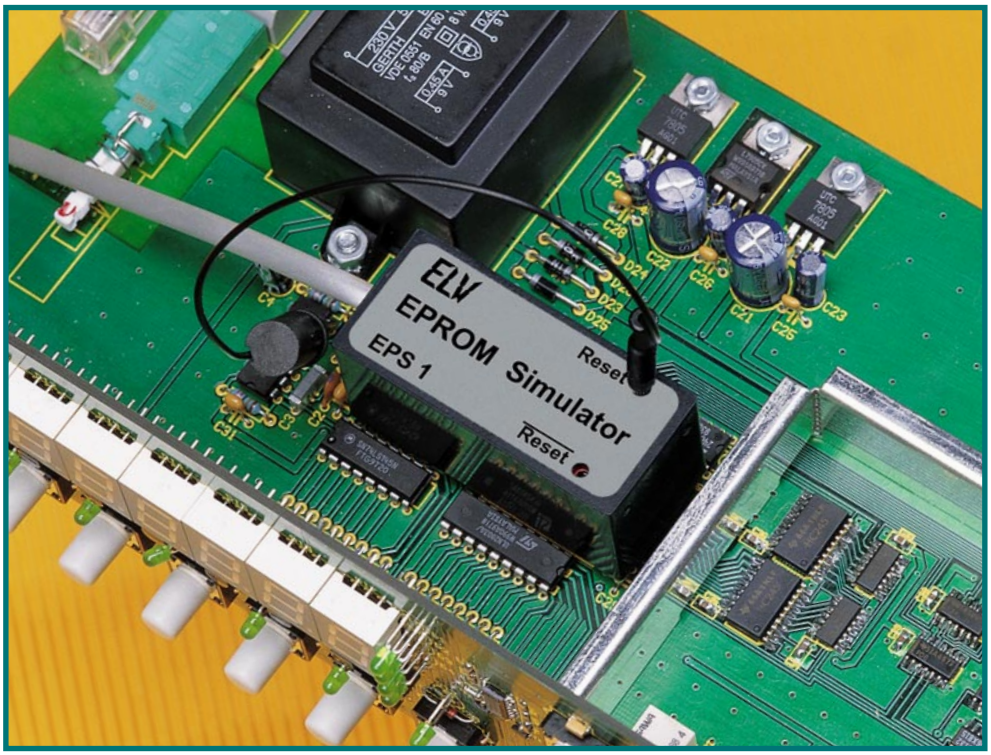

Programmieren on the Fly - der EPROM-Simulator Teil 1

Ein Programm ist erst fertig, wenn es im Zielsystem fehlerfrei läuft - eine Programmierer-Binsenweisheit. Bis man dies erreicht hat, ist zuweilen viel Zeit- und Materialaufwand erforderlich. Der neue ELV-EPROM-Simulator besticht durch seinen kompakten Aufbau, er ist transportabel ohne Datenverlust und kann auch ohne angeschlossenen PC im Anwendungssystem betrieben werden. Er simuliert 32k x 8, 64k x 8 und 128k x 8 EPROMs und deckt damit nahezu alle gängigen EPROM-Größen ab. Durch die flexible Steuerung mittels eines integrierten Prozessors sind Programmkorrekturen quasi online möglich.

Der lange Weg zum fertigen Programm

Jeder, der eigene Programme entwikkelt, kennt den Hergang. Das Programm ist geschrieben, der Assembler hat seine Arbeit getan, man „brennt" das EPROM, setzt es im Zielsystem ein und findet dann die Bugs beim Lauf dieses Zielsystems. Also EPROM herausnehmen, löschen, neu programmieren...

Natürlich gibt es schon lange EPROM-Simulatoren, die auch einiges leisten. Herkömmliche Simulatoren werfen jedoch nahezu immer irgendein Problem auf, sei es die feste Bindung an den programmierenden PC, die umständliche Handhabung durch die Größe und unkomfortable Programmiergänge, zahlreiche Verbindungskabel, Stromversorgungsprobleme, hohe Preise und, und...

Komfortabler Problemlöser - der ELV-EPROM-Simulator

Aus diesen Erfahrungen heraus entstand der neue ELV-EPROM-Simulator, der eine Reihe der beschriebenen Nachteile vermeidet. Er ist vor allem erst einmal eines - äußerst kompakt, wie im Titelfoto zu sehen ist. Er überschreitet nur unwesentlich die Ausmaße eines 32poligen EPROMs, was einen sehr unkomplizierten Einsatz direkt auch in räumlich engen Zielsystemen möglich macht. Er wird einfach in die EPROM-Fassung eingesetzt, hierüber auch mit der erforderlichen Betriebsspannung versorgt und „hält" die Verbindung zum programmierenden PC lediglich über ein dünnes 4poliges (serielles) Kabel, das aufgrund des sehr kompakten Western-Modular-Steckverbinders ebenfalls in enger Umgebung weniger Probleme verursacht als manch herkömmliches Simulator-System. Möglich ist dies durch eine serielle Datenübertragung zum und vom Simulator. Diese erfolgt über eine serielle Schnittstelle des PCs.

Nach dem Programmieren des Simulators vom PC aus kann die Verbindung auch gelöst werden, so daß der Simulator z. B. auch über längere Testphasen ohne Bindung an sein Programmiersystem im Zielsystem arbeiten kann - wichtig z. B. für Langzeiterprobungen oder transportable Systeme. Auch bei abgeschaltetem System bleiben die Daten im Simulator erhalten -

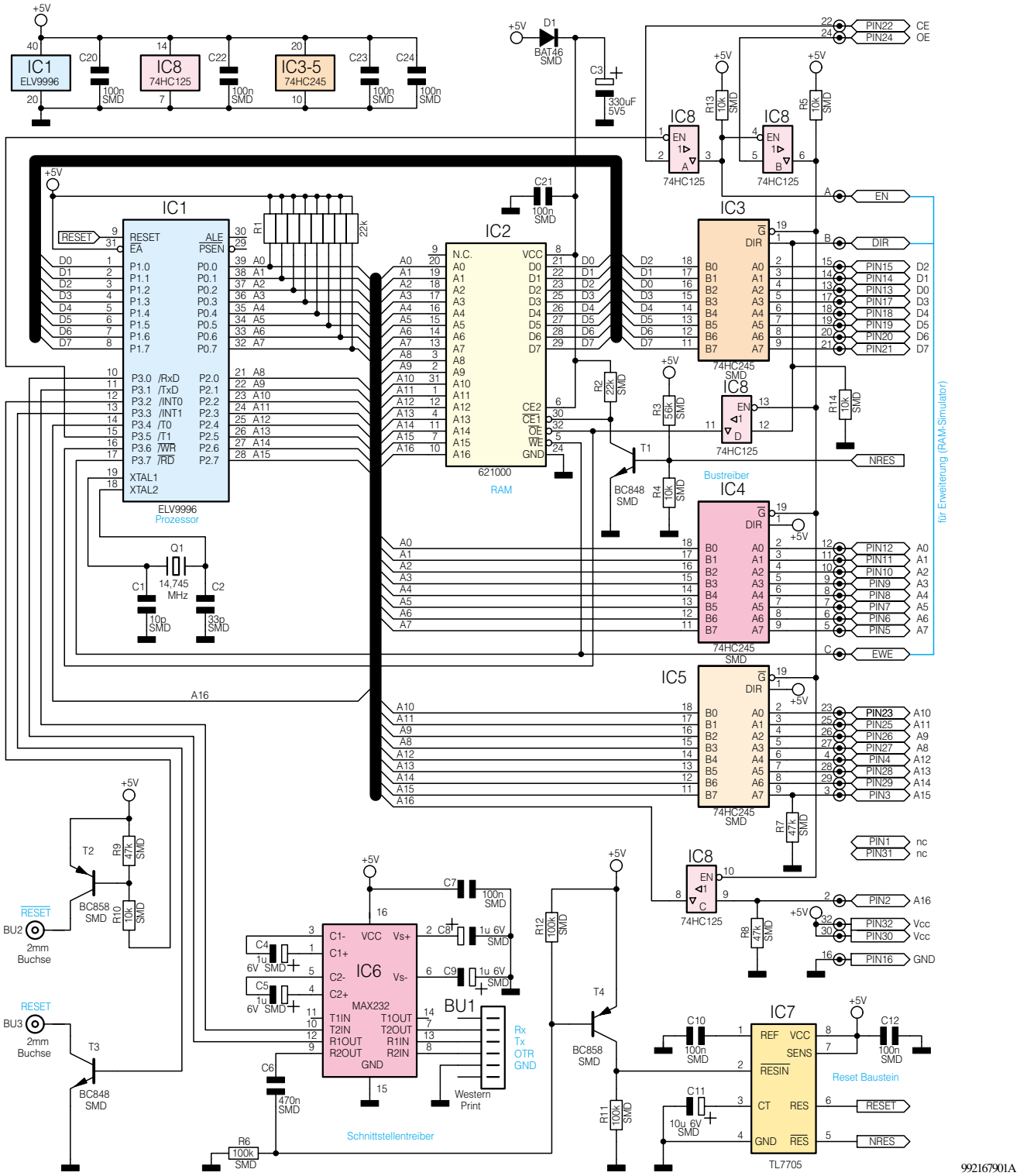

Bild 1: Schaltung des EPROM-Simulators

ein Gold-Cap sowie der Einsatz besonders leistungsarmer Bausteine sorgen für Datenerhalt über bis zu 10 Stunden. So ist der Simulator mit geladenem Programm z. B. auch gut transportabel.

Da in das kleine Gerät nach dem Aufbau nicht mehr eingegriffen werden muß, erhält es seinen Platz in einem kompakten Kunststoffgehäuse (57 x 26 x 27 mm), das die Handhabbarkeit des Simulators weiter erleichtert.

Und schließlich ist es dank des im Simulator integrierten Mikroprozessors jederzeit unkompliziert möglich, das im RAM des Simulators abgelegte Programm nach einer Modifikation neu vom programmierenden PC zu laden und das Ergebnis quasi sofort zu sehen. Der Simulator kann also stets im EPROM-Sockel des Zielsystems verbleiben.

Während vom PC Daten in den Simulator oder Daten aus diesem gelesen werden, ist die Verbindung zum Zielsystem durch die Bustreiber getrennt und die Reset-Ausgänge sind aktiviert. Erst nachdem die Datenübertragung abgeschlossen ist, werden die Verbindung zum Zielsystem wieder neu hergestellt und die Reset-Ausgänge wieder deaktiviert. Dadurch kann das Zielsystem dann neu gestartet werden. Um den Stromverbrauch des Simulators zu minimieren, schaltet der interne Prozessor nach der Datenübertragung in den Power-Down-Mode, der nur durch ein Reset vom PC über die DTR-Leitung beendet werden kann.

Neue Anwendungen

Die Möglichkeit des Runterladens von kurzen Datenbereichen verschafft dem EPROM-Simulator sehr interessante Anwendungsmöglichkeiten. So kann z. B. die Erarbeitung von Grafiken und Ausschriften z. B. für LC-Displays sehr schnell und gut kontrolliert erfolgen, Daten-Ent- und Verschlüsselungen (z. B. Scrambles) sind schneller realisierbar, Listen äußerst komfortabel zu aktualisieren.

Auch in der Ausbildung oder der eigenen Weiterbildung ist solch ein schneller Simulator sehr nützlich, Experimentierfreudige kommen ebenfalls schneller zum Ergebnis, z. B. bei der Erzeugung von Bildern und Grafiken.

Schaltung

Ein Blick auf das Schaltbild (Abbildung 1) zeigt, daß die beschriebene Funktionsvielfalt mit relativ geringem Aufwand erzielt werden konnte. Rings um den Prozessor und das Simulator-RAM ist nur wenig Peripherie erforderlich.

So z. B. der RS232-Wandler (IC 6), der die vom PC kommenden seriellen Signale für den Prozessor aufbereitet bzw. umgekehrt die Signale des Prozessors für ein Versenden im V.24-Protokoll umwandelt. Durch die Beschaltung mit C 7 bis C 9 ist hier eine Versorgungsspannung von +5 V für die interne V.24-Signalaufbereitung ausreichend.

BU 1 ist eine 6polige Western-Modularbuchse, die durch ihre Kompaktheit und leichte Handhabbarkeit den Vorzug vor anderen Verbindungs-Systemen erhalten hat.

Die mit TTL-Pegel von IC 6 ausgegebenen Signale gelangen an den seriellen Port des Prozessors IC 1, ein maskenprogrammierter 80C52. Dessen Taktversorgung erfolgt durch Q 1, C 1 und C 2.

An BU 2 und BU 3 stehen zwei zueinander negierte RESET-Ausgänge zur Verfügung, über die das Zielsystem während des Programmladens in das RAM ständig im Reset-Zustand gehalten wird, denn ohne vollständiges Programm im EPROM-Simulator ist dieses in aller Regel nicht arbeitsfähig.

Ist das Laden des Programms abgeschlossen, erfolgt eine Freigabe der RESET-Ausgänge.

Ein Universal-Reset-IC des Typs TL 7705 (IC 7) sorgt für einen definierten Reset beim Starten des Programmiervorgangs vom PC aus (siehe Funktionsablauf), beim Zuschalten der Spannungsversorgung oder ihrer Wiederkehr nach Spannungsausfall.

Der NRES-Ausgang des Reset-ICs deaktiviert über die Transistorstufe T 1 das RAM während des Hochfahrens des Prozessors beim Einschalten oder bei Spannungswiederkehr, um keine undefinierten Zustände zwischen Prozessor und RAM zuzulassen.

Über Pin 15 (Port 3.5, T1) des Prozessors und über die Torschalter IC 8 A/B erfolgt beim Beschreiben des RAMs ein Sperren der Bustreiber IC 3 bis IC 5.

Sind diese nach Abschluß des Beschreibens freigegeben, kann eine normale EPROM-Zugriffsteuerung über die Leitungen CE/OE der EPROM-Fassung erfolgen.

Die Bustreiber arbeiten durch Festlegen der Richtungssteuerung (Pin 1, DIR) nur in die jeweils definierte Richtung (Daten Out, Adressen In).

Die Steuereingänge „DIR", „EN" und „EWE" sind für eine Erweiterung des EPROM-Simulators zu einem RAM-Simulator, der in einem der nächsten Hefte vorgestellt wird, vorgesehen.

Über IC 8 C erfolgt die Aktivierung der Adresse A16 des RAMs für die Simulation des EPROM-Typs 27C010 (128k x 8). Bei der Simulation der kleineren EPROMs 27C256 und 27C512 bleibt dieser Pin wie auch die Pins 1, 31 und 32 der 32poligen Steckerleiste des EPROM-Simulators frei. R 8 sorgt dann für ständigen Low-Pegel an Pin 9 von IC 8 C, so daß jetzt nicht auf die Adreßleitung A 16 zugegriffen werden kann.

Damit ist auch die Zuordnung von unterem und oberem Adreßbereich des 128k x 8-RAMs 621000 beim Simulieren von 128k-, 64k- und 32k-EPROMs definiert.

Die Spannungsversorgung des Simulators erfolgt mit +5 V über das Zielsystem. C 3 ist ein Gold-Cap, der bei Spannungsausfall für Datenerhalt sorgt. C 20 bis C 25 arbeiten als Abblock-Kondensatoren für die einzelnen ICs.

Funktionsablauf/ Datenübertragung

Nach dem Einschalten des Simulators werden beide RESET-Signale aktiviert, eine Datenübertragungsrate von 9600 Baud sowie eine Time-Out-Zeit für den Prozessor von 1 s gesetzt.

Um die serielle Datenkommunikation zwischen PC und Simulator zu starten, ist die DTR-Leitung der RS232-Schnittstelle für min. 100 ms auf Low-Pegel und anschließend für min. 100 ms auf High-Pegel zu setzen. Dies versetzt den Prozessor in die Bereitschaft, eine Kommunikation mit dem PC aufzunehmen. Gleichzeitig werden das Zielsystem über BU 2/3 wie beschrieben in den Reset-Zustand versetzt und die Bustreiber gesperrt.

Anschließend erfolgt nun die Übertragung der Daten mit einem seriellen Protokoll.

Die Daten des PCs werden in einem Befehlsrahmen, bestehend aus:

<SOH>, [Daten], <CRC>, <EOT>

gesendet (CRC - Cyclic Redundance Check, 16Bit-CRC-Summe mit dem Generatorpolynom $X^{16} + X^{12} + X^5 + 1$).

Die Daten, die vom PC an den Simulator geschickt werden [Daten], bestehen aus einem Befehl und darauf folgenden Parametern, die u. a. auch die eigentlichen Datenbytes des an den Simulator zu übertragenden Programms enthalten. Die Befehle und Parameter sind in Tabelle 1 ausführlich beschrieben. Auf jeden Rahmen sendet der Simulator eine Bestätigung, die ebenfalls in einen Befehlsrahmen, bestehend aus:

<STX>, <Antwort>, <CRC>, <ETX>

gefaßt sind.

Die Antwort besteht entweder aus der Antwort <ACK> und eventuellen Daten (positive Bestätigung und angeforderte Daten aus dem Simulator) oder aus <NAK> und einer Fehlernummer (negative Bestätigung und Fehlercode) in der Form:

<ACK> [Daten] oder
<NAK> [Fehlernummer]

Wann welche Daten erwartet werden, ist ebenfalls der Befehlsbeschreibung in Tabelle 1 zu entnehmen.

Naturgemäß dürfen innerhalb der Datenblöcke beim Datenaustausch die Steuerzeichen für Beginn und Ende der Datenübertragung nicht im Klartext vorkommen, da dies z. B. zum Abbruch der Datenübertragung führt.

So sind diese durch die folgend beschriebenen Zeichenfolgen zu ersetzen. Das DLE-Zeichen dient hierbei dazu, dem Empfänger klar zu machen, daß das nachfolgende Zeichen das eigentliche, um 16 erhöhte Zeichen ist. Deshalb ist natürlich auch ein originales DLE-Zeichen durch eine Zeichenfolge zu ersetzen.

Vom PC zum Simulator:
<SOH>: <DLE> + <DC1>
<EOT>: <DLE> + <DC4>
<DLE>: <DLE> + <SPACE>

Vom Simulator zum PC:
<STX>: <DLE> + <DC2>
<ETX>: <DLE> + <DC3>
<DLE>: <DLE> + <SPACE>

Das gesamte Datenübertragungsverfahren wird als transparente Datenübertragung mit Code-Shifting (wegen der beschriebenen Ersetzung der Zeichenfolgen) bezeichnet.

Abschließend für die Erläuterung der

Tabelle 1: Befehlsbeschreibung für den Datenaustausch

Jeder Befehl besteht aus: **<Befehlsbyte> [Parameterbyts]**

Befehle:
- ‚0' - Datensatz schreiben Lo-Block (0-FFFF)
- ‚1' - Datensatz schreiben Hi-Block (10000-1FFFF)
- ‚2' - Datensatz schreiben beide Hälften (0-7FFF und 8000-FFFF)
- ‚8' - Daten abfragen Lo-Block (0-FFFF)
- ‚9' - Daten abfragen Hi-Block (10000-1FFFF)
- ‚B' - Baudrate setzen
- ‚R' - RESET-Ausgänge setzen
- ‚T' - Time-Out-Zeit setzen
- ‚P' - Power-Down aktivieren
- ‚I' - Simulator-RAM initialisieren

Die kompletten Datenübertragungsformate (siehe Text)

1. PC -> Simulator

SOH	Befehlsbyte	Parameterbyte	CRC	EOT

2. Simulator -> PC

STX	Antwort	Daten bzw. Fehlernr.	CRC	ETX

Befehlsbeschreibung

„B" - Baudrate setzen
Als weiteres Byte folgt der Baudratenindex :
- ‚0' – 4800
- ‚1' – 9600
- ‚2' – 14400
- ‚3' – 19200
- ‚4' – 28800
- ‚5' – 38400
- ‚6' – 57600

Als Antwort wird nur eine pos. Bestätigung ohne Daten erwartet.
Beispiel : ‚B' ‚2' -> setzt eine Baudrate von 14400.
Der Simulator antwortet mit <ACK>

„R" - RESET-Ausgänge setzen
Als weiteres Byte folgt der Ausgangscode:

‚0' –	RESET	inaktiv	$\overline{\text{RESET}}$	inaktiv
‚1' –	RESET	aktiv	$\overline{\text{RESET}}$	inaktiv
‚2' –	RESET	inaktiv	$\overline{\text{RESET}}$	aktiv
‚3' –	RESET	aktiv	$\overline{\text{RESET}}$	aktiv

Als Antwort wird nur eine pos. Bestätigung ohne Daten erwartet.

„T" - Time-Out-Zeit setzen
Als weiteres Byte folgt die Time-Out-Zeit :
- <0> - Time-Out inaktiv (kein automatisches PowerDown)
- <1>..<255> - Time-Out auf x * 20ms

Sobald die Time-Out-Zeit abgelaufen ist, schaltet sich der Prozessor ab.
Die Time-Out-Zeit wird bei jedem empfangenen Zeichen zurückgesetzt.
Als Antwort wird nur eine pos. Bestätigung ohne Daten erwartet.

„P" Power-Down aktivieren
Sofort den Power-Down-Mode aktivieren (nach Bestätigung).
Als Antwort wird nur eine pos. Bestätigung ohne Daten erwartet.

„I" Speicher initialisieren
Als weiteres Byte folgt das Initialisierungs-Byte.
Als Antwort wird nur eine pos. Bestätigung ohne Daten erwartet, die anzeigt, daß der Speicher gefüllt wurde.

„8" Daten abfragen Bank 0
Als Parameterbytes folgen:

[Quelladresse Lo] [Quelladresse Hi] [Anzahl]

„Anzahl" Datenbytes werden vom Simulator ab der Adresse "Quelladresse" in der Speicherbank 0 abgefragt.
„Anzahl" darf höchstens 64 betragen, da der Simulator keine größeren Blöcke senden kann.
Die Quelladresse liegt beim Simulator in der Bank 0.
Der Überlauf in die nächste Bank erfolgt im Simulator automatisch.
Als Antwort wird vom Simulator eine pos. Bestätigung mit anhängenden "Anzahl" Daten erwartet.

„9" Daten abfragen Bank 1
Als Parameterbytes folgen:

[Quelladresse Lo] [Quelladresse Hi] [Anzahl]

„Anzahl" Datenbytes werden vom Simulator ab der Adresse „Quelladresse" in der Speicherbank 1 abgefragt.
„Anzahl" darf höchstens 64 betragen, da der Simulator keine größeren Blöcke senden kann.
Die Quelladresse liegt beim Simulator in der Bank 1.
Der Überlauf in die nächste Bank erfolgt im Simulator automatisch.
Als Antwort wird vom Simulator eine pos. Bestätigung mit anhängender "Anzahl" Daten erwartet.

„0" Datensatz schreiben Bank 0
Als Parameterbytes folgen:

[Zieladresse Lo] [Zieladresse Hi] [Datenbyte 1] ...[Datenbyte n]

Die Datenbytes 1 bis n werden im Simulator ab der Adresse „Zieladresse" in der Speicherbank 0 abgelegt (0000-FFFF).
„n" darf höchstens 64 betragen, da der Simulator keine größeren Blöcke empfangen kann.
Die Zieladresse liegt beim Simulator in der Bank 0.
Der Überlauf in die nächste Bank erfolgt im Simulator automatisch.
Als Antwort wird vom Simulator eine pos. Bestätigung erwartet.

„1" Datensatz schreiben Bank 1
Als Parameterbytes folgen:

[Zieladresse Lo] [Zieladresse Hi] [Datenbyte 1] ...[Datenbyte n]

Die Datenbytes 1 bis n werden im Simulator ab der Adresse „Zieladresse" in der Speicherbank 1 abgelegt (10000-1FFFF).
„n" darf höchstens 64 betragen, da der Simulator keine größeren Blöcke empfangen kann.
Die Zieladresse liegt beim Simulator in der Bank 1.
Beim Überlauf der Bank wird im Simulator die Bank 0 selektiert.
Als Antwort wird vom Simulator eine pos. Bestätigung erwartet.

„2" Datensatz schreiben Bank 0 beide Hälften
Als Parameterbytes folgen:

[Zieladresse Lo] [Zieladresse Hi] [Datenbyte 1] ...[Datenbyte n]

Die Datenbytes 1 bis n werden im Simulator ab der Adresse "Zieladresse" in der Speicherbank 0 sowohl ab Zieladresse and 0x7FFF als auch ab Zieladresse or 0x8000 abgelegt.
Dies ist erforderlich bei der Simulation eine 32k-EPROMs (27C256), bei denen der Zustand von Pin 1 (A15) nicht definiert ist.
„n" darf höchstens 64 betragen, da der Simulator keine größeren Blöcke empfangen kann.
Die Zieladresse liegt beim Simulator immer in der Bank 0.
Als Antwort wird vom Simulator eine pos. Bestätigung erwartet.

Befehle noch einige Bemerkungen zur Befehlstabelle.

Die Time-Out-Zeit ist von der Grundeinstellung her auf 1 s gesetzt, sie ist aber mit dem Befehl „T" (siehe Tabelle 1) im Wert veränderbar oder ganz abschaltbar.

Für das sofortige Versetzen des Prozessors in den Power-Down-Modus und damit das unmittelbare Aufheben des RESET-Zustands für das Zielsystem steht der Befehl „P" zur Verfügung. Die Reset-Leitungen sind auch softwaremäßig mit dem Befehl „R" setzbar, dieser Zustand wird jedoch beim nächsten Power-Down (Anhalten des Prozessors nach Abschluß der Datenübertragung) automatisch wieder gelöscht. Erfolgt für die eingestellte Time-Out-Zeit keine Kommunikation mehr, so geht der Prozessor in den Power-Down-Modus über. Dabei werden die für das Zielsystem gesetzten RESET-Signale deaktiviert und die Bustreiber wie beschrieben freigegeben - das gefüllte RAM kann im normalen Simulatorbetrieb ausgelesen werden.

Damit ist die Schaltungs- und Funktionsbeschreibung des EPROM-Simulators abgeschlossen. Im zweiten Teil des Artikels erfolgt die ausführliche Beschreibung der Bediensoftware, abschließend gefolgt von der Nachbauanleitung.