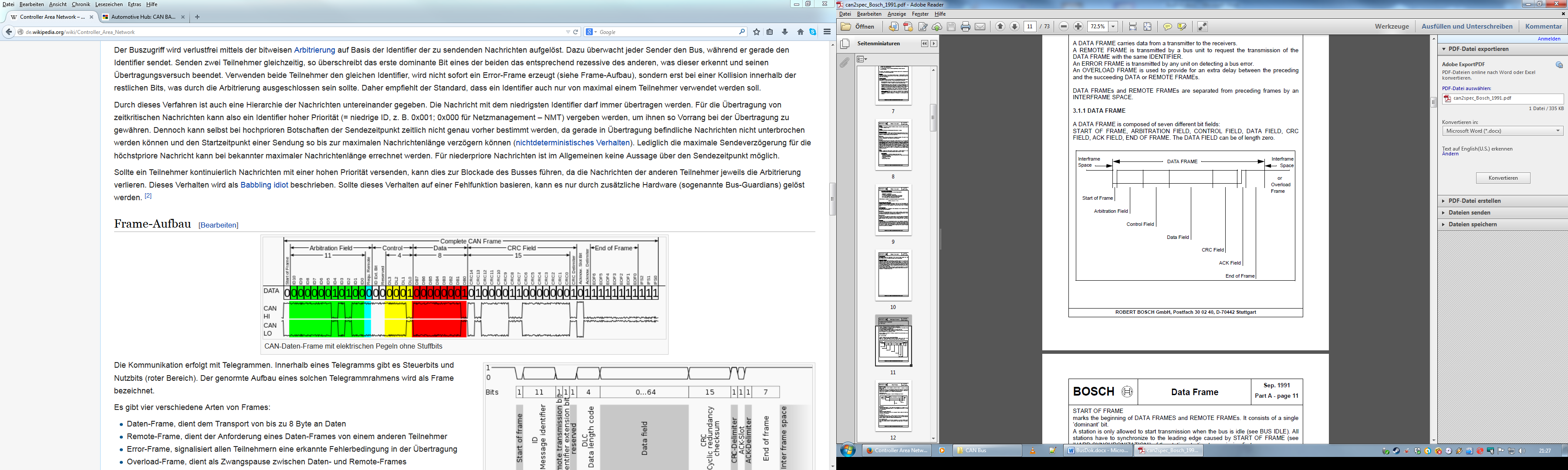
Theorie

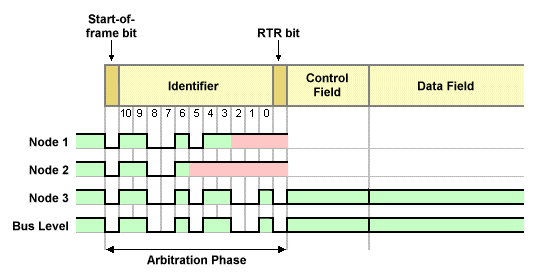
Ursprünglich wurde das Controller Area Network (CAN) von Bosch entwickelt, um die Steuergeräte in Automobilen zu verbinden. In einem solchen Umfeld ist es wichtig, dass die Kommunikation fehlerfrei und zeitnah erfolgt, da fehlerhafte oder verpasste Meldungen verheerende Folgen nach sich ziehen können (verpasstes Auslösen des ABS oder der Airbags, um nur zwei Beispiele zu nennen). Um diese Anforderungen zu erfüllen, wurden einige Vorkehrungen getroffen. Zum einen werden die Daten auf dem CAN-Bus als differenzielles Signal übertragen, d.h. auf einer Leitung wird das Signal normal und auf der anderen Leitung als Komplementär zur Ruhespannung übermittelt. Dadurch, dass die Differenz der Signale auf beiden Leitungen den eigentlich übertragenen Wert darstellt, wird sichergestellt, dass Gleichtaktstörungen auf dem Bus keinen Einfluss auf die Datenqualität haben (da ja die Differenz gleich bleibt). Eine weiteres Problem, dass bei einem Bussystem auftreten kann, sind Daten-Kollisionen zwischen den einzelnen Teilnehmern. Um diese zu verhindern, wird beim CAN das Carrier Sense Multiple Access/ Collision Resolution-Verfahren angewandt, bei dem die Teilnehmer zum Sendezeitpunkt feststellen, wann ihre Nachricht mit der eines anderen Teilnehmers kollidiert und den Konflikt entsprechend lösen.

<Grafik eines CAN-Frames, can2spec\_Bosch\_1991.pdf S.10>



Um eine Kollision zu erkennen, prüft jeder Teilnehmer simultan zum Senden, was für ein Signal gerade auf dem Bus anliegt. Stimmt das Bit auf dem Bus mit dem gerade übermittelten überein, ist keine Kollision aufgetreten oder es wurde von allen Teilnehmern gerade ein dominantes Bit (0) übermittelt. Sendet der Teilnehmer im Arbitration-Feld (das aus der Message-ID und einem Control-Bit besteht) ein rezessives Bit und liegt auf dem Bus ein dominantes Bit an, stellt der Teilnehmer seine Übermittlung ein und schickt die Meldung später selbständig erneut.

<grafik <http://techiscafe.blogspot.ch/p/the-controller-area-network-can.html>>

Überlegung

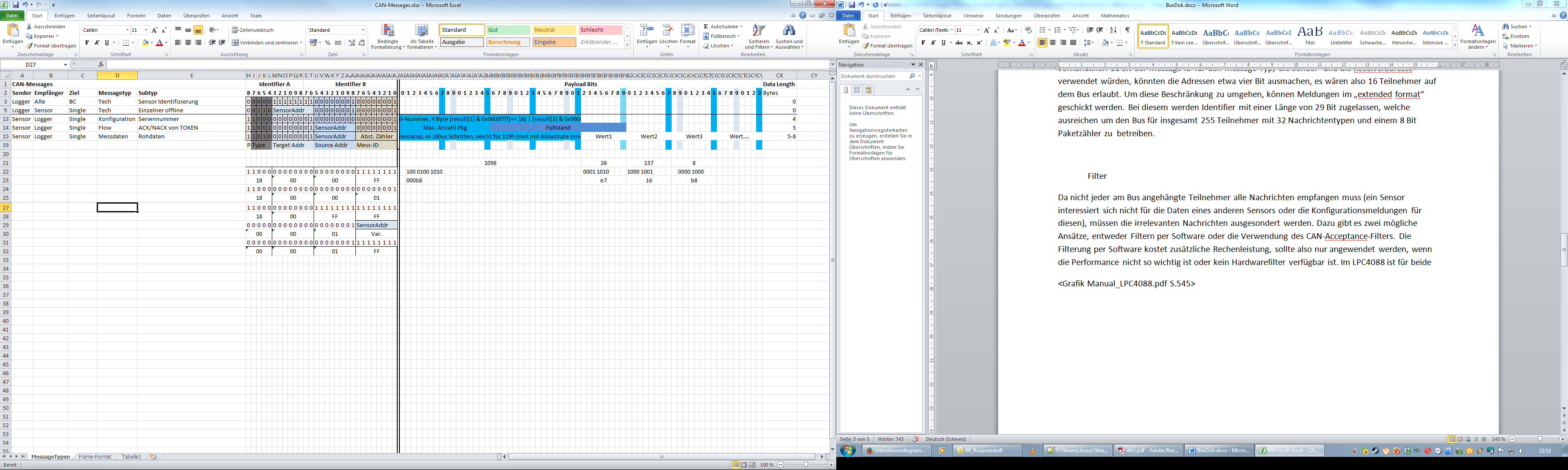
Implementation

In einem CAN-Bus wird normalerweise kein Bus-Master verwendet, da die Nachrichten nach Verwendungszweck mit IDs versehen sind und so jeder Teilnehmer herausfinden kann, welche Meldungen für ihn relevant sind (zum Beispiel interessiert sich der Auslöser des Airbags nicht für die Abgaswerte der Lambdasonde). Für dieses Bussystem ist aber neben der Art der Meldung auch noch der Absender und der Empfänger wichtig, um einerseits die direkte Kommunikation zwischen dem Datenlogger und einem Sensor und andererseits die Steuerung des Busses per Broadcast-Meldungen.

Messages

<Beispiel Message Format einfügen CAN-Messages.xlsx>

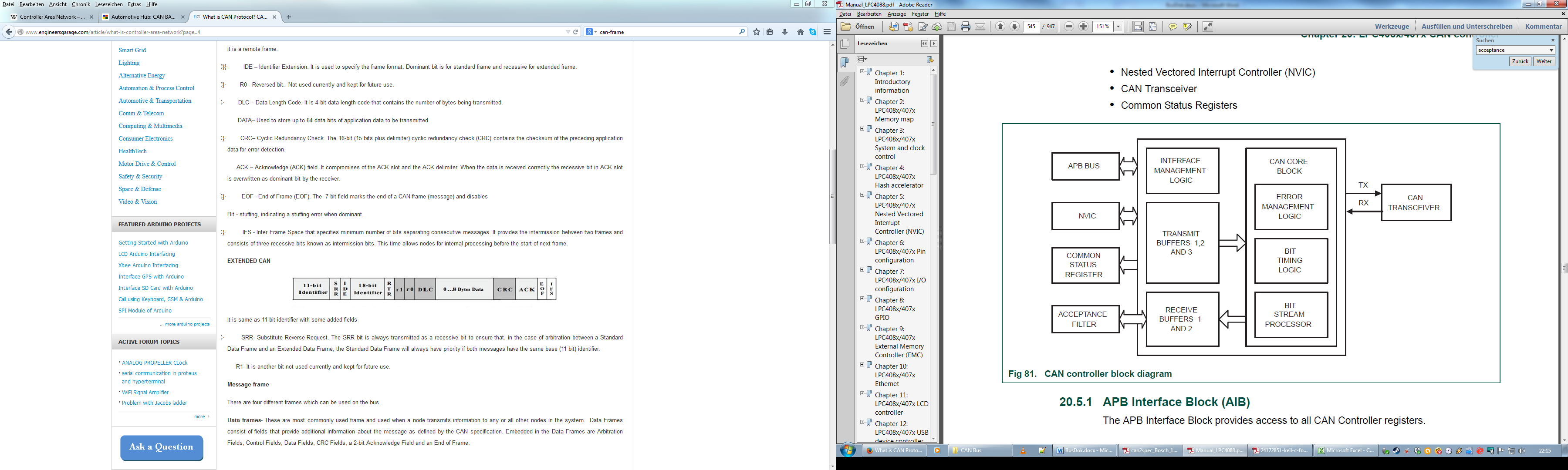
Da für den Datenlogger wichtig ist, wer welche Daten geschickt hat und die einzelnen Sensoren auch unabhängig voneinander konfiguriert werden müssen, sollte jeder Sensor eine eindeutige ID erhalten. Zusätzlich soll auch noch unterschieden werden, welcher Art die Nachrichten sind und welche Nachrichten Vorrang haben sollten, falls eine Kollision bei der Übertragung eintritt. Wenn die vorhandenen 11 Bit der Message-ID für den Message-Typ, die Sender- und die Receiveradresse verwendet würden, könnten die Adressen etwa vier Bit ausmachen, es wären also 16 Teilnehmer auf dem Bus erlaubt. Um diese Beschränkung zu umgehen, können Meldungen im „extended format“ geschickt werden. Bei diesem werden Identifier mit einer Länge von 29 Bit zugelassen, welche ausreichen um den Bus für insgesamt 255 Teilnehmer mit 32 Nachrichtentypen und einem 8 Bit Paketzähler zu betreiben.

  
Um sicherzustellen, dass der Busmaster (in diesem Fall der Datenlogger) immer Priorität vor den anderen Teilnehmern hat, wurde das erste Bit der Message-ID als Prioritätsbit verwendet und für alle Logger-Nachrichten auf 0 gesetzt. In der Konfiguration dieses Busses steht eine 0 für ein dominantes Bit, d.h. eine gleichzeitig gesendete 1 würde überschrieben und der andere Teilnehmer erkennt eine Kollision, woraufhin er die Übertragung abbricht und später erneut versucht. Die folgenden vier Bits dienen als Message-Typ (nicht zu verwechseln mit dem CAN-Message Typ, der nicht in der Message-ID vorkommt), welcher an erster Stelle ebenfalls das Prioritätsbit aufweisen. Sollten mehr als die bereits verwendeten Message-Typen notwendig werden, kann der volle Bereich ausgeschöpft werden. Die nächsten acht Bits enthalten die Empfängeradresse, darauf folgen die Senderadresse und die eigentliche Message-ID (der Paketzähler). Für eine Broadcast-Message wird die Zieladresse auf 0xFF gesetzt, ansonsten muss der gewünschte Empfänger angegeben werden (0x01 für den Busmaster und eine aufsteigende Nummerierung für jeden Teilnehmer). Als Senderadresse verwenden die noch nicht registrierten Teilnehmer eine leere Sourceadresse, die sonst die zugewiesene CAN-Id des Teilnehmers enthalten sollte. Die Reihenfolge dieser Felder wurde so gewählt, um die Möglichkeit des CAN-Acceptancefilters auszunutzen.

Filter

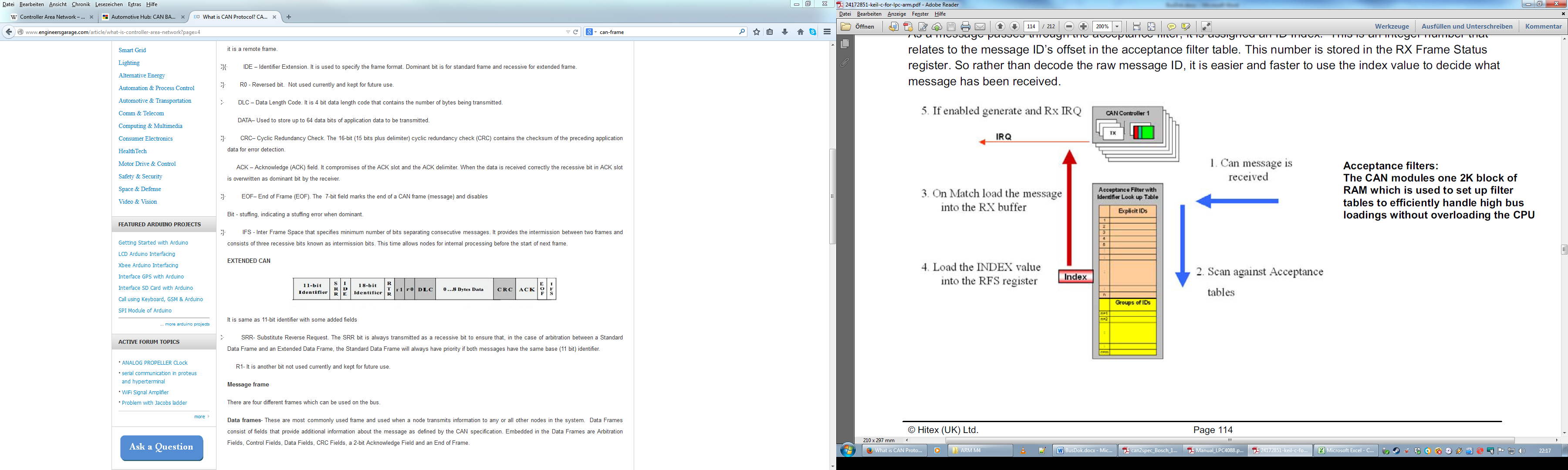
Da nicht jeder am Bus angehängte Teilnehmer alle Nachrichten empfangen muss (ein Sensor interessiert sich nicht für die Daten eines anderen Sensors oder die Konfigurationsmeldungen für diesen), müssen die irrelevanten Nachrichten ausgesondert werden. Dazu gibt es zwei mögliche Ansätze, entweder das Filtern per Software oder die Verwendung des CAN-Acceptance-Filters. Die Filterung per Software kostet zusätzliche Rechenleistung, sollte also nur angewendet werden, wenn die Performance nicht so wichtig ist oder kein Hardwarefilter verfügbar ist. Im LPC4088 ist für beide

<Grafik Manual\_LPC4088.pdf S.545>

CAN-Controller ein Hardwarefilter eingebaut, der insgesamt 1024 Standard-Identifier oder 512 Extended Identifier aufnehmen kann. Neben dem spezifischen Filtern, das nur eindeutige Treffer zulässt, kann der Filter auch so konfiguriert werden, dass ganze Bereiche von Message-IDs akzeptiert werden. Zusätzlich besteht noch die Möglichkeit, den Filter in den Full-CAN-Modus zu stellen, bei dem die empfangenen Nachrichten gleich in einem definierten Buffer zwischengespeichert werden, das Auslesen des Empfangsbuffers entfällt hier also. Da dieser Modus aber nur mit spezifischen Standard-Filtern funktioniert, wurde der hier nicht implementiert.

Der Filter besteht aus einer Lookup-Tabelle, die hierarchisch aufgebaut ist: zuerst sind die spezifischen Standard-Filter (16 Bit), dann die Gruppen Standard-Filter (2x16 Bit mit lower- und upper-Bound), dann die spezifischen Extended-Filter (32 Bit) und zuletzt die Gruppen Extended-Filter (2x32 Bit) eingetragen. Trifft nun eine CAN-Nachricht ein wird geprüft, welcher Art die erhaltene Message-ID ist. Im Anschluss wird die ID zuerst gegen die spezifischen Filter geprüft, wird kein Treffer erzielt, werden die Gruppen-Filter geprüft. Bei einem Treffer wird die Nachricht in den Receive-Buffer geladen, der CAN-Controller löst einen Interrupt aus und die ganze Meldung kann ausgelesen werden.

< 24172851-keil-c-for-lpc-arm.pdf S. 114)



Im vorliegenden Bussystem wurden die Filter des Loggers so konfiguriert, dass alle Sensor-Nachrichten akzeptiert werden. Bei den Sensoren wird die Konfiguration des Filters in drei Schritten vorgenommen: nach dem Reset eines Sensors wird der Filter so gesetzt, dass nur eine Seriennummer-Anfrage des Loggers durchgelassen wird. Auf diese Anfrage antworten die Sensoren mit dem versenden ihrer Seriennummer und dem Setzen der Broadcast-Filter. Nun können die Sensoren alle Broadcast-Meldungen des Loggers empfangen. Der Logger verschickt nun die zugeteilten CAN-Ids zusammen mit den entsprechenden Seriennummern als Broadcast. Empfängt ein Sensor den Broadcast, prüft er die Seriennummer und, falls diese mit seiner übereinstimmt, setzt die Filter für alle an ihn gerichteten Meldungen.