

Technische Spezifikationen MRBus

Version, Datum	erstellt	Bemerkung
1.0, 07.11.07	Michael Prader	Erstausgabe
1.1, 15.11.07	Michael Prader	neu Pakettyt ,X' für approach-lighting bei Signalen; Nummerierung von Bytes in Datenpaketen expliziert;

<i>Abschnitt 1.0 - Einleitung</i>	2
<i>Abschnitt 2.0 – Designgrundsätze</i>	2
<i>Abschnitt 2.1 - MRBus Physischer Layer</i>	2
<i>Abschnitt 3.0 – Adressierungsschema</i>	3
<i>Abschnitt 3.1 – Arbitrierung, Paketübertragung und Wiederholung</i>	4
<i>Abschnitt 3.1.1 - Spezifizierung Datenpakete R, W, r, w:</i>	6

Literatur:

MRBus specification – Nathan D. Holmes;

<http://www.ndholmes.com/pmwiki.php/Electronics/MRBusSpec>

Abschnitt 1.0 - Einleitung

Die Einleitung beschreibt den Werdegang des MRBus und wird ausgelassen. Interessierte können sie auf Nathans Seite nachlesen.

Abschnitt 2.0 – Designgrundsätze

Das Design von MRBus ging den umgekehrten Weg der normalen Prozedur – zuerst wurde die Hardware gewählt, dann wurden die Spezifikationen für Software und Datenelemente festgelegt. Dies wurde aus dem Grund gewählt, dass Kosten für den normalen Modellbahner ein signifikantes Kriterium sind. Ein langsames, aber funktionierendes und billiges System ist einem schnellen, teureren System vorzuziehen. Weiters sind potentiell große Systeme finanzierbar.

Das Netzwerk soll annähernd mittels des OSI-Netzwerk-Modells beschrieben werden. MRBus wird in Physischer Layer, Data Link und Transport beschrieben. Alles darüber ist dem jeweiligen Entwickler der Implementation überlassen.

Abschnitt 2.1 - MRBus Physischer Layer

Da RS485 nur den Hardware-Teil des Multi-drop-Busses spezifiziert, musste ein Protokoll für Arbitrierung und Kollisionserkennung/-verhinderung entwickelt werden. MRBus basiert auf einem früheren Telemetrie-Netzwerk-Konzept mit dem Namen RDB – Revised Differential Bus. Das Konzept war ursprünglich 1997 von Mark Finn, Michael Petersen entwickelt worden, und funktionierte auf der Basis eines „open collector“-Netzes. Null wird übermittelt, indem aktiver der Bus auf low gezogen wird, während 1 durch Deaktivierung des Transmitter für eine Bitlänge übertragen wird, wodurch der Bus durch die Pull-up-Widerstände auf High geht. Das ist eine rohe Form der Kollisionserkennung – jedes Gerät, welches eine 1 schreibt und eine 0 liest, ist in Kollision und muss sofort das Senden beenden. So lange dies schnell genug passiert, sollte das 0 sendenden Gerät nichts davon merken. Kollisionen werden also nicht nur erkannt, sondern auch auf nicht-destruktive Weise behandelt. Weiters ist das erste gesendete Byte immer die Geräteadresse. Solange jedes Gerät eine einzigartige Adresse besitzt, wird die Priorität schon vor dem Beginn der Datenbytes geregelt. Zusammen mit einer gewissen Zufälligkeit bei den Übertragungen sowie einem Rückzug-Timing-System ergab dies ein multi-access-Netzwerk mit 2400 baud.

MRBus trägt dieses Konzept weiter und optimiert es, um Datenraten auf ein akzeptierbares Maß für ein Echtzeit-Netz zu erhöhen. Das Basiskonzept des semi-passiven Busses wurde für die Arbitrierung beibehalten, aber sobald das Gerät die Kontrolle über den Bus übernommen hat, wird der Bus aktiv in beide Richtungen (1 und 0) gesteuert, wodurch eine höhere Datenrate möglich ist. Um alle 256 Geräteadressen per MRBus-Segment zu nutzen, müssen 1/8-unit load RS485 Schnittstellenbausteine benutzt werden, wie den MAX3082E. Obwohl nicht erforderlich, sind slew-rate-limitierte Bausteine zu bevorzugen, um hohe Frequenzen auf dem Bus zu minimieren, und Terminierungs- und Reflexionsprobleme zu vermeiden. Für kleinere Systeme mit einer geringeren Anzahl von Geräten (max 128) können auch 1/4-unit load ICs verwendet werden.

Jedes Bussegment sollte aus nicht mehr als 540 Meter UTP-Kabel, mit einer charakteristischen Impedanz von 125 Ohm, einem Widerstand von nicht größer als 200 Ohm per 1000 Fuß, und einer Kapazität nicht größer als 17pF/Fuß bestehen. Cat5e-Kabel erfüllt diese Spezifikationen. Der Bus kann jede Form annehmen, mit Ausnahme von Schleifen, da die niedrige Bitrate und das differentielle Signal die meisten Probleme mit Reflexion und Interferenz umgehen sollten. Jegliche Verzweigung ist erlaubt, aber kein Ring/Schleife.

Jedes Bussegment muss auch je zwei 2kOhm-Widerstände besitzen, welche den Bus in Ruhe auf 1 ziehen (A muss auf +5 V gezogen werden, B auf GND). Diese sollten bei $\frac{1}{4}$ und $\frac{3}{4}$ der Länge eingebaut werden, bei kurzen Segmenten sollte es aber egal sein.

MRBus benutzt folgendes Verkabelungsschema:

RJ45, folgendes Pinout:

- 1 - +9VDC (Weiß/Orange)
- 2 - GND (Orange)
- 3 - +9VDC (Weiß/Grün)
- 4 - MRBus:RS485-A (Blau)
- 5 - MRBus:RS485-B (Weiß/Blau)
- 6 - GND (Grün)
- 7 - DCC:RS485-A (Weiß/Braun)
- 8 - DCC:RS485-B (Braun)

Alle Geräte sollten mindestens zwei Anschlüsse zwecks Weiterführung besitzen. Mehr als zwei sind natürlich möglich, da MRBus eine Freiform-Architektur besitzt. Dieses eher bizarre Pinout wurde gewählt, um zwei Versorgungspaare und eine differentielles Datenpaar zu haben, wobei ein Crossover-Kabel nicht zur Zerstörung angeschlossener Geräte führt; allerdings ist MRBus für normale Netzkabel gedacht.

Pins 7-8 können für eine Verteilung des DCC-Signals zu Boostern entlang der Anlage genutzt werden. Dies wird vorerst nicht genutzt.

Abschnitt 3.0 – Adressierungsschema

Adressierung und mehrere Netzwerke

Ein Maximum von 256 Geräte ist auf einem einzelnen Segment erlaubt, wegen der Limitierungen der Transceiver. Falls andere als 1/8-unit load RS485-Transceiver genutzt werden, ändert sich die Anzahl der maximalen Geräte dementsprechend ($\frac{1}{4}$ -> 128, $\frac{1}{2}$ -> 64, usw.). Zieladresse 0 wird von allen ignoriert (kein Ziel), und 255 ist die Broadcast-Adresse (Ziel sind alle Geräte). Jedes Gerät, welches senden kann, muss eine einzigartige Adresse von 1 bis 254 besitzen. Geräte, welche nur empfangen können, sollten Adresse 0 zugewiesen bekommen, außer eine Adresse 1-254 ist spezifisch notwendig.

Für Netzwerke mit mehr als 256 Knoten ist eine Brücke notwendig, welche entwickelt werden müsste.

Abschnitt 3.1 – Arbitrierung, Paketübertragung und Wiederholung

Übertragungen muss eine gewisse Wartezeit vorangehen, während der der Bus auf Aktivität überwacht wird. Diese Wartezeit besteht aus einer fixen und einer variablen Komponente.

Die fixe Wartezeit ist leicht größer als zwei 4800bps-Bitlängen – 440 μsec . Der Bus wird alle 10 μsec auf Aktivität überprüft. Wird eine solche festgestellt, muss der Zyklus abgebrochen und neu gestartet werden.

Die variable Wartezeit ist etwas komplexer, und besteht aus drei Komponenten: Priorität, Einsamkeit und den unteren vier Bit der Geräteadresse. Priorität legt die Wichtigkeit des Pakets fest, und besteht im Bereich 0-12. Je kleiner die Zahl, desto höher die Priorität. 6 sollte an fast alle Pakete vergeben werden. Wichtige oder zeitsensitive Daten wie CTC-Befehle oder Fehlermeldungen sollten mit einer höheren Priorität behandelt werden. Nur Echtzeit-Pakete (Uhr-Synchronisierung) sollten eine Priorität=0 erhalten. Generelle Telemetrie-Daten sollten eine geringere Priorität haben (höhere Zahlen).

Einsamkeit ist ein schwammiges Konzept – sie stellt dar, wie sehr ein Geräte sich „mitteilen“ will. Je öfter eine Übertragung unterbrochen wird (Aktivität, Kollision), desto höher die Einsamkeit. Wie bei Priorität, bedeutet eine große Zahl eine kleine Einsamkeit, eine kleine Zahl eine hohe Einsamkeit. Nach jeder Übertragungsunterbrechung wird die Einsamkeit um 1 gegen 0 verringert. Nach erfolgter Datenübertragung sollte sie auf 6 zurückgesetzt werden. Ausnahme sind Geräte, welche öfter als 1 Mal pro Sekunde senden. Sie sollten einen höheren Wert zugewiesen bekommen.

Die variable Wartezeit wird wie folgt berechnet:

P = Prioritätszahl (0-12, nominal 6)

L = Einsamkeitszahl (0-12, nominal 6)

D = Geräteadresse (0-255)

Tv = Variable Wartezeit

$$T_v = (P + L + (0x0F \& D)) * 10\mu\text{sec}$$

Wie bei der fixen Wartezeit muss der Bus alle 10 μsec auf Aktivität überprüft werden; bei Erkennung einer solchen muss der Übertragungszyklus abgebrochen werden.

Jede Übertragung startet nach der Wartezeit mit einem Arbitrierungsbyte, in semi-passivem Modus gesendet. Das Arbitrierungsbyte ist die Geräteadresse mit 4800 bps, die Nibble mit 0 getrennt – effektiv ein 9-N-1-Arrangement. Als Beispiel:

Ein Gerät mit der Adresse 0xFF sendet (Start-Stop-Bits inklusive, links nach rechts) 0 1111 0 1111 1 als Arbitrierungsbyte. 0 wird durch aktive Steuerung des Busses gesendet, während 1 durch eine Deaktivierung des Transmitters erzeugt wird (für eine Bitlänge von 208 μsec). Der Bus sollte während dieser Zeit auf Kollision überwacht werden. Jedes Gerät, welches eine solche entdeckt, muss die Übertragung abbrechen und den Zyklus neu starten.

Die effektive Datenübertragung geschieht mit 57600 bps nach der erfolgreichen Arbitrierung, und sendet bis zu 20 Bytes in 8-N-1 mittels vollgesteuertem RS485. Ein Low-Bit mit 4800 baud erzeugt einen Framing-Fehler bei 56kbaud. 4800baud hat eine Bitlänge von 208 μ sec, und damit eine 20%-Sicherheitsmarge zu den 173 μ sec der 56k-Bytelänge (bei 8-N-1).

Oberhalb des physischen Layers, sind die Datenpakete wie folgt strukturiert:

1. Arbitrierungsbyte (Geräteadresse mit 4800 bps/9-N-1)
2. Geräteadresse (dieses und alle weiteren Bytes mit 57600 bps)
3. Zieladresse
4. Paketlänge (Anzahl der mit 57600 gesendeten Bytes, inklusive 2-4)
5. CRC16, niederwertiges Byte
6. CRC16, höherwertiges Byte
7. Pakettyp
8. 0-13 Datenbytes

Byte 1 wird zur Erhöhung der Lesbarkeit nicht explizit geschrieben, da es ein Linienprotokoll-Byte ist. Alle weiteren Paketbeschreibungen werden so erfolgen:

[SRC]	[DEST]	[LEN]	[CRC16 (2 bytes)]	[TYPE]	[DATA]	
0	1	2	3 4	5	6	(Nummerierung)

z.B.: [0x11][0xFF][0x08][CRC16]['D'][0x00][0x01]

Man beachte, dass die Datenpakete ab Null nummeriert werden (C-Indizierung).

Wenn und nur wenn ein Übertragungszyklus unterbrochen wird (entweder durch Aktivität während der Wartezeit oder durch eine Kollision in der Arbitrierungsphase), muss das Gerät warten bis ein Paket korrekt empfangen wurde oder 10ms, je was kürzer ist.

Die Kommunikation wird nur als mittelmäßig robust angenommen. Geräte sollten ein Paket speichern und verarbeiten können, währen ein weiteres empfangen wird. Dies gibt eine ausreichende Sicherheit, dass empfangene Pakete bearbeitet werden und keine Überläufe auftreten. Gegen Paketfolge oder Pakete länger als 20 gibt es keine Absicherungen. Jegliche solche Notwendigkeiten sollten im applikationsspezifischen Layer behandelt werden.

Durch Konvention (und nur durch Konvention) ist Byte 7 ein großer ASCII-Buchstabe für Befehle und Informations-Broadcast, und ein kleines ASCII-Zeichen eine entsprechende Antwort. Die folgenden Buchstaben sind definiert:

- A Generisches Ping – angesprochene Geräte sollen mit einem datenlosen 'a'-Paket antworten
- C, B, E, F Stellbefehle – stellt Weichen oder Signale
- D Data regular – regelmäßige Datenupdates, z.B. Blockdetektoren
- R EEPROM Read – liest eine bestimmte Stelle im EEPROM aus, antwortet mit 'r'
- S Signal – Informationen über den Zustand von sicherungsrelevanten Geräten
- W EEPROM Write – schreibt einen Wert an eine bestimmte Stelle im EEPROM, antwortet mit 'w'
- X Aktivierung/Deaktivierung Approach-lighting von Signalen

Die 16-Bit-CRC-Summe wird berechnet mit einem Startwert von 0xFFFF und einem Polynom $x^{16} + x^{15} + x^2 + x^0$. CRC startet bei der Geräteadresse und läuft über alle Paketbytes, außer den beiden CRC-Bytes.

Abschnitt 3.1.1 - Spezifizierung Datenpakete R, W, r, w:

Read 'R':

[SRC] [DEST] [LEN=07] [CRC 2] [TYPE='R'] [CV]

CV ist die EEPROM-Adresse, von welcher gelesen wird.

Read 'r':

[SRC] [DEST] [LEN=08] [CRC 2] [TYPE='r'] [CV] [VAL]

CV ist die EEPROM-Adresse, von der gelesen wurde; VAL ist der darin enthaltene Wert.

Write 'W':

[SRC] [DEST] [LEN=08] [CRC 2] [TYPE='W'] [CV] [VAL]

CV ist die EEPROM-Adresse, an die geschrieben wird. VAL ist der zu schreibende Wert.

Write 'w':

[SRC] [DEST] [LEN=08] [CRC 2] [TYPE='w'] [CV] [VAL]

CV ist die EEPROM-Adresse, an die geschrieben wurde. VAL ist der geschriebene Wert.