Local Interconnect Network

PG 522: AutoLab
Eine Experimentierplattform für automotive Softwareentwicklung

Universität Dortmund

WS 2007/2008 und SS 2008

Matthias Meier

2. November 2007

Inhaltsverzeichnis

1 Einführung

LIN (Local Interconnect Network) wird seit 1998 von einem Konsortium entwickelt. Federführend sind in diesem Konsortium Freescale (ehem. Motorola), Audi, BMW, Daimler, VW, Volvo und Mentor Graphics beteiligt. Der LIN-Bus dient als Subbus zur kostengünstigen Vernetzung von Steuergeräten im Auto. Einsatz findet LIN im Bereich der einfachen Sensor und Aktor Anwendungen. Für diesen kostensensiblen Bereich kam der CAN-Bus nicht in Frage, da die Bandbreite und Vielseitigkeit hier nicht benötigt wird und auch die Kosten für die Vernetzung mit diesem Bus zu hoch sind. Bevor das LIN Konsortium gegründet wurde, gab es bereits einige Bemühungen der Automobilhersteller eigene Lösungen im Bereich der kostensensiblen Subbusse zu entwickeln, um die wachsende Anzahl von elektronischen Funktionen im Auto kostengünstig zu vernetzen. Jedoch verschwanden die meisten Entwicklungen wieder. LIN hat sich heute als Subbus für einfache Sensor und Aktor Anwendungen in der Automobilindustrie etabliert und wird von vielen Automobilherstellern eingesetzt.
Um die Kosten für die einzelnen Steuergeräte niedrig zu halten, basiert LIN auf den weit verbreiteten UART-Interfaces. Dies ist eine serielle Schnittstelle, die bereits in den meisten Mikrocontrollern integriert ist. Des weiteren konnten Kosten dadurch eingespart werden, dass für die Slaves im LIN-Netzwerk bzw. LIN-Cluster keine Quarze oder Keramikresonatoren benötigt werden. In einem LIN-Cluster können bis zu 16 Busteilnehmer vernetzt werden, wobei die maximale Datenrate 20 kbit/s beträgt.
Der LIN-Bus wird für Sicherheitsunkritische Komponenten im Auto eingesetzt. In den Anwendungsbereich fallen Komfortfunktionen, wie zum Beispiel Fensterheber, Zentralverriegelung, elektrische Spiegelverstellung, elektrische Sitzverstellung, Regensensor, Lichtsensor, elektrisches Schiebedach oder Steuerung für Klimaanlage.

2 Physical Layer

LIN benutzt einen 12V Ein-Draht-Bus basierend auf dem ISO 9141 Standard. Die Datenrate wurde für den LIN-Bus auf 20 kbit/s beschränkt und die Flankensteilheit wurde angepasst. Dies wurde zum einen gemacht, damit die Taktsynchronisation in den Slaves noch gut handhabbar ist, da die Slaves keinen exakten Taktgeber besitzen und nachsynchronisiert werden müssen und zum anderen wurde mit den Maßnahmen die elektromagnetische Verträglichkeit verbessert.
Die Bits werden relativ zur Versorgungsspannung übertragen. Liegen zwischen 60 und 100 % der Versorgungsspannung auf dem Bus an, wird der rezessive Zustand übertragen, was einer logischen Eins entspricht. Bei einer Versorgungsspannung zwischen 0 und 40 % wird der dominante Zustand auf dem Bus übertragen, was einer logischen Null entspricht. Wenn keine Aktivitäten auf dem Bus stattfinden ist der rezessive Zustand auf dem Bus.

3 Data Link Layer

3.1 Konzept der Kommunikation

Ein LIN-Cluster ist nach dem Single-Master / Multiple-Slave Konzept aufgebaut. Das bedeutet, dass in dem Bus ein Knoten als Master arbeitet und alle weiteren als Slave. Alle Knoten führen einen Slave-Task aus, der die Kommunikation auf dem Bus mitliest und notwendige Daten überträgt. Der Master-Knoten hat zusätzlich noch einen Master-Task. Der Master-Task initialisiert und steuert dabei die gesamte Kommunikation im LIN-Cluster.

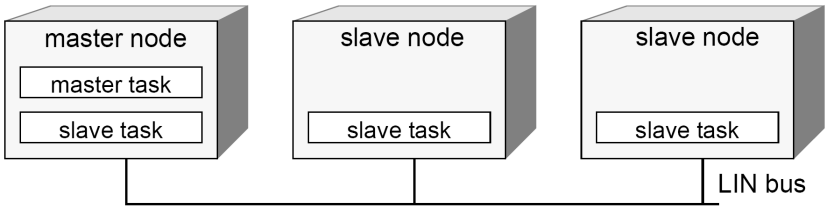

Abbildung 1: LIN-Cluster

Dazu überträgt der Master-Task einen Frame-Header auf den Bus. Dieser Header wird von allen Slaves gelesen und ausgewertet. Falls ein Slave die vom Master geforderten Informationen bereitstellen kann, sendet der Slave die Daten als Frame-Response direkt hinter dem Header über den Bus. Der Master dient zusätzlich meist als Gateway zum in der Hierarchie höheren Netzwerk, zum Beispiel dem CAN-Bus. Der Nachteil dieses Konzeptes ist, dass im Falle eines Ausfalls des Masters das gesamte LIN-Cluster ausfällt und auf dem Bus keine Kommunikation mehr stattfinden kann.

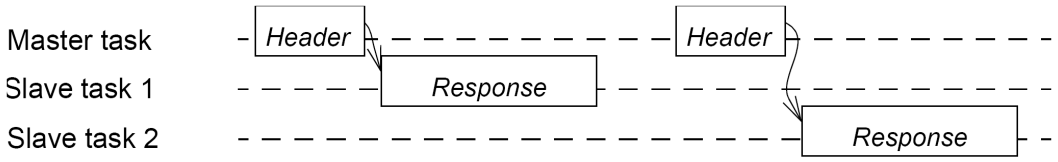

Abbildung 2: Prinzip der Kommunikation

3.2 Frame

Ein Frame setzt sich aus einem Frame-Header und einer Frame-Response zusammen. Der Header besteht aus den Feldern “Break Field”, “Sync Field” und “Protected Identifier Field”. Das Break Field dient zur Signalisierung eines neuen Frames. Es besteht aus mindestens 13 dominanten Bits und endet mit einem bis vier rezessiven Bits (Break Delimiter). Diese Werte sind so gewählt, dass jeder Slave-Knoten den neuen Frame erkennt, obwohl diese keine Quarze oder Keramikresonatoren besitzen. Im Anschluss an

das Break Field wird das Sync Field übertragen. Das Sync Field dient zur Synchronisierung der Slaves mit dem Master und besteht aus einer alternierenden Folge von dominanten und rezessiven Bits. Das letzte Feld im Header ist das Protected Identifier Field. Dieses Feld adressiert die vom Master gewünschte Nachricht und nicht direkt den Knoten der auf den Header antworten soll (inhaltsbezogene Adressierung). Der Vorteil der inhaltsbezogenen Adressierung ist, dass zu mehreren Knoten gesendet werden kann, ohne dass die Adressen der einzelnen Knoten verwaltet werden müssen. Das Protected Identifier Field setzt sich aus zwei Subfeldern zusammen. Zum einem aus dem Identifier für die Frames, die ersten sechs Bits, und zum anderen aus zwei Paritätsbits. Insgesamt stehen somit 64 verschiedene Identifier für die Frames zur Verfügung, wobei nur die Identifier 0 bis 59 für die Übertragung von Signalen benutzt werden. Die Identifier 60 und 61 werden für Diagnose- und Konfigurationsdaten benutzt. Die letzten beiden Identifier sind für zukünftige Anwendungen reserviert. Die Paritätsbits dienen zur Erkennung von Bitfehlern im Identifier und berechnen sich folgendermaßen:

$\text{P0} = \text{ID0} \oplus ID1 \oplus ID2 \oplus ID4$

$\text{P1} = \neg(ID1 \oplus ID3 \oplus ID4 \oplus ID5)$

Die Frame-Response besteht aus dem Daten Bereich und aus der Checksum. Der Datenbereich ist von variabler Länge und kann bis zu acht Datenbytes enthalten. Für die Berechnung der Checksum gibt es zwei verschiedene Verfahren, zum einen die Classic Checksum und zum anderen die Enhanced Checksum. Für Slaves mit der LIN Version 1.x wird die klassische und für Slaves mit der Version 2.x wird die erweiterte Variante benutzt. Für jeden Frame wird festgelegt, welche Version der Checksum benutzt wird, je nachdem welche Slaves an der Kommunikation von diesem Frame beteiligt sind. Die klassische Checksum berechnet sich aus der Addition der einzelnen Datenbytes. Falls bei der Addition zweier Datenbytes ein Übertrag aufgetreten ist, wird zusätzlich eine Eins aufaddiert. Anschließend wird das Ergebnis invertiert. Für die erweiterte Checksum wird zu den einzelnen Datenbytes auch noch das Protected Identifier Field des Frames addiert.

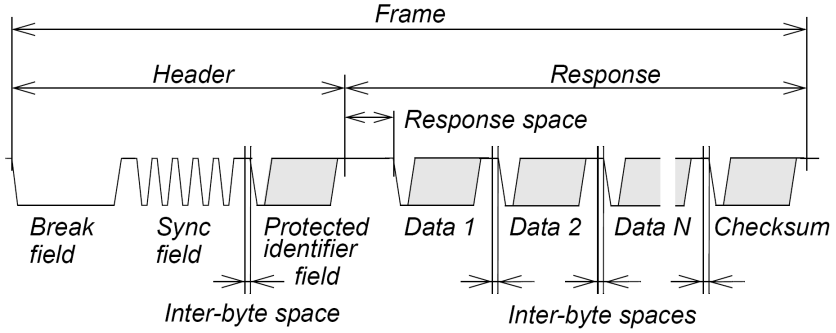

Abbildung 3: Aufbau eines Frames

3.3 Scheduling-Tabelle

Der Master richtet sich zur Steuerung der Kommunikation im LIN-Cluster nach einem Zeitplan, der Scheduling-Tabelle. In der Scheduling-Tabelle ist zum einen die Reihenfolge festgelegt, in der der Master die Frame-Header auf den Bus legt und zum anderen die Zeit, die für die Übertragung eines Frames gegeben wird. Falls das Ende einer Schedulling-Tabelle erreicht wird, wird die Tabelle vom Master wieder von Neuem abgearbeitet. Es können mehrere Scheduling-Tabellen festgelegt werden, zwischen denen umgeschaltet werden kann. So kann sich das LIN-Cluster verschiedenen Bedingungen anpassen und zum Beispiel die Reaktionszeit einzelner Knoten in einer Tabelle verbessern, in dem dieser in einem Scheduling-Zyklus mehrfach abgefragt wird. Die Scheduling-Tabellen und die Tatsache, dass die Kommunikation von einer zentralen Instanz, dem Master, initiiert und gesteuert wird, macht die Kommunikation im LIN-Cluster vorhersagbar.

3.4 Frame-Typen

3.4.1 Unconditional Frames

Die Unconditional Frames sind die "standard" Frames im LIN-Cluster und werden ständig periodisch gesendet. Die Frame-Response wird dabei immer von dem gleichen Slave übertragen und der Slave muss mit seiner Frame-Response auf den Frame-Header antworten.
Abbildung 4 zeigt drei verschiedene Szenarien. Die Datenübertragung wird dabei immer vom Master initialisiert. Im ersten Fall werden Daten von Slave 1 zum Master übertragen. Im zweiten Fall überträgt der Master Daten, die für beide Slaves relevant sind. Im letzten Szenario ist der Datenaustausch zwischen zwei Slaves gezeigt. Die Initialisierung des Datenaustausches ist aber wie immer vom Master durch einen Header gestartet.

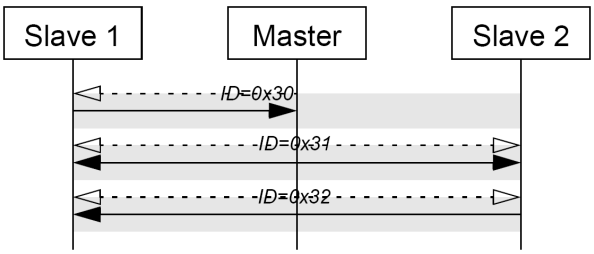

Abbildung 4: 3 Szenarien für die Kommunikation mit Unconditional Frames

3.4.2 Event Triggered Frames

Mit Event Triggered Frames können mehrere Knoten durch einen Frame abgefragt werden. Damit nicht grundsätzlich Kollisionen auftreten, antworten die angesprochenen

Knoten nur, wenn tatsächlich intern neue Daten vorliegen. Falls keiner der Knoten Daten übertragen möchte, bleibt der Bus nach Übertragung des Frame-Headers frei. Wenn mehr als ein Knoten mit seiner Frame-Response antwortet, findet eine Kollision statt. Falls die sendenden Knoten eine Kollision feststellen, brechen sie die Übertragung ab und der Master wechselt zu einer kollisionslösenden Schedulling-Tabelle. Jeder Event Triggered Frame hat eine eigene kollisionslösende Tabelle. Die jeweilige kollisionslösende Tabelle wird einmalig abgearbeitet und anschließend wechselt der Master zur vorherigen Schedulling-Tabelle zurück.
Der Vorteil von diesem Frametyp ist, dass die Reaktionszeit auf einzelne Events gesteigert werden kann, ohne viel Bandbreite des Busses opfern zu müssen. Eine mögliche Anwendung wäre die Abfrage aller Türpins bei einem viertürigen Auto.

3.4.3 Sporadic Frames

Ein Sporadic Frame setzt sich aus einer Gruppe von Unconditional Frames zusammen, die sich einen Zeitschlitz teilen. Wenn ein Sporadic Frame an der Reihe ist, kontrolliert der Master, ob sich für einen der Unconditional Frame aus dem Sporadic Frame Daten geändert haben, bzw. ob er Daten benötigt und sendet dann in dem Zeitschlitz den entsprechenden Unconditional Frame. Wenn keine Daten übertragen werden müssen, sendet der Master keinen Frame-Header und während des Zeitschlitzes bleibt der Bus frei. Falls mehrere Unconditional Frames in einem Zeitschlitz übertragen werden müssten, wird der Unconditional Frame mit der höchsten Priorität gewählt. Die Frames, die nicht übertragen wurden gehen nicht verloren, sondern sind Kandidaten für die nächsten Vorkommen dieses Sporadic Frame.
Der Zweck der Sporadic Frames ist es, dynamisches Verhalten in die Scheduling-Tabelle zu bekommen, ohne den Determinismus im Rest der Tabelle zu verlieren.

3.5 Netzwerk Management

Das Netzwerkmanagement beschreibt hauptsächlich den Übergang vom Betriebszustand in den Schlafmodus. Damit die Knoten in den Schlafmodus wechseln, gibt es zwei Möglichkeiten. Einerseits wechseln die Knoten nach vier bis zehn Sekunden Inaktivität auf dem Bus selbstständig in den Schlafmodus und andererseits kann der Master das ganze LIN-Cluster mit einem Kommando in den Schlafmodus versetzen. Dazu sendet der Master einen Frame mit dem Identifier 0x3C, bei dem das erste Datenbyte auf Null gesetzt ist und die Datenfelder zwei bis acht mit Einsen aufgefüllt werden.
Eine Weckaufforderung kann jeder Knoten im LIN-Cluster stellen. Der Knoten sendet dazu für 250μs bis 5ms den dominanten Zustand. Die anderen Knoten müssen dann nach 100ms fertig initialisiert sein und der Master sendet einen Frame aus um die Weckursache zu ermitteln.

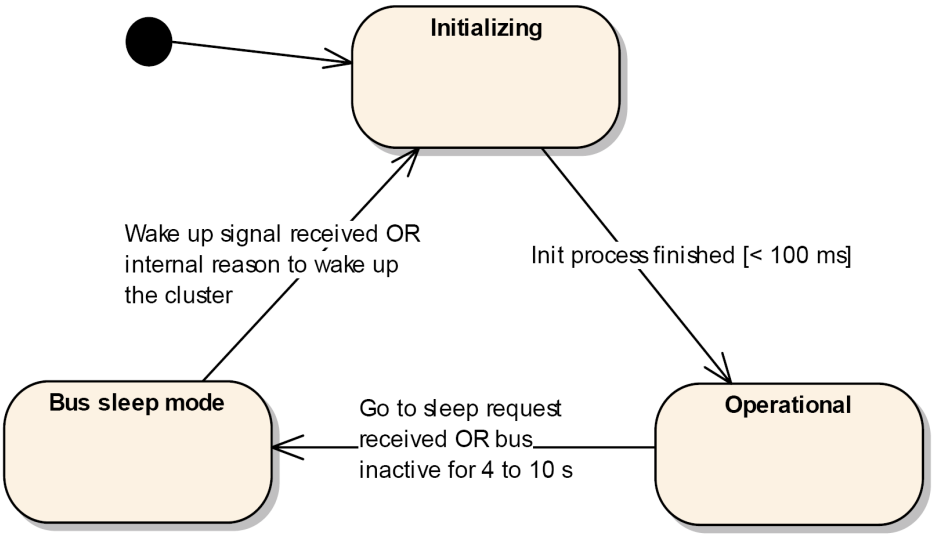

Abbildung 5: Zustandsdiagramm für die Knoten

3.6 Fehlererkennung & -behandlung

Bei LIN gibt es nur eine rudimentäre Fehlererkennung. Die Behandlung der Fehler ist in LIN nicht weiter spezifiziert und findet nur auf der Anwendungsebene statt.
Zur Erkennung von Bitfehlern auf der Busleitung lesen die Sender ihr Signal mit und brechen bei einem festgestellten Fehler die Übertragung ab. Des weiteren bieten die Paritätsbits einen leichten Schutz vor einem falschen Identifier und mit der Checksum lassen sich Fehler im Datenbereich der Frame-Response erkennen. Der Master erkennt zusätzlich wenn er auf seinen Frame-Header keine Antwort bekommt oder wenn bei der Datenübertragung eine Zeitüberschreitung aufgetreten ist. Alle Slaves im LIN-Cluster müssen den Master mit Hilfe des Response Error Bits über mögliche Fehler unterrichten. Dazu benutzen die Slaves einen ohnehin schon verwendeten Unconditional Frame und übertragen das Response Error Bit wie andere Signale im Datenbereich der Frame-Response. Der Master sammelt alle Fehlerinformationen und entsprechend der Fehler wird in den Anwendungen des Masters reagiert.

4 LIN Configuration Language

Die LIN Configuration Language ist eine Sprache mit der eine LIN Description File (LDF) erstellt werden kann. In der LDF wird ein komplettes LIN-Cluster beschrieben und konfiguriert. Mit Hilfe von Werkzeugen kann aus der LDF automatisch C-Code für die Steuergeräte erzeugt werden.

4.1 Aufbau der LDF

Zu Beginn der LDF werden grundlegende Parameter festgelegt. Zum Beispiel die Versionsnummer oder die Übertragungsgeschwindigkeit. Anschließend ist die LDF in verschiedene Blöcke gegliedert. Die grundlegenden Blöcke mit den wichtigsten Funktionen sind im folgenden kurz beschrieben.

4.1.1 Definition der Knoten

In diesem Abschnitt werden symbolische Namen für die einzelnen Knoten definiert. Für den Master wird zusätzlich noch die Zeitbasis und der Jitter angegeben.

Beispiel:

```
Nodes {
  Master: DCU, 5 ms, 0.1 ms;
  Slaves: FLWM, LMM, CPM, CEM, ...;
}
```

4.1.2 Definition der Signale

Signale enthalten die eigentlich zu übertragenden Daten. Im Datenbereich eines Frames können mehrere Signale übertragen werden. Ein Signal wird folgendermaßen in der LDF beschrieben. Zuerst bekommt jedes Signal einen symbolischen Namen. Der erste Wert beschreibt die benötigten Bits im Datenbereich des Frames für die Übertragung des Signals. Der zweite Wert ist der Initialisierungswert für das Signal. Anschließend folgt eine Liste von Knoten, wobei der erste der Sender ist und alle weiteren die Empfänger.

Beispiel:

```
Signals {
  WindowButtons: 8, 0, DCU, FLWM;
  MirrorButtonsStatus: 2, 0, DCU, LMM;
  MirrorButtons: 4, 0, DCU, LMM;
  ...
  WaterTempLow: 8, 0, CPM, CEM;
  WaterTempHigh: 8, 0, CPM, CEM;
  CPMFuelPump: 7, 0, CPM, CEM;
  ...
}
```

4.1.3 Definition der Frames

In diesem Block werden die Frames beschrieben. Zuerst bekommt wieder jeder Frame einen symbolischen Namen. Dann wird der Identifier des Frames, der Sender der Frame-Response und die Länge des Datenbereichs in Bytes angegeben. Dann werden in dem

Frame-Block die zu übertragenden Signale festgelegt, wobei für jedes Signal noch das Anfangsbit im Datenbereich bestimmt wird.

Beispiel:

```
Frames {
  DCU_Frm1: 0x01, DCU, 1 {
    MirrorButtonsStatus, 0;
    MirrorButtons, 2;
  }
  CPM_Frm1: 0x02, CPM, 5 {
    WaterTempLow, 0;
    WaterTempHigh, 8;
    CPMFuelPump, 20;
  }
  ...
}
```

4.1.4 Definition der Scheduling-Tabellen

Zunächst werden für alle Scheduling-Tabellen symbolische Namen festgelegt. In den Blöcken der einzelnen Tabellen werden dann die Frames mit dem für die Übertragung festgelegten Timing eingetragen.

Beispiel:

```
Schedule_tables {
  Normal_Schedule {
    DCU_Frm1 delay 15 ms;
    CPM_Frm1 delay 25 ms;
  }
  ...
}
```

Literatur

[1] *LIN Specification Package. Rev. 2.1.* November 2006. - http://lin-subbus.org/

[2] Zimmermann, Werner ; Schmidgall, Ralf: *Bussysteme in der Fahrzeugtechnik; Protokolle und Standards.* 2. Auflage. Wiesbaden : Vieweg, 2007

[3] *LIN-Spezifikation.* August 2007. - http://vector-worldwide.com/vi_lin_de.html

[4] *LIN.* - http://www.hto.fh-deggendorf.de/komm/automotive/technik/techniklin.html

[5] Seiler, Remo: *LIN-Bus.* Januar 2006. - http://prof.hti.bfh.ch/uploads/media/LIN-Bus.pdf

[6] Form, Prof. Dr.-Ing. Thomas: *Datenbussysteme in Straßenfahrzeugen.* Dezember 2006. - http://www.ifr.ing.tu-bs.de/lehre/downloads/skripte/skript_dbf_31.pdf