

Die Einsteigerzentralen z21 und z21Start von Roco sind seit ihrem Erscheinen bei Modellbahner sehr beliebt. Das liegt zum einen ganz sicher an ihrem günstigen Anschaffungskosten. Zum anderen haben sie auch durch die Beigabe in den digitalen Startpackung von Roco und Fleischmann gerade bei Anfängern eine nicht unerhebliche Verbreitung gefunden. Gebrauchte Zentralen oder überzählige aus solchen Startpackungen findet man zudem auch wieder recht günstig im Gebrauchtmrkt. Die "kleine" z21 bietet für den Einstieg alles was man zum digitalen Fahren und Schalten seiner Modellbahn braucht, bei Interesse (und ggf. mit Freischaltcode) auch in Verbindung mit einer PC-Steuerung über LAN oder (mit WLAN-Router) auch WLAN. Einzig die Erweiterungsmöglichkeiten direkt an der Zentrale sind gegenüber der "großen" Z21 sichtlich eingeschränkt. Neben dem X-Bus für die Verbindung zu Handreglern und dem B-Bus für den Anschluss externer Booster bietet sie hier lediglich den Roco-eigenen R-Bus als Möglichkeit, Rückmeldungen von der Anlage direkt in die Zentrale zu bekommen. Das Angebot an passenden Modulen hierfür ist überschaubar, neben Roco bieten Digikeijs bzw. YaMoRC und der Schweizer Hersteller Modellbahnelektronik.ch (siehe auch [1]) geeignete Fertigprodukte hierfür an. Günstige Bausätze oder Eigenbaulösungen hierfür sind mir zumindest bisher nicht bekannt. Das liegt sicherlich auch daran, dass Roco seinen R-Bus nicht öffentlich zugänglich dokumentiert. Informationen hierzu sind aber trotzdem im Internet zu finden und da ich auch Spaß am Selbstbau von Elektronikkomponenten für die Modellbahn habe, war es nur eine Frage der Zeit, bis hier etwas Eigenes entstand.

Der R-Bus ist physikalisch ein RS485-Bus, welcher analog zum X-Bus mit einer Übertragungsrate von 62,5kbit/s und einem Datenformat mit 9 Datenbits und ohne Paritätsbit arbeitet. Dieses etwas exotische Datenformat schließt den direkten Anschluss an den PC weitestgehend aus, da die in modernen PCs oder USB-UART-Adaptern verbauten Bausteine üblicherweise keine Datenübertragung mit 9 Datenbits können. Auch in Sachen Übertragungsprotokoll lehnt sich der R-Bus sehr stark an den X-Bus bzw. XpressNet an, weshalb z.B. die Zentrale von Digikeijs bzw. deren Nachfolger beide Busse auch physikalisch über einen Anschluss bedienen kann. Mit Hilfe der Informationen aus dem Internet und eigener Analysen mit einem Logikanalysator konnte ich mir das Telegrammformat erklären.

Die Zentrale sendet am R-Bus-Anschluss in sehr kurzen Abständen (ca. 8ms) ein sogenanntes Rufbyte mit gesetztem 9. Bit und für die XpressNet-Adresse 31 (0x15F). An meiner z21 hört keiner auf diesen Ruf, weshalb hier keine Antworten zu sehen sind. Einige Sekunden nach dem Start taucht im Datenstrom auch mal das für den R-Bus herangezogene Rufbyte 0x1DE (eigentlich das XpressNet-Rufbyte für das Gerät mit der Adresse 30) auf. Darauf reagieren dann angeschlossene R-Bus-Module und die Zentrale sendet nur noch diese Anfragen. Nach dem Rufbyte folgt jeweils ein sogenanntes Headerbyte. Für das Einlesen von Rückmeldungen wird 0x0FC gesendet. Die unteren 4 Bit stehen in der Definition von XpressNet für die Anzahl der Bytes. Für die normale Rückmeldung beim R-Bus kommen mit dem Infobyte, den jeweils 1 Byte Rückmeldedaten für 10 Modulen und der Checksumme 12 Bytes (12dez = 0x0C) zusammen. Bei der Adressprogrammierung wird 0x0F1 als Header gesendet. Für das dann folgende Infobyte habe ich nur die 4 Möglichkeiten gesehen:

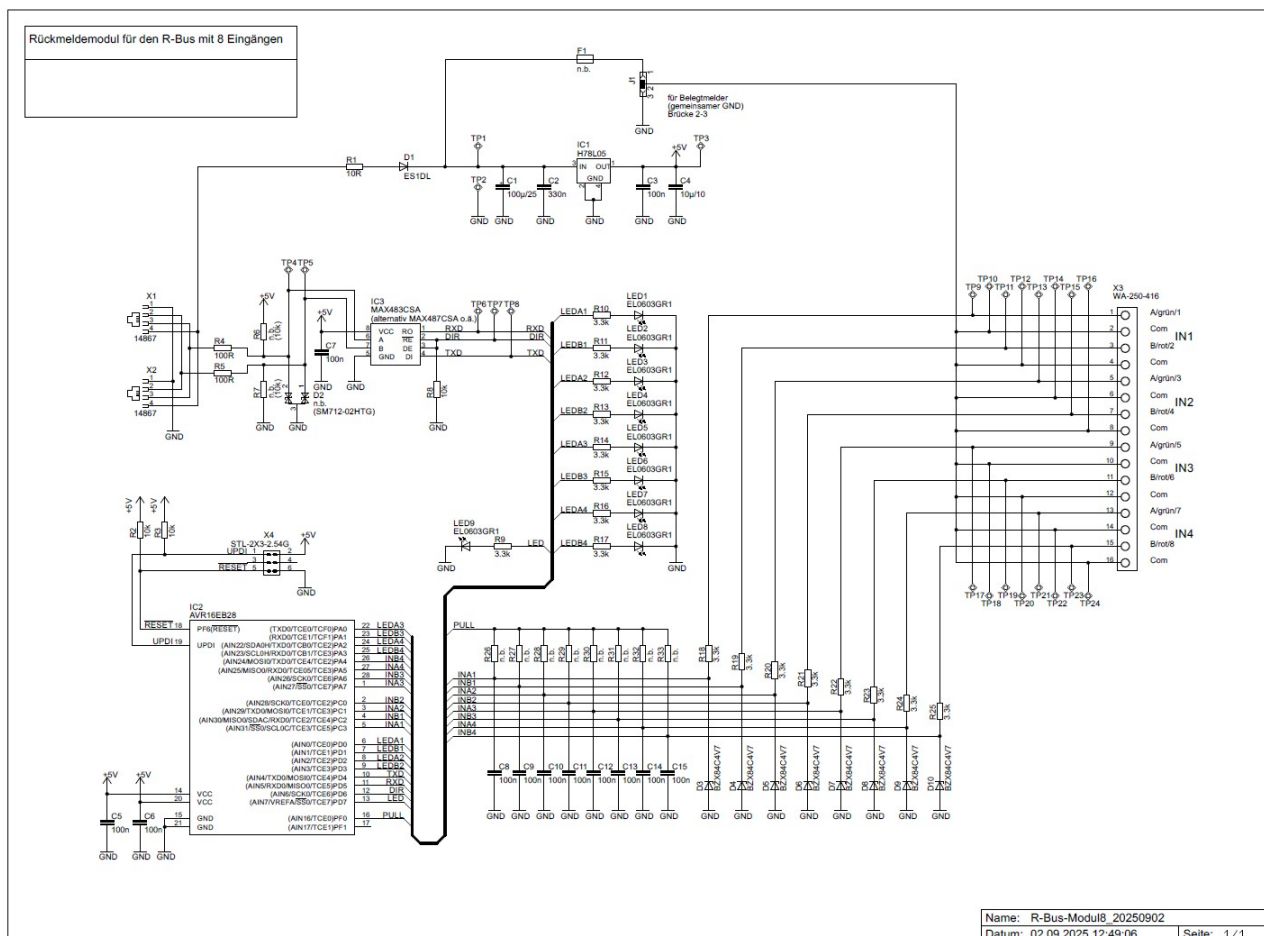
0x000 Rückmeldeinformationen Gruppe 0,
0x010 Rückmeldeinformationen Gruppe 1,
0x0C? Programmierung Adresse Gruppe 0 und
0x0D? Programmierung Adresse Gruppe 1 (die Fragezeichen stehen für die zuzuweisende Moduladresse)

Nach einem Infobyte 0x000 legt das Modul mit der Adresse 1 seine 8 Bit auf den Bus, dann das Modul mit Adresse 2 usw. bis zum Modul mit Adresse 10. Zum Abschluss sendet wieder die Zentrale eine Checksumme. Dieser Ablauf erklärt auch die Einschränkung am R-Bus, dass die

Adressen zwingend fortlaufen aufsteigen mit 1 beginnend verwendet werden müssen. Sendet kein Modul mit Adresse 1 sein Datenbyte weiß auch ein Modul mit Adresse 2 nicht, wann es senden soll und bleibt stumm. Die Moduladressen sind beim R-Bus wie oben schon erwähnt in 2 Gruppen aufgeteilt. Nach der Abfrage der Module der Gruppe 0 folgt ein erneutes Rufbyte, Headerbyte und dann das Infobyte 0x010 für die Module mit den Adressen 11..20. Es ist also theoretisch entgegen obiger Aussage möglich, ein Modul mit der Adresse 11 ohne vollständig vorhandene Module 1 bis 10 zu verwenden. Nur in den beiden Gruppen müssen alle Module aufsteigend vorhanden sein. Die Abfragen kommen in sehr kurzen Abständen von ca. 8ms. Ein Abfragezyklus für alle 20 Module dauert nach meiner Messung nur ca. 25 ms. Auch den Ablauf der Adressprogrammierung für die R-Bus-Module über das Maintenance-Tool von Roco habe ich mir mit dem Logikanalysator angesehen und die Funktion in der Software entsprechend umgesetzt. Hier wird statt der Infobytes für die Abfrage der Modulgruppen ein Byte 0x0C? oder 0x0D? gesendet, wobei ersteres für die Module 1 bis 10 und letzteres für die Module 11 bis 20 verwendet wird. An der Stelle des Fragezeichens steht dann die zuzuordnende Moduladresse 1 bis A(hex), also 0xC1 bis 0xCA für die Module mit den Adresse 1 bis 10 und 0xD1 bis 0xDA für 11 bis 20. Da jedes Modul auf diese Befehle zur Adresszuweisung reagiert, darf hier immer nur das zu programmierende am R-Bus angeschlossen sein.

Soweit die (Software-)Theorie. Die für die Umsetzung nötige Hardware gestaltet sich dann auch sehr übersichtlich. Nachdem ich vor längerem bereits mit einem Aufbau auf Lochrasterplatine und einem älteren ATmega8515 erfolgreich Daten an meine z21 übertragen konnte, sollte nun eine kleine, moderne „professionelle“ Platine entstehen. Neben dem zentralen Mikrocontroller, der Eingangsbeschaltung für die digitalen Eingänge, den LEDs für eine Statusanzeige und einem zugehörigen Spannungsregler ist nur noch der Treiberbaustein für die RS485-Schnittstelle und ein passender Steckverbinder hierfür vorhanden. Für die R-Bus-Anschlüsse habe ich zwei 4-polige Modularsteckverbinder verwendet. Diese sind kleiner als die an der Zentrale verwendeten 6-poligen Ausführungen. Bei Selbstherstellung der Verbindungskabel lassen sich aber beide Varianten leicht adaptieren. Es werden ja auch nur 4 Signalleitungen verwendet. Die Anschlüsse für die Rückmeldeeingänge habe ich mit Federkraftklemmen im Rastermaß 2,5mm bestückt. Hier gehen aber alternativ je nach Vorliebe auch kleine Schraub- oder Steckklemmen im selben Rastermaß. Die von mir verwendeten Federkraftklemmen sind nicht als 16-polige Variante verfügbar. Ich habe sie mir aus jeweils zwei 8-poligen („Steckbaukasten“) zusammengebaut. Man kann auch mit 4 Stück in 3-poligen Ausführen arbeiten, wenn sich 2 benachbarte Eingänge den gemeinsamen Anschluss teilen (z.B. in Kombination mit den LB101 von Lenz). Ein Pin bleibt dann zwischen den Klemmen frei. Der verwendete Mikrocontroller ist ein noch verhältnismäßig neuer und weniger bekannter Typ aus der AVR-Reihe von Microchip (vormals Atmel), bietet aber mit seinem sehr guten Preis-Leistungsverhältnis eine gute Auswahl für dieses Projekt. Die vorhandenen 16 kByte Programmspeicher werden nur zu einem sehr kleinen Teil genutzt. Er benötigt (für diese Anwendung) keinen externen Quarz und bietet zudem mit dem UPDI-Anschluss eine einfache und komfortable Möglichkeit zur Programmierung und zum Debugging. Die Eingangsbeschaltung der digitalen Eingänge orientiert sich am Einsatz zusammen mit dem Belegtmelder LB-101 von Lenz oder ähnlichen Modulen, welche ihre Signalausgänge potentialgetrennt über einen Optokoppler gegen einen gemeinsamen GND-Anschluss schalten. Die Möglichkeit, das Potential des gemeinsamen Anschlusses für die Eingänge über den Lötjumper J1 auch auf die positive Versorgungsspannung des Moduls zu legen, erlaubt hardwareseitig auch den Anschluss von Gebern mit gemeinsamen positiven Anschluss, wird hier aber nicht genutzt (und bedarf für eine korrekte Anzeige der Meldung in der Zentrale dann eine Invertierung der Signalzustände in der Software). Die Eingänge sind über Serienwiderstände entkoppelt und mit Z-Dioden und Kondensatoren gegen Überspannung geschützt. Statt der externen Pullup-Widerstände R10 bis R17 werden die internen verwendet. Der Spannungsregler IC1 erzeugt stabile 5V für den Mikrocontroller und der RS485-Treiber. Beide verfügen über die üblichen Abblockkondensatoren und die RS485-Leitungen enthalten ebenfalls Serienwiderstände zum Schutz der Anschlüsse. Ein Abschlusswiderstand ist auf

der Platine nicht vorgesehen und wird nach meiner Erfahrung für diese Anwendung auch nicht zwingend benötigt. Noch ein Hinweis zum vorgesehenen RS485-Treiber: verwenden Sie bitte hier lieber den angegebenen langsameren Typ (Max487 oder Max483) statt des oft eingesetzten Max485. Letzterer ist für Datenübertragungen bis 12Mbit/s vorgesehen und erzeugt dafür sehr steile Schaltflanken auf den Signalleitungen. Dies ist für den Einsatz im Modellbahnbereich mit seinem oft nicht perfekten Leitungsverläufen eher nachteilig und hier auch überhaupt nicht nötig. Die langsameren Typen reichen vollkommen aus. Die Widerstände R6 und R7 waren bei mir nicht nötig. Die Schutzdiode D2 kann im Normalfall auch entfallen. Für die 9 Status-LEDs habe ich ein Exemplar mit geringer Stromaufnahme aber ausreichender Helligkeit verwendet (grün, EL0603GR1 bei [2]). Bei anderen Typen (Bauform 0603) oder anderer Helligkeit können die Vorwiderstände (im Rahmen) angepasst werden. Bedienelemente wie Taster o.ä sind nicht vorgesehen.



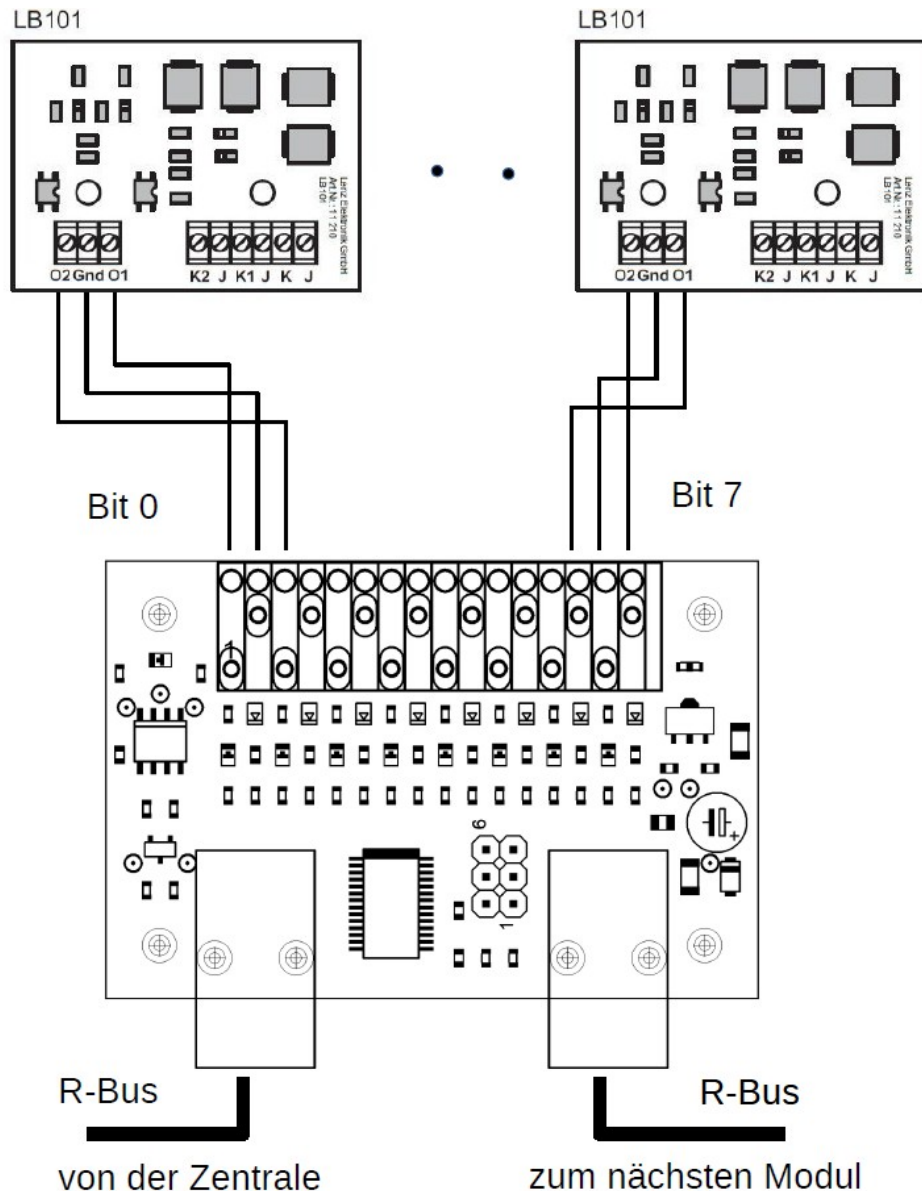
Der Aufbau sollte sich für im Lötten von SMD-Bauteilen geübte Bastler problemlos gestalten. Man kann sich die SMD-Bauteile natürlich auch bei der Bestellung der Platinen gleich Bestücken lassen. Lohn der Mühe mit den kleinen Bauteilen (und des Kompromisses mit dem 4-poligen R-Bus-Anschluss) ist eine Platine mit sehr geringen Abmessungen von ca. 41mm x 61mm. Vergessen Sie beim Bestücken bitte nicht, den Lötjumper mit einem kleinen Klecks Lötzinn entsprechend zu schließen. Die Brücke muss zwischen dem mittleren Pad und dem Pad in Richtung Platinenmitte geschlossen werden.

Die Software ist von mir wieder in einfachem C geschrieben. Eine Arduino-Bibliothek gibt es für den R-Bus wie im bereits erwähnten Artikel in [1] geschrieben offensichtlich nicht. Als Entwicklungsumgebung nutze ich seit vielen Jahren das AtmelStudio. Die Quelltexte stehen zur (nichtkommerziellen) Verfügung. Für die Programmierung des Mikrocontrollers auf der Platine ist ein 6-poliger ISP-Anschluss vorhanden. Es werden aber nur die Versorgungsspannung und das

Signal UPDI genutzt. Dementsprechend ist ein UPDI-fähiger Programmieradapter nötig. Ich selbst verwende einen AtmelIce-Basic, welcher aber zwingend die aktuelle Firmware 1.2d hierfür benötigt. Alternative UPDI-Programmieradapter findet man im Internet ([3]). Hier sollte man aber explizit auf die Unterstützung für die Programmierung der AVR-Ex-Serie achten. Für eine sichere Inbetriebnahme und Programmierung kann man sich ein Anschlusskabel für ein Labornetzteil (ca. 8 bis 16V Gleichspannung) für X1 bzw. X2 bauen. Alternativ geht natürlich auch der Anschluss an die Zentrale. Deren Versorgungsspannungsausgang am R-Bus ist mit einer selbstrückstellenden Sicherung 0,5A abgesichert. Das Modul selbst nimmt maximal ca. 15 mA auf (alle LED eingeschaltet).

Nach erfolgreichem Aufbau und Programmierung muss dem Modul wie bei den kommerziellen Geräten auch eine Adresse zugewiesen werden. Im Quelltext ist als Default-Adresse die 1 vorgegeben. Die Adressvergabe funktioniert ganz genau so wie z.B. für die Module von Roco im Maintenance-Tool beschrieben. Nach erfolgreicher Programmierung der Adresse wird die aktuelle Adresse (die LED an Eingang 1 ist das niederwertigste Bit der Anzeige im hexadezimalen Format) kurz über die Status-LEDs angezeigt. Bei jedem Anstecken des Moduls an die Zentrale (oder eine andere Spannungsversorgung) werden zuerst kurz alle LEDs aktiviert. Danach erfolgt wieder die Anzeige der aktuellen Adresse und im laufenden Betrieb signalisiert eine leuchtende LED den Status des zugehörigen Einganges. Die einzelne Status-LED blinkt nach Verbindung mit dem R-Bus oder einer anderen Spannungsquelle schnell und blinkt langsamer, wenn die Kommunikation mit der Zentrale ordnungsgemäß abläuft.

Für den Test an meiner Anlage habe ich 4 Module auf die beiden Gruppen aufgeteilt (Adressen 1 bis 3 und Adresse 11) und mir die Datenübertragung und die Anzeige im Maintenance-Tool von Roco am PC angesehen. Eine Belegung (die betreffende LED leuchtet) wird dort (mit etwas Verzögerung) in rot dargestellt. Offene Eingänge oder Eingänge von nicht vorhandene Module erscheinen grün. Als Signalgeber für die Eingänge verwende ich neben Eigenbauten Belegtmelder auch die erwähnten LB101 von Lenz. Das Bild zeigt beispielhaft den Anschluss dieser Melder an das R-Bus-Modul8.



Die Module funktionieren bei mir ohne Probleme. Mit dem Wissen um die Funktion des R-Bus habe ich in meine selbst entworfenen Weichendekoder eine RS485-Schnittstelle und die passende Softwareergänzung eingebaut, sodass ich die Stellung der daran angeschlossenen Weichen (oder besser der Weichenantriebe MP1, da die Erfassung nur über deren Endabschaltung erfolgt) auch über den R-Bus einlesen und perspektivisch damit auf einem ebenfalls selbst entwickeltem und selbstgebaurem Gleisbildstellpult darstellen kann. Weiter denke ich auch über verschiedene Möglichkeiten nach, den vorhandenen X-Bus und R-Bus logisch miteinander zu verknüpfen und die Beschränkungen des Letztgenannten möglicherweise etwas aufzuweichen. Das Ganze ginge z.B. mit Loconet und gekauften Komponenten wahrscheinlich einfacher, aber die Entwicklung elektronischer Baugruppen speziell für die Modellbahn bereitet mir eben sehr viel Freude. Leider fehlt die Zeit dann für die eigentliche Modellbahn, aber das ist ein anderes Kapitel.

[1] Digitalen Modellbahn Heft 2/2025 ab Seite 24

[2] Reichelt.de

[3] Übersicht AVR-Programmieradapter auf <https://github.com/waginator/AVR-Programmer>