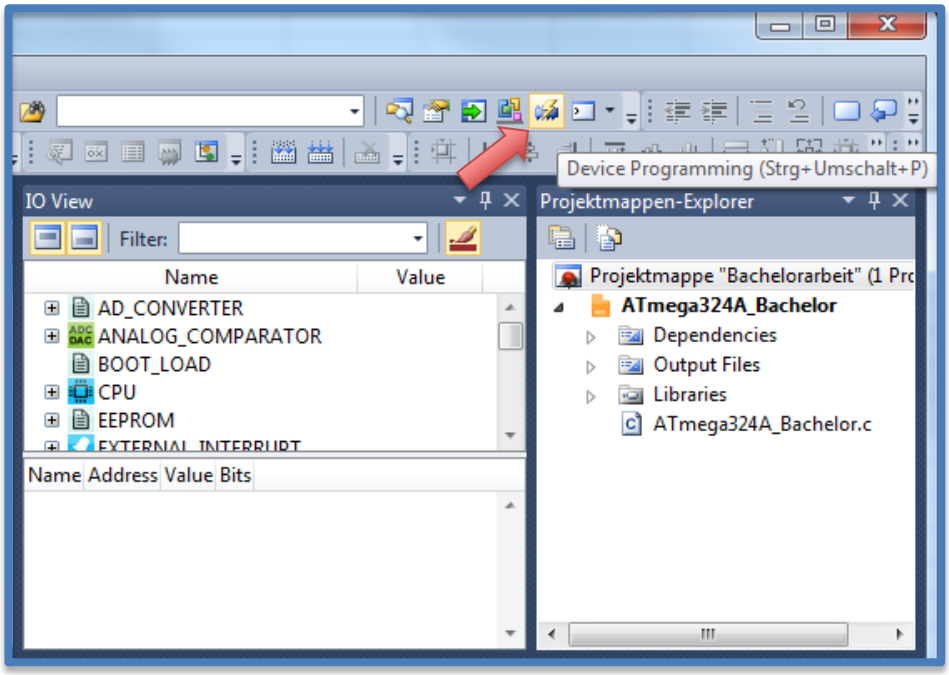

Abbildung 4.23 : Einbindung des aTeVaL Evaluationsboardes

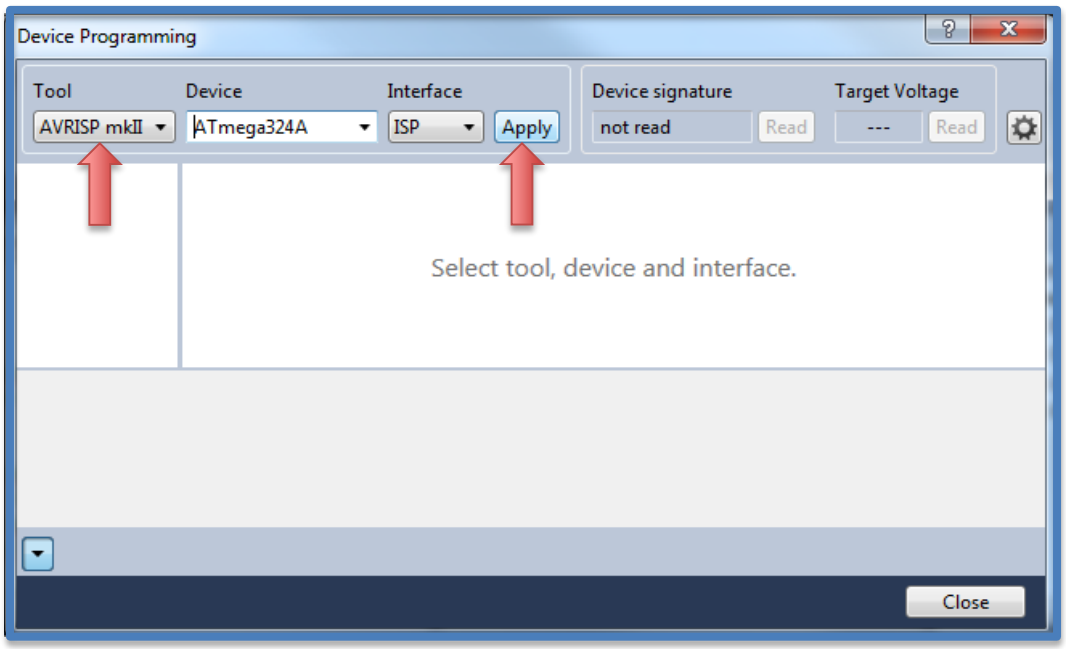

Abbildung 4.24 : Auswahl des AVRISP mkII Clones

Anschließend müssen unter dem Reiter „Fuses“ die Einstellungen des Fuse-Low-Bytes geändert und auf den Controller programmiert werden.

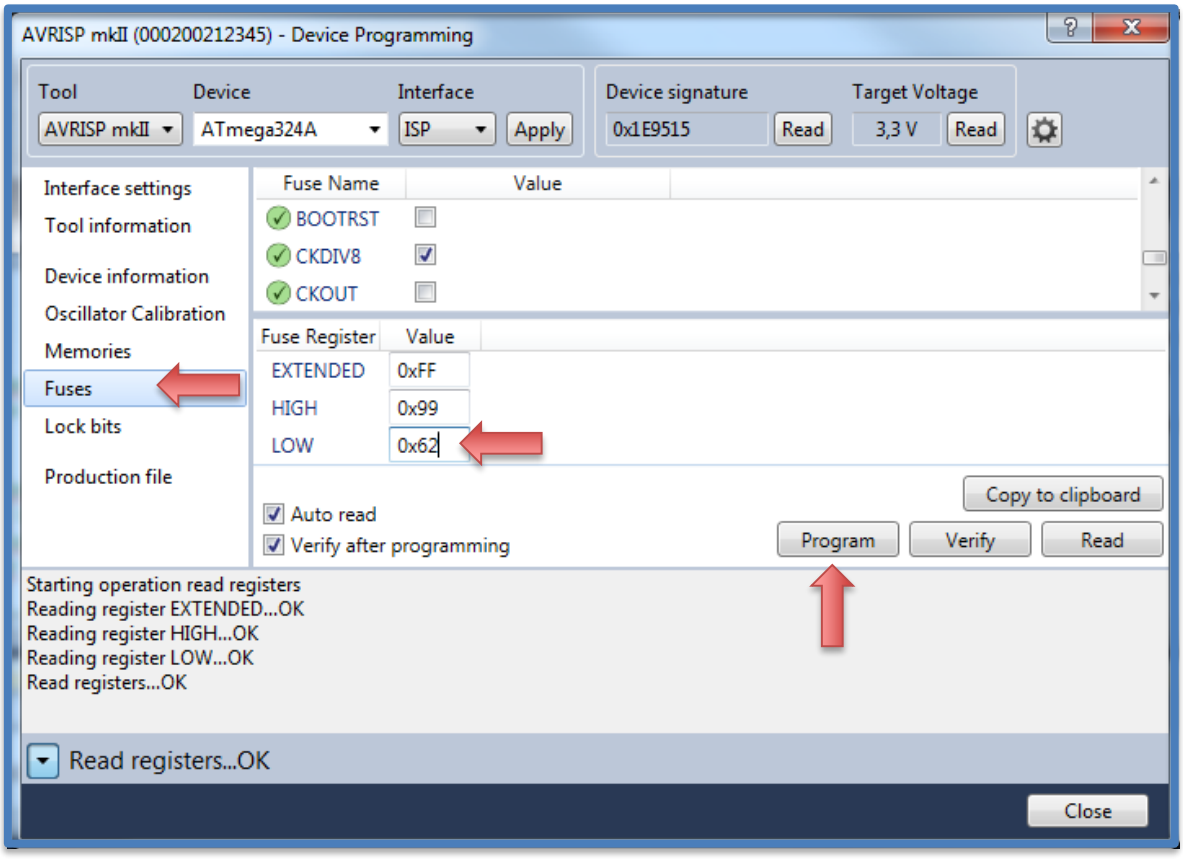

Abbildung 4.25 : Setzen des Fuse Low Bytes

4.7.3 Übertragung des Quellcodes auf den AVR

Nach erfolgreicher Programmierung der Fuse-Bits, muss nur noch das C-Programm auf den Mikrocontroller übertragen werden. Dabei wird zunächst der Quellcode in einen Maschinencode umgewandelt werden. Dies erfolgt durch Erstellen einer Projektmappe.

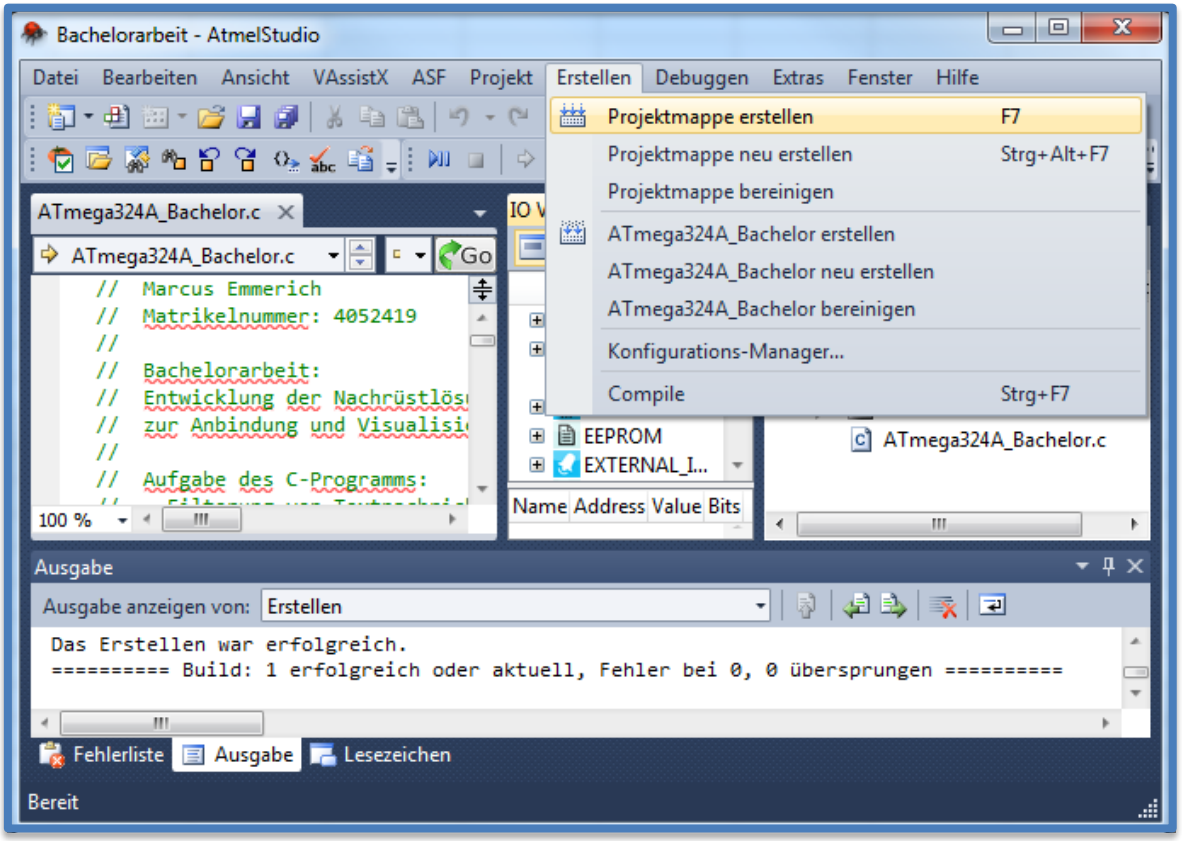

Abbildung 4.26 : Erstellen einer Projektmappe

Nach erfolgreicher Projektierung wird wiederum, unter dem Punkt Device Programming die Programmierung vorgenommen. Dabei wird das aTeVal Evaluationsboard eingebunden und danach im Reiter „Memories“ die Übertragung vorgenommen.

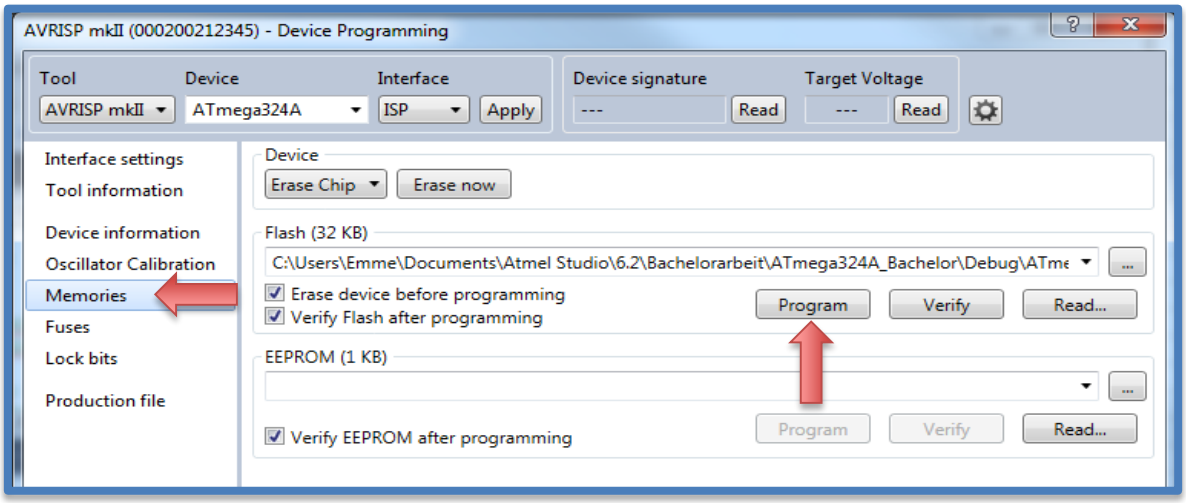

Abbildung 4.27 : Programmierung des Flash-Speichers

Nun ist der Mikrocontroller erfolgreich programmiert. Der nächste Schritt ist es, das RS232/KNX Gateway mit Hilfe der Software KNX-Gate2 zu programmieren, damit alle gefilterten Texte mit Gruppenadressen verknüpft werden können.

5 KNX-Gate2

5.1 Inbetriebnahme

Beim ersten Start zeigt die KNX-Gate2 ein leeres Projekt an.

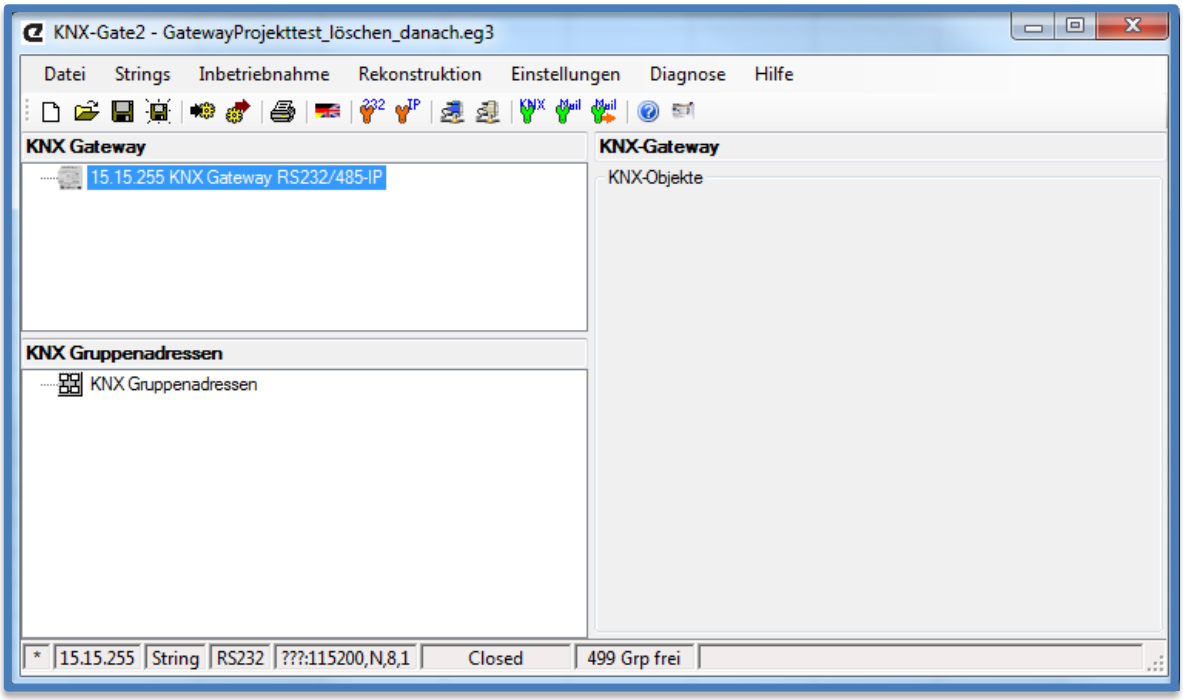

Abbildung 5.1 : Öffnen eines leeren Projektes

Anschließend muss ein neues Projekt unter „Datei“ -> „Neues Projekt“ geöffnet werden. Daraufhin folgen die Einstellungen des verwendeten Protokolls und der Inbetriebnahmeschnittstelle.

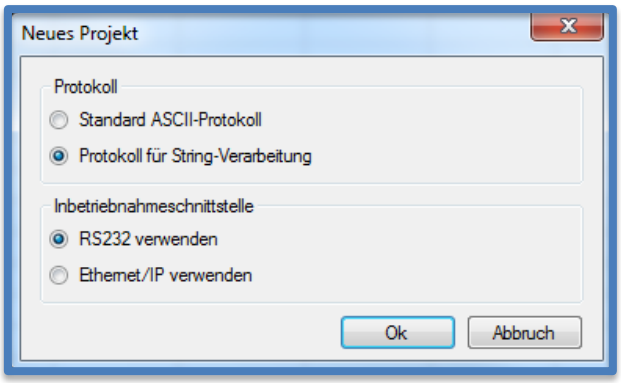

Abbildung 5.2 : Einstellung des Protokolls und der Inbetriebnahmeschnittstelle

Wie in Abbildung 5.2 zu sehen ist, muss dabei das Protokoll „Protokoll für String-Verarbeitung“ und die RS232-Schnittstelle ausgewählt werden (Programmierung über RS232 oder Ethernet/IP). Als nächsten und sehr wichtigen Schritt folgt die Adressierung der physikalischen Adresse. Diese muss identisch der Dummy Adresse in der ETS Software sein, damit das RS232/KNX Gateway erfolgreich eingebunden werden kann. Je nach Wahl der Adresse wird diese unter den Punkt „Einstellungen“ -> „Gateway“ -> „KNX“ definiert (siehe Abbildung 5.3).

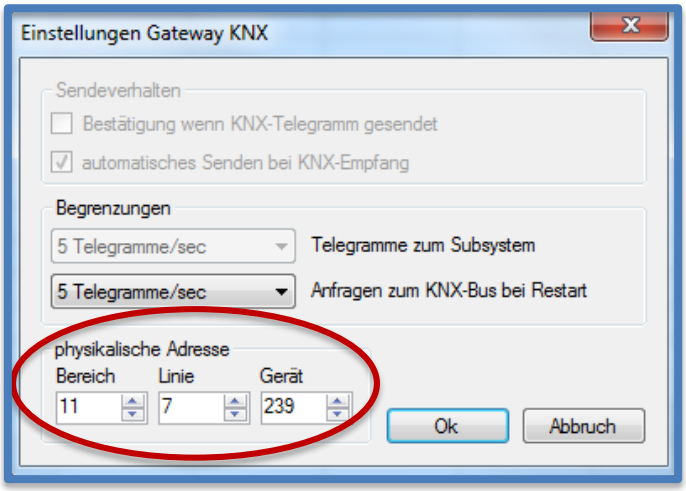

Abbildung 5.3 : Einstellung der physikalischen Adresse

Die Einstellungsparameter für das Übertragen von Daten, über die RS232-Schnittstelle, muss für diese Anwendung nicht extra eingestellt werden, da das Gateway standardmäßig eine Einstellung von 9600 Baud, 1 Stoppbit und 8 Datenbits mit sich bringt. Besteht dennoch der Wunsch die Übertragung anders zu gestalten, können die Einstellung unter „Einstellungen“ -> „Gateway“ -> „Schnittstellen“ -> „RS232 (Einstellungen)“ vorgenommen werden.

5.2 Projektierung

An einem kurzen Beispiel wird nun die eigentliche Projektierung des RS232/KNX Gateway gezeigt. Man betrachte sich eine Brandmeldeanlage, die einen Rauchmelder beinhaltet. Alle aus der Brandmeldeanlage ausgehenden Textinformationen werden durch den ATmega324A gefiltert und an das Gateway weitergeleitet. Nun müssen alle diese Textnachrichten mit Gruppenadressen verbunden werden, damit diese zur späteren Visualisierung genutzt werden können.

Daher sollen zuerst alle verwendeten Textstrings in der KNX-Gate2 Software implementiert werden. Dabei sei angemerkt, dass der Rauchmelder die Meldernummer „0001/1“ besitzt.
Textstrings können in der Software unter dem Punkt „Strings“ -> „Stringtabelle verwalten…“ -> „Text“ erstellt werden. Dabei ist auf die Schreibweise besonders zu achten, damit keine Fehler entstehen können.

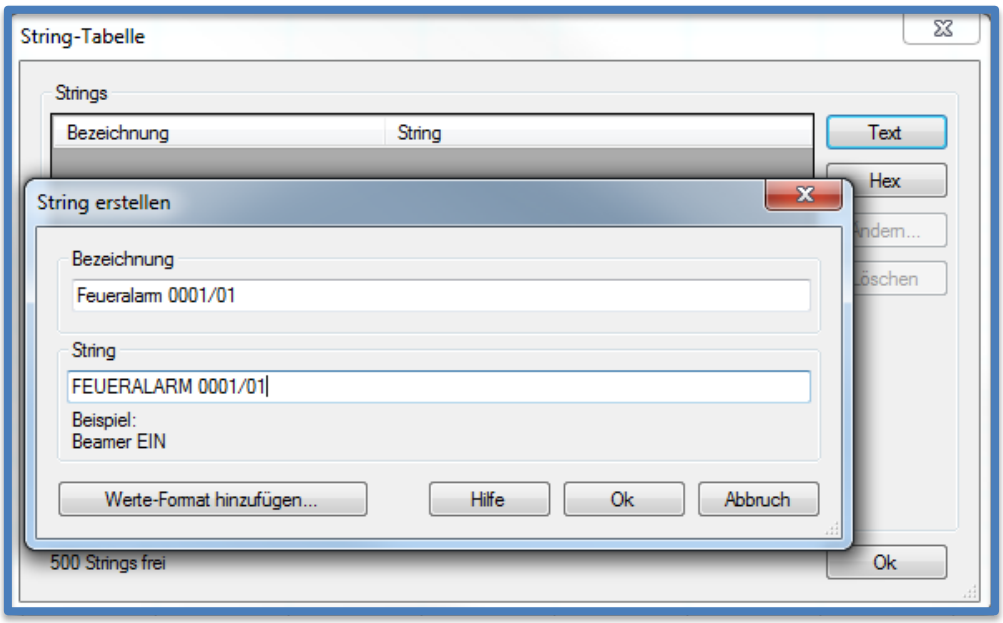

Abbildung 5.4 : Definierung von Strings

Das gleiche Schema wird nun für jede zu verwendende Textmeldung durchgeführt.

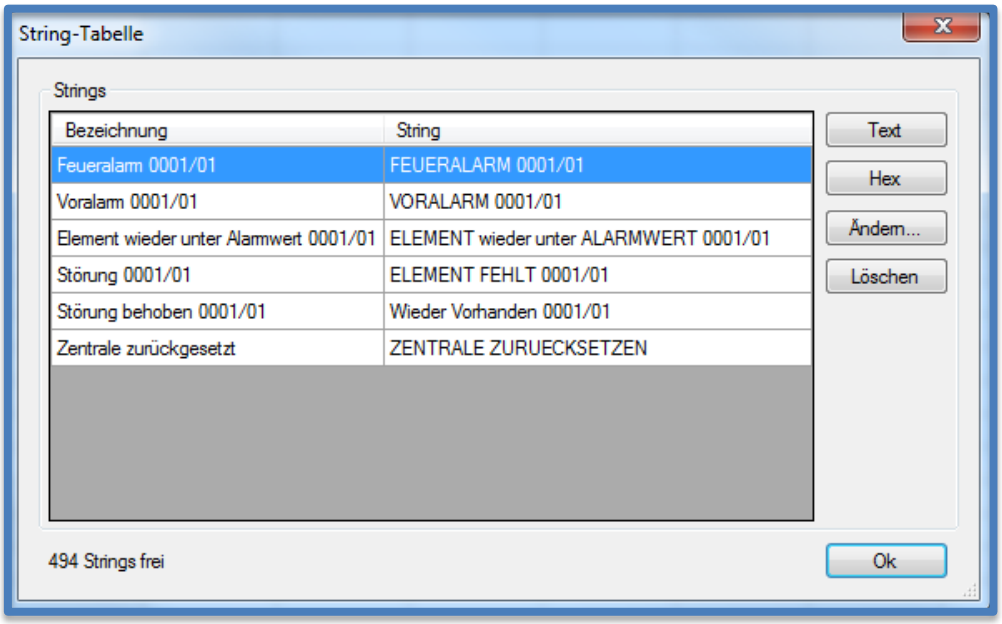

Abbildung 5.5 : Übersicht aller Meldungen eines Melders

Nun werden anschließend Gruppenadressen definiert, damit diese mit den vorher definierten Textstrings verbunden werden können.
Zunächst wird dafür ein neues Projektobjekt hinzugefügt, das anschließend mit Gruppenadressen verknüpft werden kann. Im Objektbaum (Fenster in Abbildung 5.6) können über das Kontextmenü (rechte Maustaste) die benötigten KNX-Objekte erstellt werden.

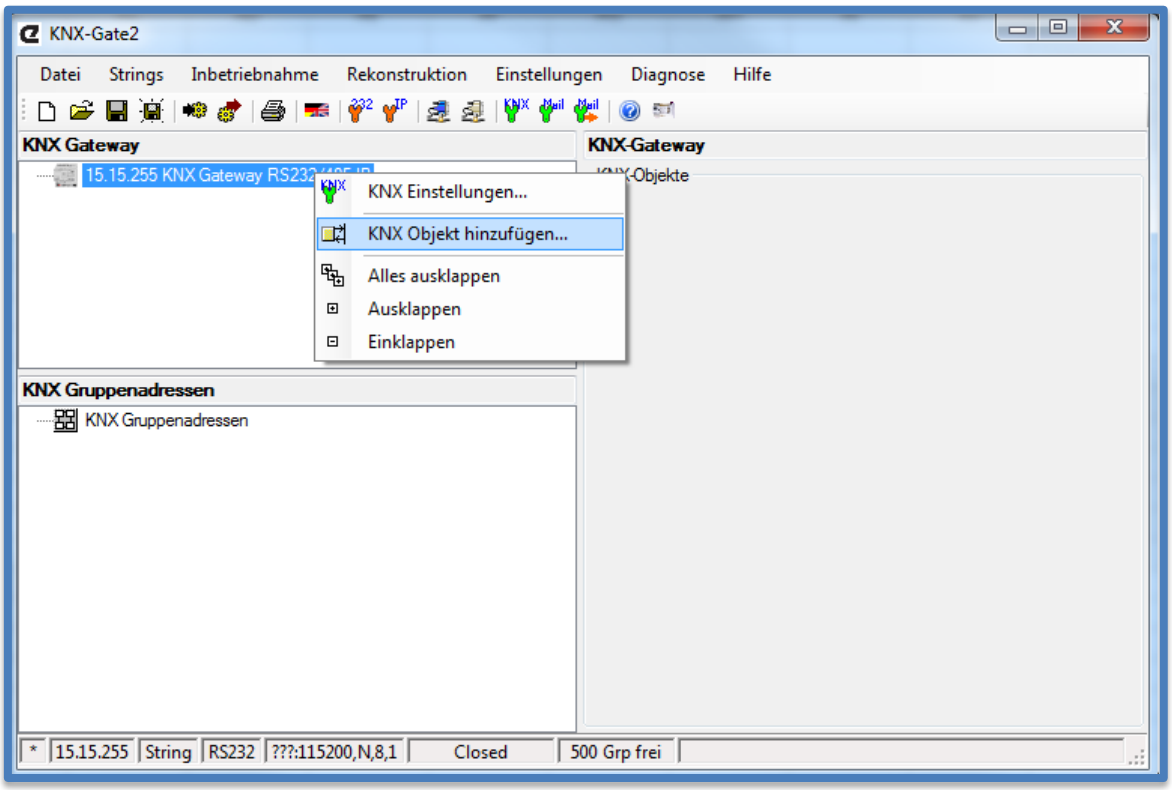

Abbildung 5.6 : Hinzufügen eines Objektes

Dabei wird ein einfaches Schaltobjekt hinzugefügt das entweder eine „1“ oder eine „0“ überträgt.

Die KNX-Gruppenadressen können anschließend entweder aus einem bestehenden Projekt der ETS Software übernommen oder manuell erzeugt werden. Es sei ebenfalls nochmal darauf hingewiesen, dass eine Gruppenadresse (z.B. 2/3/4) in eine Hauptgruppe, Mittelgruppe und Untergruppe unterteilt ist. Diese werden wiederum durch einen Querstrich voneinander getrennt. Innerhalb der Baumstruktur können über die rechte Maustaste mit den Befehlen „Hauptgruppe hinzufügen“, „Mittelgruppe hinzufügen“ und „Gruppenadresse hinzufügen…“ neue Gruppenadressen erzeugt werden. Um ein KNX-Objekt mit den Gruppenadressen verbinden zu können, werden die Untergruppen auf die entsprechenden KNX-Objekte gezogen. Wenn einem Objekt eine Gruppenadresse zugewiesen ist, kann diese im Objektbaum selektiert werden. Mit dem Kontextmenü (rechte Maustaste) in den Listen Sende-Strings oder Empfangsstrings kann dann ein String aus der vorher definierten String-Tabelle hinzugefügt werden. Zusätzlich muss selbstverständlich festgelegt werden, welcher Wert, neben der Gruppenadresse, bei Eintreffen eines Strings gesendet werden soll („1“ oder „0“).

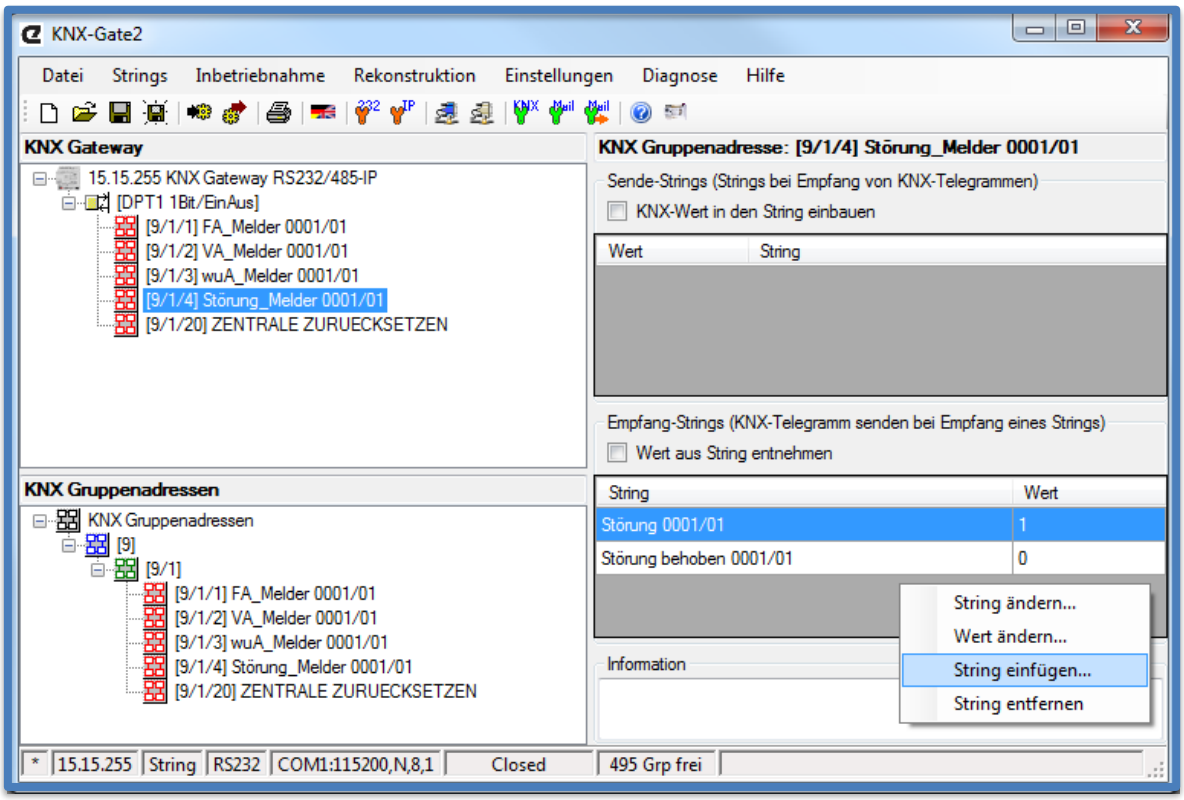

Abbildung 5.7 : Verknüpfung von Strings und Gruppenadressen

Folgende Tabelle zeigt ein Beispiel eines Melders, bei denen alle Statusmeldungen berücksichtigt und mit Gruppenadressen verbunden wurden.

Tabelle 5.1 : Beispiel einer Verknüpfung von Strings und Gruppenadressen

Textstring	KNX-Gruppenadresse	Wert
FEUERALARM 0001/01	9/1/1	1
VORALARM 0001/01	9/1/2	1
ELEMENT wieder unter ALARMWERT 0001/01	9/1/3	0
ELEMENT FEHLT 0001/01	9/1/4	1
Wieder Vorhanden 0001/01	9/1/4	0
ZENTRALE ZURUECKSETZEN	9/1/20	0

Nachdem alle gewünschten Melder und deren Statusmeldungen implementiert und mit Gruppenadressen verbunden wurden, muss die Programmierung noch auf das Gateway übertragen werden.
Dies erfolgt entweder über die RS232-Schnittstelle oder einer Ethernet-Schnittstelle eines Laptops. Je nach Wahl, muss dies in der KNX-Gate2 eingestellt werden (unter den Punkt „Inbetriebnahme“ -> „Inbetriebnahmeschnittstelle“), standardmäßig ist aber RS232 eingestellt. Die Übertragung der Programmierung auf das Gateway folgt anschließend durch den Punkt „Inbetriebnahme“ -> „Download“. Zusätzlich darf nicht vergessen werden, dass das Gateway mit der Firmware auszustatten ist. Dazu muss ein einmaliger Firmware Download durchgeführt werden („Inbetriebnahme“ -> „Firmware Download“).

Nachdem diese Programmierung durchgeführt wurde, ist es nun möglich, die gewünschten Textnachrichten mit anderen Teilnehmern im KNX System zu verbinden und somit auch eine mögliche Visualisierung ermöglichen zu können.

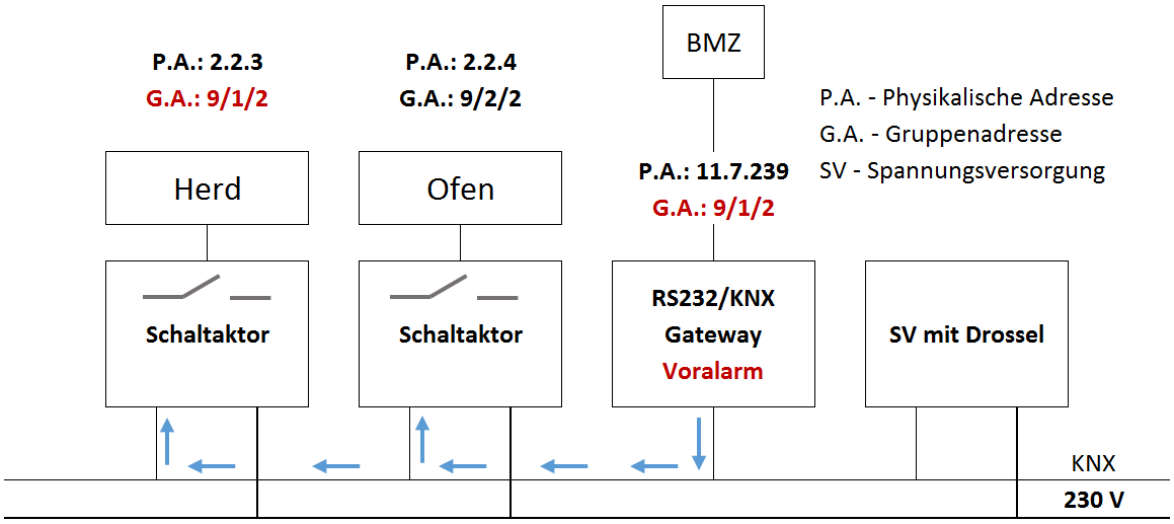

Abbildung 5.8 : Kommunikationsbeispiel einer BMZ mit einem Schaltaktor

In Abbildung 5.8 wird noch einmal kurz gezeigt, welche Möglichkeit eine Vernetzung mit dem KNX-Netz bietet. Nach Auslösen eines Voralarms wird die Gruppenadresse und der dazugehörige Wert „1“ auf den Bus gesendet. Daraufhin wird der Schaltaktor (negative Logik) des Herdes abgestellt, damit eine Erhöhung des Gefahrenherdes minimiert werden kann. Bei der Visualisierung aller eintretenden Statusmeldungen erfolgt dies nach dem gleichen Schema. Dabei werden lediglich durch Leuchtsignale oder Textnachrichten die aufgetretenen Statusmeldungen angezeigt.

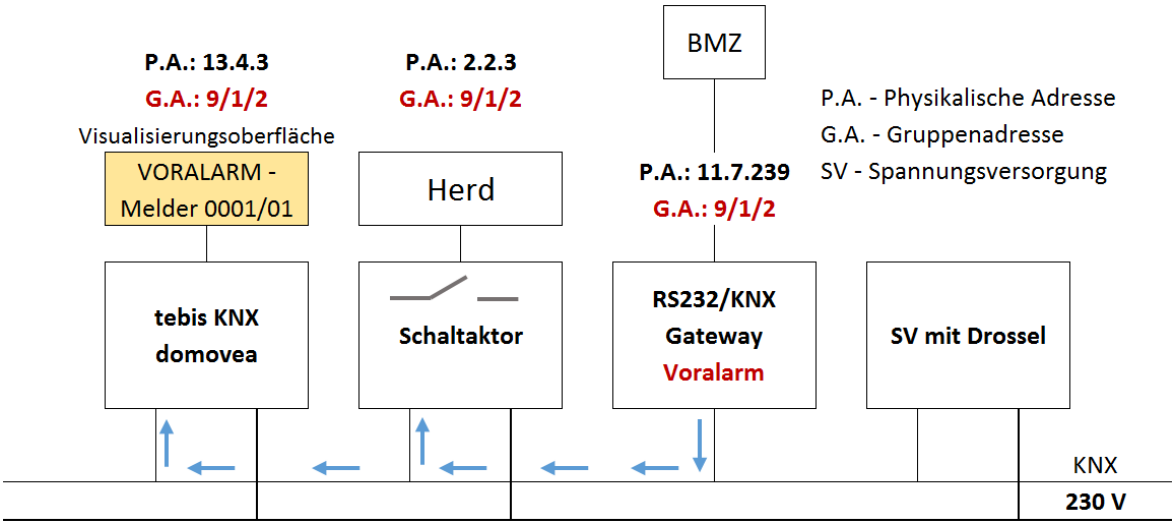

Abbildung 5.9 : Kommunikationsbeispiel einer zusätzlichen Visualisierungsoberfläche

6 Visualisierung mit „tebis KNX domovea“

Nach erfolgreicher Verknüpfung der Brandmeldeanlage mit dem KNX-System soll nun ein kleiner Einblick in den Bereich der Visualisierung verschafft werden. Die Visualisierung wird mit Hilfe der Software „tebis KNX domovea“ realisiert. Diese ist eine weitgehend vorkonfigurierte und damit besonders einfach zu installierende und zu bedienende Visualisierung, welche von „Hager“ entwickelt wurde. Mit dem System tebis KNX domovea ist es möglich alle Gebäudefunktionen von PCs oder mobilen Endgeräten zu visualisieren und zu steuern. Grundvoraussetzung hierzu ist das vernetzte Gebäude mit dem Busstandard KNX. Ein Hausbesitzer kann mit diesem System zusätzlich zu den Schaltern seine Elektroinstallation wie z.B. Beleuchtung oder Rollladen über Touchscreen, PC oder mobilen Endgeräten bedienen. Darüber hinaus können mit domovea Sequenzen, Logiken, Zeitfunktionen und Szenarien erstellt werden. Damit ist die domovea-Visualisierung eine Schnittstelle zwischen Mensch und Gebäude.

6.1 Das System

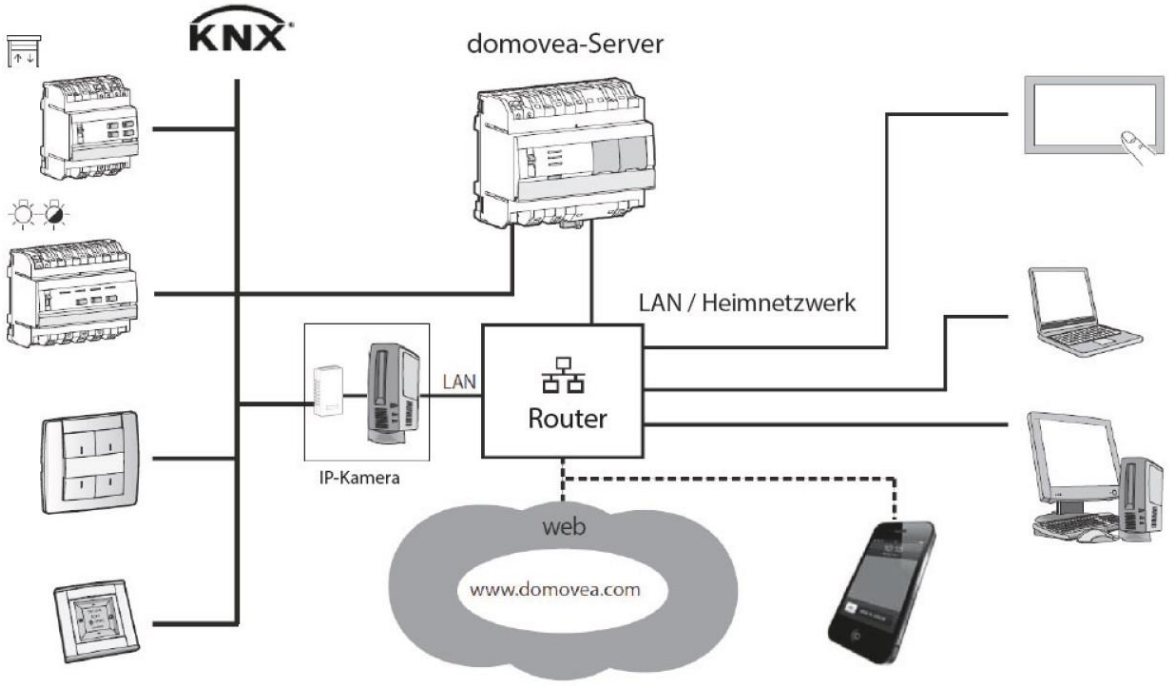

Abbildung 6.1 : Systemaufbau des „tebis KNX domovea“

Voraussetzung für die Verwendung von „tebis KNX domovea“ ist das Vorhandensein eines KNX-Netzes, in dem alle KNX-Sensoren, KNX-Aktoren, Stellantriebe usw. installiert sind. Des Weiteren benötigt es zur Visualisierung ein Ethernet-Netz, in dem alle IP-Clients, wie PC, Touchscreens, Kameras usw. mit dem LAN (lokales Netzwerk) oder W-LAN (drahtloses lokales Netzwerk) verbunden sind. Das domovea-System verbindet diese zwei Netze miteinander. Dafür wird ein domovea-Server benötigt, der die Schnittstelle zwischen KNX-Netz und LAN- bzw. W-LAN-Netz darstellt. Das domovea-System besteht dabei aus drei Modulen:

- Der Server: Schnittstelle zwischen KNX-Bus und dem lokalen Netzwerk des Hauses
- Das Konfigurationstool: Es muss auf einem Touchscreen oder einem PC/Laptop installiert werden. Der Server kann max. 30 Clients bedienen.
- Die Clientsoftware: Die Software wird für die Konfigurierung und die Programmierung der Client-Schnittstelle verwendet.

6.2 Konfigurationstool

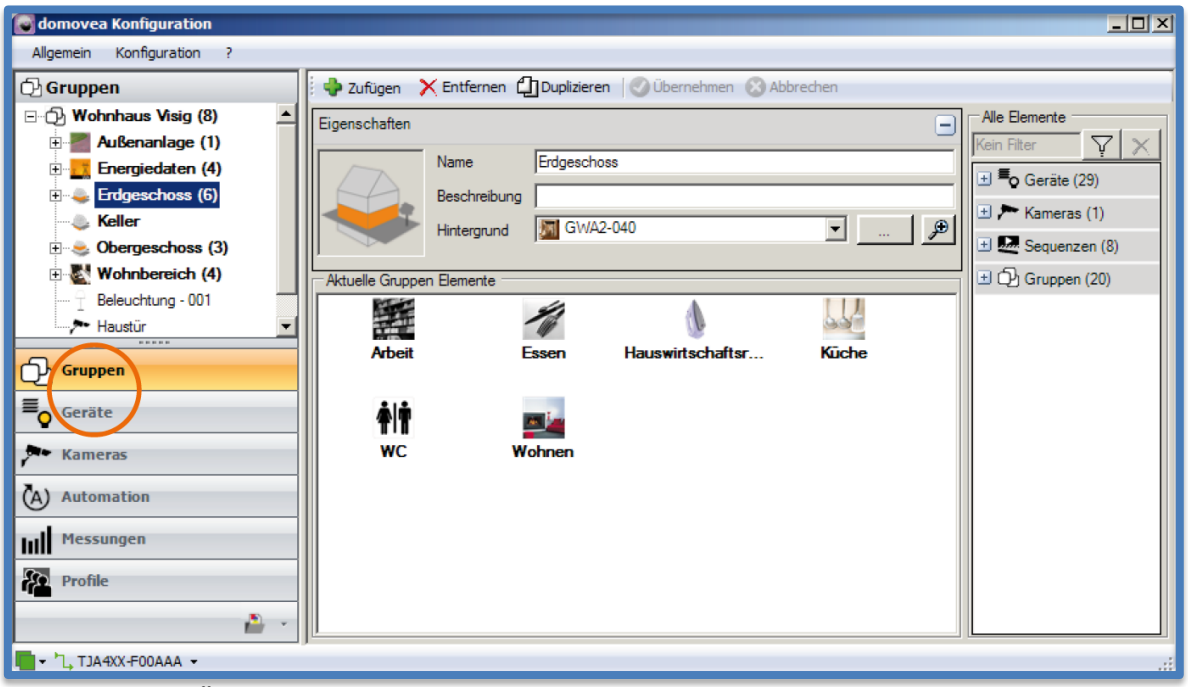

Abbildung 6.2 : Übersicht des Konfigurationstools

Mit dem Konfigurationspool werden die Struktur des Gebäudes und die Gebäudefunktionen angelegt („Gruppen“ werden erzeugt, „Geräte“ werden hinzugefügt). Unter „Geräten“ versteht man eine Funktion in der KNX-Kundenanlage wie z. B. Beleuchtung, Rollladen, Heizung, usw., die mit einer Komponente der KNX Installation verbunden ist und mit Hilfe des domovea- Clients gesteuert oder visualisiert werden kann (pro Installation können maximal 500 „Geräte“ eingebunden werden).
Dabei bestände zum Beispiel die Möglichkeit, eine eigene Gruppe für die Brandmeldeanlage zu errichten, in der alle an der Brandmeldeanlage angeschlossenen Melder integriert sind. Um daraufhin im Konfigurationstool auf die im KNX Projekt verwendeten Gruppenadressen zugreifen zu können, werden aus der ETS-Projektierung die Gruppenadressen über einen Export zu OPC (esf-Datei) exportiert. Diese OPC-Datei enthält alle wichtigen Informationen wie Name, Datengröße und Datenpunkttyp (beinhaltet die Funktion eines Gerätes, z.B. Helligkeitssteuerung einer Lampe) der Gruppenadressen. Anschließend muss diese Datei in das Konfigurationstool importiert werden. Nachdem alle Geräte konfiguriert und die Gruppenadressen importiert wurden, können diese in dem Konfigurationstool miteinander verknüpft werden. Damit kann sozusagen realisiert werden, dass die Gruppenadressen, die aus dem RS232/KNX Gateway in das KNX-System gesendet werden, mit Objekte in dem Konfigurationstool der „tebis KNX domovea“ verknüpft werden können, sodass zum Beispiel ein Feueralarm oder eine

Störung durch ein Lichtsignal oder als erscheinenden Warntext angezeigt bzw. signalisiert werden. Der Kunde sieht dies dann als Darstellung in der Bedieneroberfläche (Client-Software).

6.3 Die Visualisierungs- und Bedienoberfläche (Client-Software)

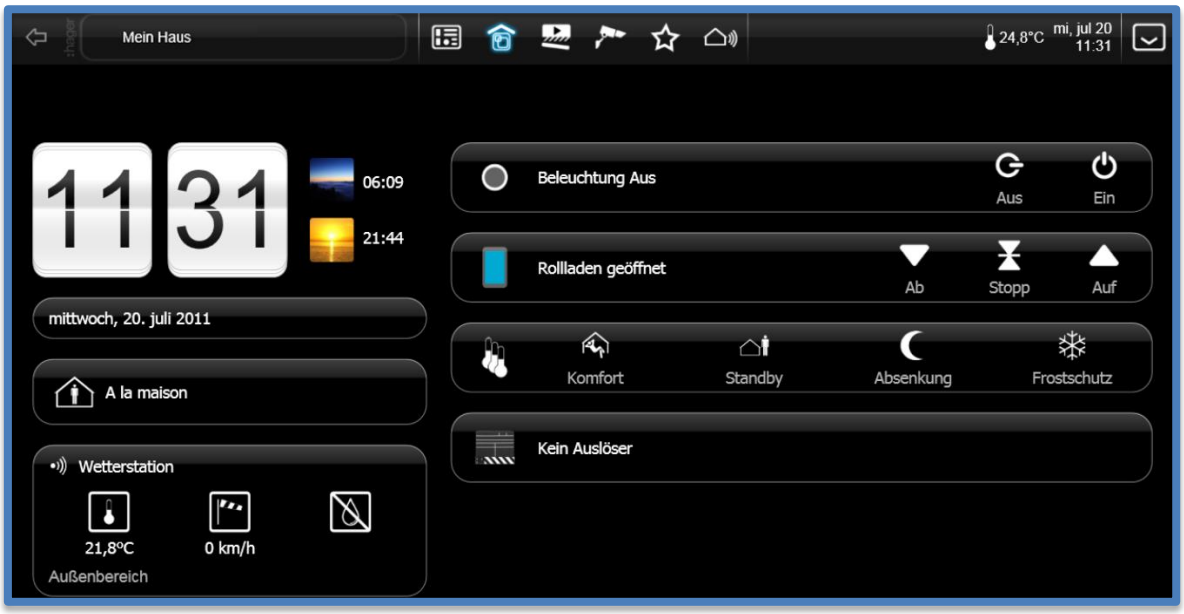

Abbildung 6.3 : Bedienoberfläche des "tebis KNX domovea"

In der Bedienoberfläche können dann alle integrierten Geräte angezeigt und bedient werden. Domovea ist an der Stelle, in der Client-Server-Architektur, sehr frei. Der Client kann beispielsweise ein Standard PC sein, auf den eine windowsbasierte Software läuft oder mit Hilfe eines Smartphones oder Tablets bedient werden (IOS und Android basiert). Daher ist es ohne Probleme möglich, auch aus der Entfernung auf alle Geräte im Haus zugreifen zu können, bzw. sich Informationen über ein Status eines Gerätes anzeigen zu lassen. Um den Fernzugriff zu ermöglichen, muss über das domovea Portal (https://www.domovea.com) der Fernzugriff freigeschaltet werden. Die Voraussetzungen dafür sind, dass der domovea Server installiert und eingeschaltet ist. Zusätzlich muss dieser mit einen Netzwerk verbunden werden, dass Zugriff auf Internet gewährleistet. Des Weiteren erhält man eine Lizenzdatei des Herstellers, die in Verbindung mit der Seriennummer des Servers, benötigt wird, um den hauseigenen domovea Server mit dem Internet Server des Herstellers zu verbinden. Durch diese Verknüpfung kann erreicht werden, dass mobile Geräte mit dem KNX-System des Hauses kommunizieren können. Dadurch ist es zu jeder Zeit möglich, nun auch den Status der Brandmeldeanlage verfolgen zu können.

7 Zusammenfassung und Ausblick

Ziel der Arbeit war es, die Nachrüstlösung einer Brandmeldeanlage zur Anbindung und Visualisierung an das KNX-System zu entwickeln. Hierbei war es wichtig, alle aus der Brandmeldeanlage auftretenden Statusmeldungen zu berücksichtigen, um die Grundlage zu schaffen, dass all diese visuell im KNX-System dargestellt werden können.
Um dieses Problem zu realisieren, wurde zunächst analysiert, welche Möglichkeiten einerseits die Brandmeldeanlage und andererseits das KNX-System bieten, eine Kommunikation mit Fremdsystemen herzustellen. Daraus resultierte die Verwendung eines RS232/KNX Gateway, da dieses Textnachrichten über eine RS232-Schnittstelle aufnehmen und mit Gruppenadressen verbinden kann. Das Problem bestand dabei wiederum, dass das Gateway es nur ermöglicht konstante Strings aufzunehmen. Da die Brandmeldeanlage aber auch zeitlich veränderliche Textinformation ausgibt, war die einfache Anbindung somit nicht möglich. Daher mussten die ausgehenden Textinformationen zusätzlich mit Hilfe eines Mikrocontrollers gefiltert werden, bevor diese zum RS232/KNX Gateway weitergeleitet werden konnten. Darauf folgte, dass alle gefilterten Texte mit Gruppenadressen verbunden werden konnten und somit eine Verknüpfung zum KNX-System hergestellt wurde. Dies beherbergt somit die Möglichkeit, mit Hilfe der „tebis KNX domovea“ eine zusätzliche Visualisierung aller auftretenden Ereignisse realisieren zu können.

Da die Lösung der gestellten Aufgabe zum Beispiel mit Hilfe eines Evaluationsboardes realisiert wurde, das über eine USB-Schnittstelle die Spannungsversorgung des Mikrocontrollers und der beiden RS232/TTL Wandler bereit stellte, wäre diese Lösung für die Verwendung und dem Verkauf an einen Kunden sehr ungeeignet. Daher wäre der nächste Schritt, ein eigenständiges Modul zu entwickeln, das die einfachere Bedienbarkeit ermöglichen und somit für die Anbringung an einen Kunden verbessern würde.

Anhang

A. Filterprogramm des ATmega324A

```
1  //*********************************************************************
2  //
3  // Hochschule Anhalt - FB EMW
4  // Marcus Emmerich
5  // Matrikelnummer: 4052419
6  //
7  // Bachelorarbeit:
8  // Entwicklung der Nachrüstlösung einer Brandmeldeanlage
9  // zur Anbindung und Visualisierung an das KNX System
10 //
11 // Aufgabe des C-Programms:
12 // - Filterung von Textnachrichten einer Brandmeldeanlage
13 // - Empfang von Textnachrichten über UART
14 // - Statusmeldungen und Meldernummern sollen gefiltert werden
15 // - gefilterte Textzeilen sollen über UART ausgegeben werden
16 //
17 //*********************************************************************
18
19 #define BAUD 9600 //geforderte Baudrate
20 #define F_CPU 7372800UL//Geschwindigkeit des Quarzes
21
22 #include <avr/io.h> // enhält Namen der verwendeten Register
23 #include <util/setbaud.h> //definiert Baudrate im UART Register
24
25 #include <stdio.h> //vereinbart Ein- und Ausgabefunktionen, Typen und Makros
26 #include <string.h> //enthält Funktionen zur Bearbeitung von Zeichenketten
27
28 #define laenge 200 //Längenangabe der Zeichenketten
29 #define laenge2 200
30 #define laengeelka 200
31
32 /*UNTERPROGRAMME
33 ____________________________________________________________________________*/
34
35 /*Uart-Schnittstelle initialisieren
36 ----------------------------------*/
37 void uart_init(void)
38 {
39  UBRR0= UBRR_VALUE;
40
```

```
41  #if USE_2X
42  /* U2X-Modus erforderlich */
43  UCSR0A |= (1 << U2X0);
44  #else
45  /* U2X-Modus nicht erforderlich */
46  UCSR0A &= ~(1 << U2X0);
47  #endif
48
49  UCSR0C  = 0b10000110; // Asynchron 8 Daten- und 1 Stoppbit
50  UCSR0B |= (1<<TXEN0); // TXEN setzen/einschalten
51  UCSR0B |= (1<<RXEN0); // RXEN setzen/einschalten
52 }
53
54 /*Empfangen von Daten
55 ----------------------*/
56 uint8_t uart_getc(void)
57 {
58  // warten bis Zeichen verfuegbar
59  while (!(UCSR0A & (1<<RXC0)))
60  ;
61  return UDR0; // Zeichen aus UDR an Aufrufer zurückgeben
62  }
63
64 void uart_gets( char* Buffer, uint8_t MaxLen )
65 {
66  uint8_t NextChar;
67  uint8_t StringLen = 0;
68
69  NextChar = uart_getc(); // Warte auf und empfange das nächste Zeichen
70
71  // Sammle solange Zeichen, bis:
72  // * entweder das String Ende Zeichen kam
73  // * oder das aufnehmende Array voll ist
74  while( NextChar != '\n' && StringLen < MaxLen - 1 )
75  {
76   *Buffer++ = NextChar;
77   StringLen++;
78   NextChar = uart_getc();
79  }
80  // Noch ein '\0' anhängen um einen Standard
81  // C-String daraus zu machen
82  *Buffer = '\0';
83 }
84
```

```
85  /*Senden von Daten
86  --------------------*/
87  int uart_putc(char data)
88  {
89   while (!(UCSR0A & (1<<UDRE0))) // warten bis Senden moeglich
90   ;
91   UDR0 = data; // sende Zeichen
92   return 0;
93  }
94
95  /* puts ist unabhaengig vom Controllertyp */
96  void uart_puts (char *s)
97  {
98   while (*s)
99   {/* so lange *s != '\0' also ungleich dem
100   "String-Endezeichen(Terminator)" */
101   uart_putc(*s);
102   s++;
103  }
104 }
105
106
107
108 /*HAUPTPROGRAMM
109 ____________________________________________________________________________*/
110 int main (void)
111 {
112  uart_init();
113
114  //Definitionen der Logiken, müssen hier zuerst definiert werden
115  //-------------------------------------------------------------------------
116
117  int ZZ_logik,Gr_logik,zsp=0;
118  char elkatxt[laengeelka]=""; // muss leer sein
119  int endtxt=0; //dient zur Ausgabe, signalisiert Länge der Statusmeldung
120
121
122  /*Hauptdurchlauf des Programms
123  ____________________________________________________________________________*/
124  while(1)
125  {//Definitionen
126   int i;
127   char zeichenkette[laenge];
128   // Definitionen Gruppennummer
```

```
129   char suchGr[]={"Gruppe"}, suchFA[]={"FEUERALARM"};
130   char suchVA[]={"VORALARM"}, suchEF[]={"ELEMENT FEHLT"};
131   char suchwuA[]={"wieder unter ALARMWERT"};
132   char suchwv[]={"Wieder Vorhanden"}, suchZZ[]={"CKSETZEN"};
133   char zeichenkette2[laenge2]="\0"; //muss als erstes leer sein
134   char* suchErg;
135
136   int leerz1=0;
137   int leerz2=0;
138
139   /*-----------------------------------------------------------------------*/
140   // Müssen bei jedem Durchlauf neu "0" gesetzt werden
141
142   ZZ_logik=0;
143   Gr_logik=0;
144
145   /*Empfangen der Textinformationen
146   ------------------------------------------------------------------------ */
147
148   uart_gets(zeichenkette, sizeof(zeichenkette));
149
150   /*Suche nach Wörter
151   -----------------------------------------------------------------------*/
152   //-----------------------------------------------
153   //Suche Wort 'FEUERALARM' in der Zeichenkette
154   suchErg=strstr(zeichenkette,suchFA);
155   //schreibe Zeiger in Speicher von elkatxt
156   if(suchErg!=0)
157   {strcpy(elkatxt,suchErg);
158    endtxt=10; // 10 entspricht die Länge des Wortes Feueralarm
159    suchErg=0; // muss auf "0", wegen nächsten Durchlauf des Programms
160    zsp = 1; //signalisiert, dass ein Wort abgespeichert ist
161   }
162   //-----------------------------------------------
163   //Suche Wort 'VORALARM' in der Zeichenkette
164   suchErg=strstr(zeichenkette,suchVA);
165   //schreibe Zeiger in Speicher von elkatxt
166   if(suchErg!=0)
167   {strcpy(elkatxt,suchErg);
168    endtxt=8; // 8 entspricht die länge des Wortes Voralarm
169    suchErg=0; // muss auf "0", wegen nächsten Durchlauf des Programms
170    zsp = 1; //signalisiert, dass ein Wort abgespeichert ist
171   }
172
```

```
173   //-------------------------------------------------
174   //Suche Wort 'ELEMENT FEHLT' in der Zeichenkette
175   suchErg=strstr(zeichenkette,suchEF);
176   //schreibe Zeiger in Speicher von elkatxt
177   if(suchErg!=0)
178   {strcpy(elkatxt,suchErg);
179    endtxt=13; // 13 entspricht die länge der Zeichenkette element fehlt
180    suchErg=0; // muss auf "0", wegen nächsten Durchlauf des Programms
181    zsp = 1; //signalisiert, dass ein Wort abgespeichert ist
182   }
183
184   //-------------------------------------------------
185   //Suche Wort 'wieder unter ALARMWERT' in der Zeichenkette
186   suchErg=strstr(zeichenkette,suchwuA);
187   //schreibe Zeiger in Speicher von elkatxt
188   if(suchErg!=0)
189   {strcpy(elkatxt,suchErg);
190    endtxt=22; // 10 entspricht die länge des Wortes
191    suchErg=0; // muss auf "0", wegen nächsten Durchlauf des Programms
192    zsp = 1; //signalisiert, dass ein Wort abgespeichert ist
193   }
194
195   //-------------------------------------------------
196   //Suche Wort 'wieder vorhanden' in der Zeichenkette
197   suchErg=strstr(zeichenkette,suchwv);
198   //schreibe Zeiger in Speicher von elkatxt
199   if(suchErg!=0)
200   {strcpy(elkatxt,suchErg);
201    endtxt=16; // 10 entspricht die länge des Wortes
202    suchErg=0; // muss auf "0", wegen nächsten Durchlauf des Programms
203    zsp = 1; //signalisiert, dass ein Wort abgespeichert ist
204   }
205
206   //-------------------------------------------------
207   //Suche Wort 'CKSETZEN' in der Zeichenkette
208   suchErg=strstr(zeichenkette,suchZZ);
209   //schreibe Zeiger in Speicher von elkatxt
210   if(suchErg!=0)
211   {suchErg="ZENTRALE ZURUECKSETZEN";
212    strcpy(elkatxt,suchErg);
213    endtxt=22; // 22 entspricht "ZENTRALE ZURUECKSETZEN"
214    suchErg=0; // muss auf "0", wegen nächsten Durchlauf des Programms
215    ZZ_logik=1; // Logik erfüllt, da Wort enthalten
216   }//Zwischenspeicher nicht erforderlich,
```

```
217     //da keine Gruppe mit ausgegeben wird
218
219   //-------------------------------------------------
220   //Suche Wort 'Gruppe' in der Zeichenkette
221   suchErg=strstr(zeichenkette,suchGr);
222   //schreibe Zeiger in Speicher von "zeichenkette2"
223   if(suchErg!=0)
224   {strcpy(zeichenkette2,suchErg);
225    // muss abgespeichert werden, um damit weiterzuarbeiten
226    //damit die Meldernummer ermittelt werden kann
227   }
228
229
230   /*Filterung der Meldernummer
231   ------------------------------------------------------------------------*/
232   //soll nur durchlaufen werden, wenn Zwischenspeicher voll ist
233   if(zsp==1)
234   {
235    for(i=0;*(zeichenkette2+i)!=' ';i++)
236     {//gibt an, an welcher Stelle das erste "Leerzeichen" ist
237      if(*(zeichenkette2+(i+2))==' ')
238      // +2, Stelle nach " " wird benötigt
239      leerz1=i+2;
240      else
241      if(i==laenge2-1)
242      {Gr_logik=0;break;} //Vermeidung eines Überlaufes,
243       //falls kein Leerzeichen vorhanden sein sollte
244       // und Logik 0, da keine Meledernummer vorhanden
245     }
246     //Stelle nach dem ersten Zeichen
247     leerz1=leerz1+1;
248     if(leerz1>1)
249     {
250      for(i=leerz1;*(zeichenkette2+i)!=' ';i++)
251      {if(*(zeichenkette2+(i+2))==' ')
252       {//Stelle von 2.Leerzeichen
253        leerz2=i+1;
254        Gr_logik=1; //erst dann ist Gruppennachricht enthalten
255       }
256       else
257       if(i==laenge2-1)
258       {Gr_logik=0;break;} //Vermeidung eines Überlaufes,
259       // falls kein Leerzeichen vorhanden sein sollte
260       // und Logik 0, da keine Meledernummer vorhanden
```

```
261       }
262      }
263    }
264
265    /*Ausgabe zum RS232/KNX Gateway
266    ------------------------------------------------------------------*/
267
268    //Bedingung muss efüllt sein
269    if((Gr_logik==1 && zsp==1)||ZZ_logik==1)
270    {
271     if(Gr_logik==1 && zsp==1)
272     {
       elkatxt[endtxt]=' '; //sorgt für Leerzeichen nach der
273  Statusmeldung
274      endtxt=endtxt+1; //bei "ZENTRALE ZURUECKSETZEN" kein “ “ gefordert
275      //Speichert die Meldernummer in elkatxt
276      for(i=leerz1;i<=leerz2;i++)
277      {
278       elkatxt[endtxt]=*(zeichenkette2+i);
279       endtxt=endtxt+1;
280      }
281     }
282     //An Elkatxt soll ein \r gehangen werden
283     elkatxt[endtxt]='\r';
284     endtxt=endtxt+1;
285     //und eine "\0" gehangen werden
286     elkatxt[endtxt]='\0';
287
288     uart_puts( elkatxt );
289     zsp = 0; // Da Ausgabe erfolgte, kann Speicher "gelöscht werden"
290
291    }
292   } // Ende der while-Schleife
293   return 0; // never reached
294  }
```

Abbildungsverzeichnis

Abbildung 2.1 : Kommunikationablauf im KNX-System 5
Abbildung 2.2 : Aufbau einer Brandmeldeanlage 6
Abbildung 2.3 : DB-9 Stecker (männlich) und DB-9 Buchse (weiblich) 8
Abbildung 2.4 : Auftauchende Namensänderungen bei der RS232-Schnittstelle 9
Abbildung 2.5 : Benötigtes System zur Verknüpfung des KNX-System mit der BMZ 12
Abbildung 2.6 : Verwendung eines RS232 zu KNX Gateways 12
Abbildung 2.7 : System zur Filterung von Textnachrichten 13
Abbildung 2.8 : Zielsetzung der gewünschten Filterung 14
Abbildung 2.9 : Benötigte RS232 zu TTL Schnittstelle 15
Abbildung 2.10 : Zennio SKX Open 16
Abbildung 2.11 : KNX Gateway RS232/485-IP von „Elka Elektronik“ 17
Abbildung 2.12 : KNX-GW-RS232-RS485 von „arcus-eds“ 18
Abbildung 2.13 : ATmega324A Blockschaltbild 21
Abbildung 2.14 : Schwingquarz 22
Abbildung 2.15 : Programmierungsabfolge 23
Abbildung 2.16 : AVRISP mkll 24
Abbildung 2.17 : Struktur des AVRISP mkll 25
Abbildung 2.18 : DIAMEX PROG-S 25
Abbildung 2.19 : ATMEL Evaluations-Board Ver. 2.0.1 von Pollin 26
Abbildung 2.20 : aTeVaL (Atmel Evaluationsboard) 27
Abbildung 2.21 : Schaltbild eines RS232-TTL-Converters von Pollin 28
Abbildung 2.22 : RS232 zu TTL Wandler 28
Abbildung 3.1 : Anschluss des RS232/KNX Gateways 30
Abbildung 3.2 : Kreuzung der Anschlüsse 31
Abbildung 3.3 : Nullmodem-Kabel in Verbindung eines Gender Changer 32
Abbildung 3.4 : TTL - Anschlussverbindungen 32
Abbildung 4.1 : Beispiel einer Textausgabe der BMZ 35
Abbildung 4.2 : Programmablaufplan 36
Abbildung 4.3 : Senden und Empfangen mit Hilfe eines Laptops 38
Abbildung 4.4 : Übertragung von Textinformationen mit Hilfe von HTerm 0.8.1beta 38
Abbildung 4.5 : Initialisierung der UART-Schnittstelle Teil A 45
Abbildung 4.6 : Initialisierung der UART-Schnittstelle Teil B 46
Abbildung 4.7 : Senden einer Zeichenkette Teil A 46

Abbildung 4.8 : Senden einer Zeichenkette Teil B 47

Abbildung 4.9 : Empfangen von Zeichenketten mit dem UART 48

Abbildung 4.10 : Stringbeispiel zur Erläuterung des Suchverlaufes 50

Abbildung 4.11 : Definitionen der gesuchten Zeichenketten und Suchablauf einer Statusmeldung 50

Abbildung 4.12 : Suchablauf der Statusmeldung "ZENTRALE ZURÜCKSETZEN" 51

Abbildung 4.13 : Suchablauf der Zeichenkette "Gruppe" 52

Abbildung 4.14 : Filterung der Zeichenkette "Gruppe" 53

Abbildung 4.15 : Filterung der Meldernummer 53

Abbildung 4.16 : Suchablauf des ersten Leerzeichens 53

Abbildung 4.17 : Suchablauf des zweiten Leerzeichens 54

Abbildung 4.18 : Verknüpfung der Meldernummer mit der Ausgabe-Variable 55

Abbildung 4.19 : Setzen der Bedingung und eines Leerzeichens 55

Abbildung 4.20 : Ausgabe aller gefilterten Texte 56

Abbildung 4.21 : Erstellung eines neuen Projektes 57

Abbildung 4.22 : Programmübersicht des Atmel Studios 6.2 58

Abbildung 4.23 : Einbindung des aTeVaL Evaluationsboardes 62

Abbildung 4.24 : Auswahl des AVRISP mkll Clones 63

Abbildung 4.25 : Setzen des Fuse Low Bytes 63

Abbildung 4.26 : Erstellen einer Projektmappe 64

Abbildung 4.27 : Programmierung des Flash-Speichers 64

Abbildung 5.1 : Öffnen eines leeren Projektes 65

Abbildung 5.2 : Einstellung des Protokolls und der Inbetriebnahmeschnittstelle 65

Abbildung 5.3 : Einstellung der physikalischen Adresse 66

Abbildung 5.4 : Definierung von Strings 67

Abbildung 5.5 : Übersicht aller Meldungen eines Melders 67

Abbildung 5.6 : Hinzufügen eines Objektes 68

Abbildung 5.7 : Verknüpfung von Strings und Gruppenadressen 69

Abbildung 5.8 : Kommunikationsbeispiel einer BMZ mit einem Schaltaktor 70

Abbildung 5.9 : Kommunikationsbeispiel einer zusätzlichen Visualisierungsoberfläche 70

Abbildung 6.1 : Systemaufbau des „tebis KNX domovea“ 71

Abbildung 6.2 : Übersicht des Konfigurationstools 72

Abbildung 6.3 : Bedienoberfläche des "tebis KNX domovea" 73

Tabellenverzeichnis

Tabelle 2.1 : Maximale Übertragungsrate und Leitungslänge 8

Tabelle 2.2 : Pinbelegungen der 9-poligen RS232-Schnittstelle 9

Tabelle 2.3 : Ausschnitt der ASCII Zeichentabelle 11

Tabelle 2.4 : Festgelegte Standards eines Zeilenumbruchs 11

Tabelle 2.5 : Kommunikationsbeispiel 12

Tabelle 2.6 : Übersicht aller gesuchten Textinformationen 15

Tabelle 2.7 : Fehlertoleranzen in Abhängigkeit der Taktfrequenz 22

Tabelle 2.8 : : Programmiersoftare für Mikrocontroller 23

Tabelle 3.1 : Pinbelegungen des 40-poligen Wannensteckers – Teil 1 33

Tabelle 3.2 : Pinbelegungen des 40-poligen Wannensteckers – Teil 2 34

Tabelle 4.1 : Programmübersicht 37

Tabelle 4.2 : UCSRA Resgister 39

Tabelle 4.3 : UCSRB Register 40

Tabelle 4.4 : UCSRC Register 41

Tabelle 4.5 : UPM1:0 Definierung 42

Tabelle 4.6 : USBS Bit 42

Tabelle 4.7 : UCSZ2:0 Bits 42

Tabelle 4.8 : Fuse High Byte 58

Tabelle 4.9 : Extended Fuse Byte 60

Tabelle 4.10 : Fuse Low Byte 60

Tabelle 4.11 : Device Clocking Option 61

Tabelle 4.12 : Low Power Crystal Oscillator Operating Modes 61

Tabelle 4.13 : Start-up Times for the Low Power Crystal Oscillator Clock Selection 62

Tabelle 4.14 : Fuse Low Byte Veränderungen 62

Tabelle 5.1 : Beispiel einer Verknüpfung von Strings und Gruppenadressen 69

Literaturverzeichnis

Bücher und Zeitschriften

[Sch10] Schmitt, G.
Programmierung in Assembler und C – Schaltungen und Anwendungen
Oldenbourg Verlag München, 2010

[Wie03] Wiegelmann, J.
Softwareentwicklung in C für Mikroprozessoren und Mikrocontroller
Hüthig Verlag Heidelberg, 2003

[Spa11] Spanner, G.
AVR-Mikrocontroller in C programmieren
Franzis Verlag Poing, 2011

[Lie06] Liehr,B.-C.
Einführung in der Programmierung des AVR-Controllers
Diplomarbeit, Fachhochschule Augsburg, 2006

[Sch08] Schäffer, F.
Hardware und C-Programmierung in der Praxis
Elektor-Verlag, 2, Aachen, 2008

[BFG08] Bundestechnologiezentrum für Elektro- und Informationstechnik e.V.
KNX Grundkurs
KNX Association, bfe Oldenburg, 2008

Webseiten

[mcn] Mikrocontroller.net
http://www.mikrocontroller.net/

[wiki] Wikipedia: RS-232
http://de.wikipedia.org/wiki/RS-232

[knx] KNX Association
http://www.knx.de/knx-de/

[esa] Elektroniknet.de: Strategien zur Auswahl der richtigen MCU
http://www.elektroniknet.de/halbleiter/mikrocontroller/artikel/100982/

[aca] Atmel Corporation: ATMEGA324A
http://www.atmel.com/devices/ATMEGA324A.aspx

[eab] eHaJo: aTeVaL-Board
http://dokuwiki.ehajo.de/artikel:ateval
KNX Association, bfe Oldenburg, 2008

[aam] Atmel Corporation: AVRISP Mkll
http://www.atmel.com/webdoc/avrispmkii/index.html

[pae] Pollin: ATMEL Evaluations-Board Version 2.0.1 - Bausatz

http://www.pollin.de/shop/dt/MTY5OTgxOTk-/Bausaetze_Module/Bausaetze/ATMEL_Evaluations_Board_Version_2_0_1_Bausatz.html

[rdp] Reichelt.de: DIAMEX-PROG-S

http://www.reichelt.de/Programmer-Entwicklungstools/DIAMEX-PROG-S/3/index.html?;ACTION=3;LA=2;ARTICLE=115385;GROUPID=2969;artnr=DIAMEX+PROG-S;SID=11UHbcA38AAAIAAATaUeg106ca2b70a1f9b49c1071601dd4f2cb5l

Danksagung

An dieser Stelle möchte ich mich für die Unterstützung beim Anfertigen meiner Bachelorarbeit bei EAB – G. Sandow GmbH bedanken. Insbesondere die positive und offene Einstellung etablierter Mitarbeiter Studenten gegenüber erleichterte mir die Arbeit sehr. Ein besonderer Dank geht an meinen Betreuer Dominique Lieder für seine hilfreichen Anregungen und seine konstruktive Kritik.