Janich & Klass
Computertechnik GmbH

Jk82 SLAVE-CPU I

TECHNISCHE BESCHREIBUNG

Bestellnummer:

TIJ-Z-1602 SLAVE-CPU I (6MHz Version) bestückt und getestet

Ihr autorisierter Händler:

17.12.83

Neumarktstr.46·D-5600 Wuppertal-1·Tel.(0202)442480

Inhaltsverzeichnis:

1. Allgemeine Beschreibung:

Die jk82 Slave-CPU I erweitert Ihr Mikrocomputersystem zu einem Multiprozessorsystem. Da die Kommunikation nur über I/O-Ports erfolgt, ist jedes jk82-Bus kompatible Europakartensystem im Rahmen des freien I/O-Adreßraumes ohne Hardware-Änderungen beliebig zu erweitern.

Die Slave-CPU I besitzt 64KB RAM sowie einen abschaltbaren 28poligen EPROM-Steckplatz für EPROMs von 2KB - 16KB. Damit ist der gesamte adressierbare Speicherraum einer Z80-CPU abgedeckt.

Die Slave-CPU I besitzt ein 2,8" x 2,4" großes Lochrasterfeld zur Aufnahme Ihrer Applikation. Aber auch ohne Erweiterungen kann schon die Grundversion die Effektivität Ihres Computers erheblich steigern. Zum Beispiel lassen sich mit der Slave-CPU zeitaufwendige mathematische Berechnungen parallel zu anderen Programmteilen ausführen.

2. Die technischen Daten in Stichworten:

- Kommunikation über das Z8038-FIO und zwei weitere I/O-Ports
- Asynchroner oder synchroner (WAIT-Leitungen) Datentransfer möglich
- Vector-Interrupt-Steuerung in beiden Richtungen möglich
- 64KB dynamisches RAM
- 28poliger Steckplatz für 2K - 16K EPROMs
- über Jumper einstellbare vollständige Dekodierung der I/O-Portadressen
- 2,8" x 2,4" großes Lochrasterfeld
- Standard-Version 6MHz

3.2. EPROM-Sockel:

Auf der Karte sind folgende EPROMs über Jumper wählbar:

EPROM	S8	S9	S10	S11	
2716	a	a	a	a	
2732	b	a	b	a	**Auslieferzustand**
2764	b	a	a	b	
27128	b	b	b	b	

Der BOOTSTRAPLOADER wird in einem EPROM 2732 mitgeliefert.

3.3. BAI-BAO:

Der Jumper S12 schließt die BAI-BAO Daisy-Chain. Falls gesetzt, muß die Daisy-Chain auch verdrahtet sein.

4. Schaltungsbeschreibung:

4.1. BOOT-Logik:

Das EPROM kann mittels eines "OUT" Befehls ein- oder ausgeblendet werden. Ist das EPROM eingeblendet, so spricht man vom **BOOT-Betrieb.** Der Teil des dynamischen RAMs, der zu dem EPROM parallel liegt, kann dann nicht adressiert werden. In dem EPROM befindet sich üblicherweise der BOOTSTRAPLOADER (s. Beispielprogramm) oder ein beliebiges Applikationsprogramm. Der BOOTSTRAPLOADER ermöglicht es, ein spezielles Anwenderprogramm vom System in den Speicher des Slave zu laden. Nach dem Ladevorgang kann das EPROM ausgeblendet werden.

4.2. Takterzeugung:

Die Karte erzeugt einen eigenen Takt, der im Bereich von 1MHz bis 6MHz liegen kann. Es hängt von der jeweiligen Applikation ab, welche Taktfrequenz die optimale darstellt. Die Standardausführung ist 6MHz (Z80B-CPU). Damit ist die Slave-CPU in den meisten Fällen schneller als das Hauptsystem.

Das Taktsignal Φ wird durch den Transistor T1 auf den erforderlichen Spannungspegel angehoben.

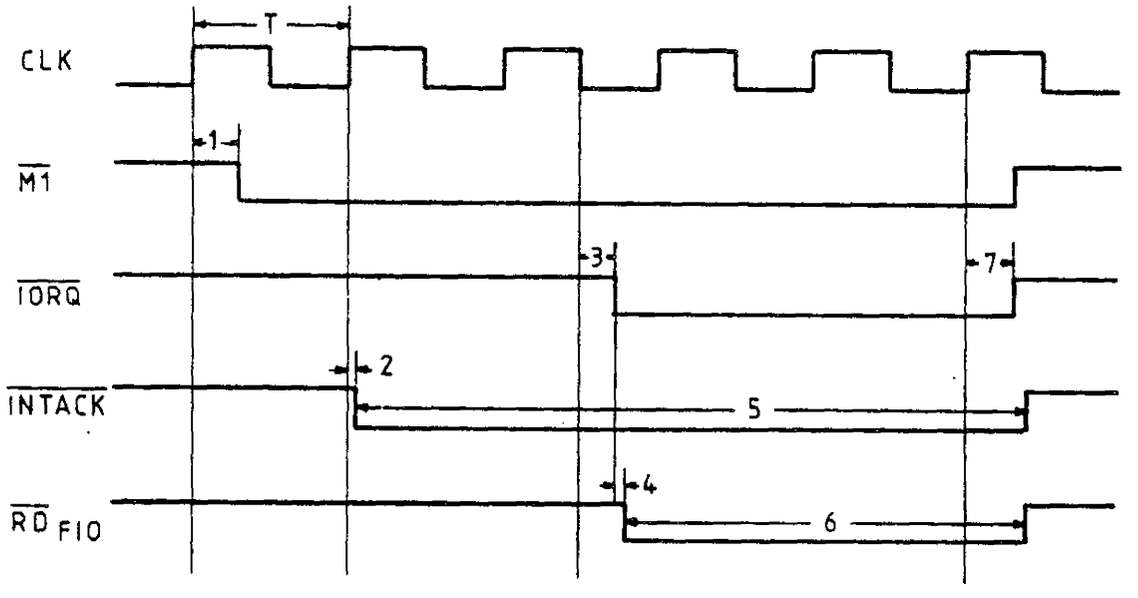
4.3. Erzeugung des INTACK-Signals:

Abweichend vom Z80-Timing benötigt das Z8038-FIO ein INTACK-Signal, um einen Interrupt-Acknowledge-Zyklus anzuzeigen. Dieses Signal zeigt den Beginn des Interrupt-Acknowlege-Zyklus an. Nach Ablauf der Daisy-Chain-Settle-Time muß mit einem READ-Zyklus der Interrupt-Vector gelesen werden. Im folgenden Timing ist das RD-Signal des FIOs mit RD(FIO) bezeichnet.

Die Erzeugung der INTACK-Signale geschieht folgendermaßen:

Das INTACK-F.F. IC7 wird während eines aktiven M1-Zyklus freigegeben. Dieser Zyklus ist entweder ein OPCODE-Fetch oder ein Interrupt-Acknowledge-Zyklus. Ist bis zur nächsten steigenden Flanke des Systemtaktes das MRQ-Signal noch nicht aktiv, so handelt es sich um einen Interrupt-Acknowledge-Zyklus. Der Interrupt-Vector muß mit einem RD-Signal gelesen werden. Dieses Signal wird von IORQ abgeleitet. Das zugehörige Timing zeigt das folgende Bild.

INTACK-Timing:



Die Zeiten sind für eine Taktfrequenz von 4MHz des Hauptsystems nachgerechnet und stellen alle den jeweiligen "worst case" dar.

Lt. Z80-Timing ist:

(T) = 250ns typ.
(1) = 100ns max.
(3) = 85ns max.
(7) = 85ns max.

Ferner wird (7)min. zu 0ns angenommen (worst case).

Schaltungsbedingt ist:

(2) = 30ns max.
(4) = 30ns max.

Die folgenden Zeiten berechnen sich zu:

(5) = (4 * (T)) - (2) + (7)min. = 970ns min.
(6) = (2,5 * (T)) - (3) - (4) + (7)min. = 510ns min.

(6) muß lt. FIO-Timing mind. 400ns sein.

Die Interrupt-Daisy-Chain-Settle-Time von M1 bis IORQ ist für das FIO um (T) - (1) + (2) = 180ns verkürzt . Z80-Bausteine haben eine IEI-IEO-Laufzeit von max. 150ns (4MHz) falls sie nicht mit einer Carry-Lock-Ahead-Logik beschaltet sind.

üblicherweise können also max. 5 Z80 I/O-Bausteine ohne Carry-Look-Ahead in der Daisy-Chain liegen. Ist das FIO an erster oder zweiter Stelle der Priorität müssen im Einzelfall die Zeiten nachgerechnet werden. Das FIO selbst ist mit Carry-Look-Ahead beschaltet. Andere jk82-Bus Karten meistens auch, so daß hier normalerweise kein Problem entsteht.

Auf der Seite der Slave-CPU hat das FIO immer die höchste Priorität, so daß hier kein Zeitproblem entsteht.

5. Steckerbelegungen:

5.1. Steckerbelegung der Pfostenleiste:

Auf der Pfostenleiste stehen alle für weitere I/O-Bausteine benötigten Signale zur Verfügung. In der folgenden Tabelle ist die Pinbelegung der Steckerleiste sowie der Betrag der maximalen Eingangs- oder Ausgangsströme für den Fall, daß das Signal Low ist, aufgeführt. Es kann an dieser Stelle schlecht mit Fan in/out gerechnet werden, da einige Bausteine CMOS Eingänge besitzen.

Die Werte der Inputs sind auf ganze 100yA aufgerundet, die der Outputs entsprechend abgerundet.

Ein LS-Fan-In von eins sinkt einen Strom von 0,4mA, ein CMOS-Eingang (Z80-Peripherie) einen Strom von 10yA.

Signal			Strom (mA) In	Out	Kommentar
1	o	IEO	---	1,9	FIO hat höchste Priorität
2	o	D7	0,1	1,9	Datenbit
3	o	D6	0,1	1,9	Datenbit
4	o	D5	0,1	1,9	Datenbit
5	o	D4	0,1	1,9	Datenbit
6	o	D3	0,1	1,9	Datenbit
7	o	D0	0,1	1,9	Datenbit
8	o	D2	0,1	1,9	Datenbit
9	o	D1	0,1	1,9	Datenbit
10	o	A7	---	1,5	Adreßbit
11	o	A6	---	1,5	Adreßbit
12	o	A5	---	1,5	Adreßbit
13	o	A4	---	1,5	Adreßbit
14	o	GND			
15	o	+5V			100nF pro 50mA Last gegen GND
16	o	A3	---	1,5	Adreßbit
17	o	A2	---	1,3	Adreßbit
18	o	A1	---	1,3	Adreßbit
19	o	A0	---	1,5	Adreßbit
20	o	$\overline{M1}$	---	1,2	Machine Cycle 1
21	o	$\overline{RESET}$	---	7,9	Reset
22	o	$\overline{IORQ}$	---	1,8	Input/Output Request
23	o	$\overline{WAIT}$	2,0	---	Wait
24	o	$\overline{WR}$	---	1,9	Write
25	o	$\overline{RD}$	---	1,2	Read
26	o	$\overline{INT}$	2,0	---	Interrupt
27	o	ϕ	---	20	Clock

Diese Signale stellen alle die internen Signale der Slave-CPU dar. Insbesondere ist auch das Signal ϕ ein 6MHz-Signal und ist nicht mit dem gleichnamigen Signal des ECB-Busanschluß zu verwechseln.

5.2. Busbelegung:

Input/Output		LS-Fan out	in
A0	Adresse 0	60	2
A1	Adresse 1	60	1
A2	Adresse 2	60	1
A3	Adresse 3	60	1
A4	Adresse 4	60	1
A5	Adresse 5	60	1
A6	Adresse 6	60	1
A7	Adresse 7	60	1
D0	Data 0	60	3
D1	Data 1	60	3
D2	Data 2	60	3
D3	Data 3	60	3
D4	Data 4	60	2
D5	Data 5	60	2
D6	Data 6	60	2
D7	Data 7	60	2
MRQ	Memory Request	60	1
IORQ	I/O Request	60	2
RD	Read	60	1
WR	Write	60	1
M1	Maschinenzyklus 1		3
ϕ	Clock		2
PWRCL	Power on clear		1
IEI	Int. Enable in	--	2
IEO	Int. Enable out	5	--
INT	Interrupt	3,2mA	
WAIT	Wait	3,2mA	
BAI	Busacknowledge In		
BAO	Busacknowledge Out	S12	

Belegung der VG 64 Leiste

	a		c	
+5V	1	o	1	+5V
D5	2	o	2	D0
D6	3	o	3	D7
D3	4	o	4	D2
D4	5	o	5	A0
A2	6	o	6	A3
A4	7	o	7	A1
A5	8	o	8	
A6	9	o	9	A7
WAIT	10	o	10	
	11	o	11	IEI
BAI	12	o	12	
	13	o	13	
	14	o	14	D1
	15	o	15	
	16	o	16	IEO
BAO	17	o	17	
	18	o	18	
	19	o	19	
M1	20	o	20	
	21	o	21	INT
	22	o	22	WR
	23	o	23	
	24	o	24	RD
	25	o	25	
	26	o	26	PWRCL
IORQ	27	o	27	
	28	o	28	
	29	o	29	ϕ
	30	o	30	MRQ
	31	o	31	
GND	32	o	32	GND

5.3. Stromaufnahme:

Die typische Stromaufnahme dieser Karte beträgt ca. 500mA gemessen in der 6MHz Version mit einem EPROM 2732.

6. Software:

6.1. Initialisierung des Z8038:

Die nun folgende Beschreibung der Initialisierung des Z8038 bezieht sich ausschließlich auf den Betrieb als CPU-CPU Interface mit Z80-Timing. Diese Funktion ist durch die Schaltung bereits festgelegt. Die anderen Betriebsarten dieses Bausteins wie 2- oder 3-Draht Handshake werden hier nicht behandelt. Die genaue Beschreibung ist dem Manual zu entnehmen, jedoch soll darauf hingewiesen werden, daß die Beschreibung nicht vollständig besonders bezüglich der Reihenfolge der Programmierung ist.

Das FIO hat zwei Interface-Ports, die intern über ein 128 Byte tiefes FIFO-RAM und einige Register miteinander kommunizieren können.

Jeder Interface-Port belegt zwei Adressen. Der Datenport (128 x 8 FIFO) wird mit A0 = 0 adressiert, der Kontrollport mit A0 = 1. Jeder Kontrollport besitzt 16 Kontrollregister, die teils Lese/Schreibregister, teils nur Lese- oder nur Schreibregister sind.

Der Zugriff auf ein Kontrollregister erfolgt in zwei Schritten. Zuerst wird die Adresse des Registers in den FIO-Kontrollport geschrieben. Dann kann das adressierte Register beschrieben oder gelesen werden. Das Lesen eines Registers kann nach der Adressierung beliebig oft erfolgen (Polling), das Beschreiben ist jedoch nur einmal möglich. Beim nächsten Schreibzugriff muß das Register erst wieder adressiert werden.

Die Register im einzelnen:

Kontroll-Register 0 Adresse 0H

D7	D6	D5	D4	0	1	0	D0

D0: 1 = RESET, kann von beiden Interface-Ports aus gesetzt werden. Alle Kontrollregister werden rückgesetzt. Falls der Interface-Port 1 das RESET-Bit setzt, wird auch der Interface-Port 2 zurückgesetzt, umgekehrt nicht. Der Interface-Port 2 muß vom Port 1 initialisiert werden. Wann dies geschehen ist, kann die CPU des Ports 2 feststellen, indem sie das Kontrollregister 0 ließt und auf 01H vergleicht. Ist der Port noch nicht initialisiert, so sind die Datenleitungen immer im Tristate. Durch einen Pull-Up-Widerstand kann z.B. D7 definiert auf High gelegt werden.

Erst **nachdem** das RESET-Bit rückgesetzt wurde, können andere Register geladen werden. Das gilt ebenfalls für die restlichen Bits dieses Registers.

D4: 1 = VIS, Vector Includes Status. Der Interrupt-Vector wird je nach dem jeweils anliegenden Status modifiziert. Siehe auch Interrupt-Status-Register 0 bis 3.

D5: 1 = NV, No Vector on Interrupt. Ist in einem Z80-System nicht sehr sinnvoll, entspricht dem IM1.

D6: 1 = DLC. Disable Lower Daisy-Chain. IEO wird auf Low gesetzt.

D7: 1 = MIE, Master Interrupt Enable. Gibt alle Interrupts frei. Vorher müssen die Interrupt-Status-Register beschrieben sein.

Kontroll-Register 1 Adresse 1H

D7	D6	D5	D4	D3	D2	D1	D0
0	D6	D5	D4	0	0	0	D0

D0: 1 = WAIT enabled, gibt den Wait frei. WAIT wird generiert, um die zwei CPUs zu synchronisieren. Die schreibende CPU bekommt ein WAIT-Signal wenn der FIO-Buffer voll ist, die lesende CPU bekommt WAIT wenn der Buffer leer ist.
Dieses Bit darf erst gesetzt werden, wenn die Datenübertragungsrichtung festgelegt ist. Ein Software-Fehler kann zu einem statischen WAIT-Signal führen.

D4: 1 = Message Mailbox Register under Service, Read Only Bit. Dieses Bit zeigt an, daß das Message-Register zwar noch nicht gelesen wurde, aber die jeweils andere CPU schon durch einen Interrupt von dem Vorhandensein einer Message unterrichtet wurde.

D5: 1 = Message Mailbox Register full, Read Only Bit. Schreibt eine CPU in das Messageregister, so wird dieses Bit gesetzt, ließt die jeweils andere CPU das Register aus, so wird das Bit zurückgesetzt.

D6: 1 = Freeze Status Register Count. Dieses Bit stoppt den Byte Counter, so daß das Byte Count Register korrekt gelesen werden kann. Nach dem Lesezugriff auf das Byte Count Register wird dieses Bit automatisch zurückgesetzt.

Kontroll-Register 2 Adresse 9H

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	D0

D0: 1 = Port 2 Side enabled. **Nach** der Initialisierung des Interface-Ports 1 kann dieser durch Setzen dieses Bits den Port 2 freigeben. Von der Port 2 CPU aus ist dieses Register Read Only und hat den Inhalt 0H.

Kontroll-Register 3 Adresse AH

D7	D6	D5	D4	0	0	0	0

D4: Data Direction Bit.
1 = Input to CPU, CPU will von dem Datenbuffer lesen.
0 = Output to CPU, CPU will in den Datenbuffer schreiben.
Es ist zu beachten, daß durch Bit D5 zuerst festgelegt werden muß, welche CPU die Datentransferrichtung kontrolliert.

D5: 0 = Master kontrolliert die Datentransferrichtung.
1 = Slave kontrolliert die Datentransferrichtung.

D6: Clear FIFO Buffer. Der Datenbuffer wird geleert, das Byte Count Register wird zurückgesetzt.
Es ist zu beachten, daß durch Bit D7 zuerst festgelegt werden muß, welche CPU den Datenbuffer leeren darf.
Um den Datenbuffer beschreiben zu können, muß das Bit wieder gesetzt werden.

D7: 0 = Master kontrolliert Clear FIFO Buffer.
1 = Slave kontrolliert Clear FIFO Buffer.

Das Register muß analog zum Register 0 normalerweise zweimal beschrieben werden. üblicherweise wird in der ersten Initialisierung des FIOs festgelegt, welche CPU die Datentransferrichtung und die Clear-FIFO Operation ausführen darf. Nach RESET stehen die Bits richtig für den Master.

6.1.1. Interrupt-Struktur des Z8038:

Da der Z8038 eigentlich ein Mitglied der Z8000-Familie ist, ist seine Interrupt-Struktur von der der Z80 I/Os verschieden. Der Ablauf ist allerdings wesentlich klarer und übersichtlicher als beim Z80.

Soll ein Interrupt erlaubt werden, so ist IE zu setzen. Tritt die Interrupt-Bedingung auf, so setzt der Baustein sein IP-Bit. Ein INT-Signal wird nur generiert, falls das IUS-Bit 0 ist, IEI High ist und kein Vector-Interrupt-Acknowledge-Zyklus läuft. Nach einiger Zeit generiert die CPU einen Interrupt-Acknowledge-Zyklus um den Vektor der Interrupt-Quelle zu lesen. Zu diesem Zeitpunkt setzt der Baustein sein IUS-Bit. Die Interrupt-Service-Routine muß am Ende das IUS-Bit und das IP-Bit zurücksetzen. Kann nicht Sichergestellt werden, daß eine weitere Interrupt-Bedingung für dieselbe Quelle während der Service-Routine auftritt, muß am Anfang der Service-Routine das IE-Bit zurückgesetzt werden. Am Ende der Routine wird es dann wieder gesetzt.

Das IUS-Bit übernimmt die Funktion des RETI bei Z80 I/Os. Probleme der Bussteuerung bei der Weiterleitung dieses OPCODES werden so vermieden.

Nähere Erläuterungen sind in den entsprechenden Manuals zu finden.

Interrupt Status Register 0 Adresse 2H

D7	D6	D5	0	0	0	0	0

Dieses Register verhält sich zwischen Lesen und Schreiben unterschiedlich. Als Leseregister gelten folgende Zuordnungen:

D5: 1 = Message Interrupt Pending (IP).
D6: 1 = Message Interrupt Enabled (IE).
D7: 1 = Message Interrupt Under Service (IUS).

Beschrieben wird das Register wie folgt:

D0, D1, D2, D3, D4 = 0

	D7	D6	D5
Null Code	0	0	0
Clear IP & IUS	0	0	1
Set IUS	0	1	0
Clear IUS	0	1	1
Set IP	1	0	0
Clear IP	1	0	1
Set IE	1	1	0
Clear IE	1	1	1

Interrupt Status Register 1 Adresse 3H

D7	D6	D5	D4	D3	D2	D1	D0

Dieses Register verhält sich zwischen Lesen und Schreiben unterschiedlich. Als Leseregister gelten folgende Zuordnungen:

D0: 1 = Pattern Match Flag.
D1: 1 = Pattern Match Interrupt Pending (IP).
D2: 1 = Pattern Match Interrupt Enabled (IE).
D3: 1 = Pattern Match Interrupt Under Service (IUS).

D4: 0
D5: 1 = Data Direction Change Interrupt Pending (IP).
D6: 1 = Data Direction Change Interrupt Enabled (IE).
D7: 1 = Data Direction Change Interrupt Under Service (IUS).

Das Pattern Match Flag wird gesetzt, falls das letzte Byte im Datenbuffer mit dem Inhalt des Pattern Match Registers 0DH maskiert durch das Pattern Mask Register 0EH übereinstimmt.

Beschrieben wird das Register wie folgt:
Es korrespondieren die Bit-Set und Bit-Reset Operationen für IE, IP und IUS mit den jeweiligen Bits im Lesemodus.

D0 = 0, D4 = 0

	D7	D6	D5
Null Code	0	0	0
Clear IP & IUS	0	0	1
Set IUS	0	1	0
Clear IUS	0	1	1
Set IP	1	0	0
Clear IP	1	0	1
Set IE	1	1	0
Clear IE	1	1	1

	D3	D2	D1
Null Code	0	0	0
Clear IP & IUS	0	0	1
Set IUS	0	1	0
Clear IUS	0	1	1
Set IP	1	0	0
Clear IP	1	0	1
Set IE	1	1	0
Clear IE	1	1	1

Interrupt Status Register 2 Adresse 4H

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0

Dieses Register verhält sich zwischen Lesen und Schreiben unterschiedlich. Als Leseregister gelten folgende Zuordnungen:

D0: 1 = Underflow Error.
D1: 1 = Error Interrupt Pending (IP).
D2: 1 = Error Interrupt Enabled (IE).
D3: 1 = Error Interrupt Under Service (IUS).
D4: 1 = Overflow Error

D5: 1 = Byte Count Compare Interrupt Pending (IP).
D6: 1 = Byte Count Compare Interrupt Enabled (IE).
D7: 1 = Byte Count Compare Interrupt Under Service (IUS).

Beschrieben wird das Register wie folgt:
Es korrespondieren die Bit-Set und Bit-Reset Operationen für IE, IP und IUS mit den jeweiligen Bits im Lesemodus.

D0 = 0, D4 = 0

	D7	D6	D5
Null Code	0	0	0
Clear IP & IUS	0	0	1
Set IUS	0	1	0
Clear IUS	0	1	1
Set IP	1	0	0
Clear IP	1	0	1
Set IE	1	1	0
Clear IE	1	1	1

	D3	D2	D1
Null Code	0	0	0
Clear IP & IUS	0	0	1
Set IUS	0	1	0
Clear IUS	0	1	1
Set IP	1	0	0
Clear IP	1	0	1
Set IE	1	1	0
Clear IE	1	1	1

Interrupt Status Register 3 Adresse 5H

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0

Dieses Register verhält sich zwischen Lesen und Schreiben unterschiedlich. Als Leseregister gelten folgende Zuordnungen:

D0: 1 = Buffer Empty.
D1: 1 = Empty Interrupt Pending (IP).
D2: 1 = Empty Interrupt Enabled (IE).
D3: 1 = Empty Interrupt Under Service (IUS).

D4: 1 = Buffer Full.
D5: 1 = Full Interrupt Pending (IP).
D6: 1 = Full Interrupt Enabled (IE).
D7: 1 = Full Interrupt Under Service (IUS).

Beschrieben wird das Register wie folgt:
Es korrespondieren die Bit-Set und Bit-Reset Operationen für IE, IP und IUS mit den jeweiligen Bits im Lesemodus.

D0 = 0, D4 = 0

	D7	D6	D5
Null Code	0	0	0
Clear IP & IUS	0	0	1
Set IUS	0	1	0
Clear IUS	0	1	1
Set IP	1	0	0
Clear IP	1	0	1
Set IE	1	1	0
Clear IE	1	1	1

	D3	D2	D1
Null Code	0	0	0
Clear IP & IUS	0	0	1
Set IUS	0	1	0
Clear IUS	0	1	1
Set IP	1	0	0
Clear IP	1	0	1
Set IE	1	1	0
Clear IE	1	1	1

Message Out Register Adresse BH

Das in dieses Register geschriebene Byte kann von der jeweils anderen CPU im Message In Register abgeholt werden. Das Register kann beschrieben und gelesen werden.

Message In Register Adresse CH

Hält das Byte, das die jeweils andere CPU in ihr Message Out Register geschrieben hat. Das Register kann nur gelesen werden.

Die zwei letztgenannten Register können zum Austausch von Parametern zwischen den CPUs benutzt werden.

6.2. SLAVE-Bootstrap-Loader Rev. 1.0:

Die Verbindung von Slave-CPU und CPU erfolgt durch den Kommunikationsbaustein Z8038-FIO, der u.a. einen 128 Byte-Puffer enthält und von zwei CPUs als I/O-Port angesprochen werden kann.

Im Gegensatz zu einem Hauptrechner, der RESET durch Knopfdruck oder beim Einschalten erhält und zunächst ein Betriebssystem von Diskette läd, wird der Nebenrechner durch Befehle des Hauptrechners gesteuert und erhält alle zum Betrieb benötigten Programme und Daten von diesem.

Hier wird die Funktion des Urladers der Slave-CPU beschrieben.

6.2.1. Kommunikationsprinzip:

Legende:

```
-------> und <------ : übertragung per Message-Register
=======> und <====== : übertragung per Puffer
```

```
        <==========> 128 Byte Puffer <==========>
MASTER  -------> 1 Byte Message Register --------> SLAVE
        <------- 1 Byte Message Register <--------
```

Bei der Kommunikation wie sie im Urlader 1.x verwirklicht ist, werden alle Befehle und Meldungen über die Message-Register übertragen. Das Eintreffen einer Nachricht im Empfangsregister und auch die "Abholung" einer Nachricht im Senderegister können abgefragt oder über Interrupts verarbeitet werden.

Daten werden ausschließlich über den Puffer übertragen. Es gibt sowohl Möglichkeiten zur Abfrage des Füllungsgrades als auch automatische Synchronisation durch WAIT als auch die Möglichkeit, bei Overflow, Underflow oder sonstigen Füllungsgraden Interrupts auszulösen.
Im Boot 1.x wurden ausschließlich Abfrageverfahren und automatische Synchronisation verwandt. Dies schränkt jedoch nicht die Möglichkeiten auf der Masterseite ein.

6.2.2. Kommunikationsprotokoll:

Initialisierung auf Masterseite:

1. RESET des Nebenrechners auslösen (IN-Befehl).

2. Initialisieren der Z8038-FIO (s. techn. Beschreibung)

3. Auslösen eines NMI (Nicht Maskierbarer Interrupt) auf der Slave-CPU (IN-Befehl. Daraufhin initialisiert der Bootstrap der Slave-CPU ihre Seite der FIO und meldet Empfangsbereitschaft.

6.2.3. Betrieb:

Der Urlader der Slave-CPU verschickt nur drei verschiedene Meldungen an den Hauptrechner:

READY = 00 = Empfangsbereitschaft / letztes Kommando ausgeführt
EXECUTING = 01 = Kommando empfangen / Ausführung läuft
ERROR = FF = Fehler

Darüber hinaus werden auf Anforderung auch Daten übertragen.

Alle diese Meldungen werden über Message-Register verschickt. Auf das Auslesen durch die Master-CPU wird erst dann gewartet, wenn eine neue Meldung abgeschickt werden muß.

6.2.4.Befehle:

Der READY-Befehl wird stets am Ende der Sequenz genannt. Dies ist eine inhaltlich richtige Zuordnung, da die READY-Meldung den Abschluß des Befehls kennzeichnet. Im Betrieb sollte die READY-Meldung jedoch erst am Anfang der darauffolgenden Befehlssequenz abgeholt werden, da man sonst zu einem abwechselnden Warten der CPUs kommt! Beachten Sie bitte den READY-Befehl, der beim Abschluß der Initialisierung ausgesandt wird.

Numerische Angaben sind Hexadezimal.

ll steht für low Byte
hh steht für high Byte

Die Masterseite ist in den Abbildungen stets links.

NULL: Dient zum Prüfen der Verständigung. Bewirkt nichts.

```
00 ---->
      <---- 01 (EXECUTING)
      <---- 00 (READY)
```

REVISION: Fragt nach der Revision des Urladers. Die Revision hat die Form "x.y". Beide Bytes sind binär zu verstehen. Programme, die legal mit dem Urlader kommunizieren, müßen nicht geändert werden, solange das "x" sich nicht ändert.

```
01 ---->
      <---- 01 (EXECUTING)
      <---- x  (Revision Teil 1)
      <---- y  (Revision Teil 2)
      <---- 00 (READY)
```

LADERLÄNGE: Der Urlader kann mit allen wesentlichen Teilen auf jeder Adresse außerhalb des EPROMs laufen. Er kann beliebig oft verschoben werden. Dazu muß man u.U. die Länge dieses Urladerteils wissen.

```
        02 ---->
             <---- 01 (EXECUTING)
             <---- ll (low byte)
             <---- hh (high byte)
             <---- 00 (READY)
```

EPROMLÄNGE: Die Länge des EPROMs ist beim Verschieben des Laders wichtig. Interfaceprogramme dürfen nicht von einer festen Länge des EPROMs ausgehen.

```
        03 ---->
             <---- 01 (EXECUTING)
             <---- ll (low byte)
             <---- hh (high byte)
             <---- 00 (READY)
```

RELOKATIEREN: Verschieben des Laders. Alle Befehle außer den explizit genannten können auch auf der neuen Laufadresse bearbeitet werden - also insbes. auch dieser Befehl.

```
                04 ---->
                     <---- 01 (EXECUTING)
(low byte)      ll ---->
(high byte)     hh ---->
                     <---- 00 (READY)
```

EPROM AUSBLENDEN: Zum Laden des Speicherbereiches, der parallel zum EPROM liegt, muß dieser ausgeblendet werden.

```
        05 ---->
             <---- 01 (EXECUTING)
             <---- 00 (READY)
```

RÜCKSPRUNG IN DAS EPROM: Das EPROM wird zunächst eingeblendet und danach angesprungen.

```
        06 ---->
             <---- 01 (EXECUTING)
             <---- 00 (READY)
```

START EINES PROGRAMMS: übergabe der Kontrolle an ein geladenes Programm.

```
        07 ---->
             <---- 01 (EXECUTING)
(startadr. low b.) ll ---->
(    "     high ") hh ---->
                          (Keine READY-Meldung)
```

START EINES PROGRAMMES MIT ABSCHALTEN DES EPROMS:

Wenn im EPROM gearbeitet wird, der Lader aus Raummangel nicht verschoben werden kann und die Startadresse im zum EPROM parallelen Speicher liegt, so kann mit diesem Befehl die Ausführung des Programms trotzdem gestartet werden, wenn irgendwo oberhalb des EPROMs noch mindestens 4 Byte frei sind.

```
                    08 ----->
                         <---- 01 (EXECUTING)
(Hilfsadr. low b.)ll ----->
(    "    high " )hh ----->
(Sprungadr. low" )ll ----->
(    "     high" )hh ----->
                                  (Keine READY-Meldung)
```

LADEN: Laden des Zweitrechnerspeichers ab einer angegebenen Adresse. Die Übertragung endet durch Senden eines beliebigen Bytes über das Message-Register. Die Übertragung der Daten erfolgt durch den Puffer!

```
                    09 ----->
                         <---- 01 (EXECUTING)
(Ladeadresse)       ll ----->
                    hh ----->
("Bytes")           xx ====>
                    xx ====>
                    ...
(Ende)              ** ----->
                         <---- 00 (READY)
```

SPEICHERTEST: Diese Routine kann nur vom EPROM aus gerufen werden. Sie zerstört den gesamten RAM-Inhalt. Das Testverfahren besteht im Laden des gesamten Speichers mit Zufallsbits und anschließendem Vergleich. (Der benutzte Zufallszahlengenerator wird in Knuth: Seminumerical Algorithms besprochen und hat eine Periode von mind. 2**55.)

```
                    0A ----->
                         <---- 01 (EXECUTING)
                         <---- 00 oder FF (OK oder NOT OK)
                         <---- 00 (READY)
```

ANDERE BEFEHLE SIND ILLEGAL:

```
                    zz ----->
                         <---- 01 (EXECUTING)
                         <---- FF (ERROR)
                         <---- 00 (READY)
```

```
'SLAVE BOOTSTRAP LOADER REV 1.0'       MACRO-80 3.44   09-Dec-81       PAGE    1


                                .Z80
                                NAME    ('SLBOOT')
                                TITLE   'SLAVE BOOTSTRAP LOADER REV 1.0'
0000'                           ASEG
                                ORG     0
                                .SALL
                        ;
                        ;---------------------------------------------------------------
                        ; REVISION
0001                    REV1    EQU     1       ; REVISION PART 1 (1 BYTE)
0000                    REV2    EQU     0       ; REVISION PART 2 (1 BYTE)
                        ;-------
                        ; EPROM SIZE
                        ;ROMBYT EQU     800H    ; 2716
1000                    ROMBYT  EQU    1000H    ; 2732  STANDARD
                        ;-------
                        ; FIO PORTS
00F8                    FIO     EQU     0F8H    ; Z8038-FIO-PORTS
                                                ; +0 : DATA
                                                ; +1 : CONTROL/STATUS
                                                ; +2 : BOOT EPROM ON
                                                ; +3 : BOOT EPROM OFF
                        ;---------------------------------------------------------------
                    +   PUTFIO  MACRO   FIOREG,VALU     ; SET FIO REGISTER TO VALUE
                    +           LD      A,FIOREG
                    +           OUT     (FIO+1),A
                    +           LD      A,VALU
                    +           OUT     (FIO+1),A
                                ENDM
                        ;-------
                    +   GETFIO  MACRO   FIOREG,REG      ; GET FIO REGISTER INTO CPU-REGISTER
                    +           LD      A,FIOREG
                    +           OUT     (FIO+1),A
                    +           IN      A,(FIO+1)
                    +           COND    '&REG' NE 'A'
                    +           LD      REG,A
                    +           ENDC
                                ENDM
                        ;-------
                    +   WAIT    MACRO   FIOREG,BITNR,VALU       ; WAIT FOR BIT OF FIO REG TO BE 0/1
                    +           LOCAL   AGAIN
                    +           LD      A,FIOREG
                    +           OUT     (FIO+1),A
                    +   AGAIN:  IN      A,(FIO+1)
                    +           BIT     BITNR,A
                    +           COND    VALU EQ 0
                    +           JR      NZ,AGAIN
                    +           ELSE
                    +           JR      Z,AGAIN
                    +           ENDC
                                ENDM
                        ;-------
                    +   SEND    MACRO   MSG     ; WAIT UNTIL LAST MESSAGE READ - SEND NEW MESSAGE
                    +           WAIT    REGC1,5,0       ; WAIT UNTIL BYTE READ BY MASTER
                    +           PUTFIO  REGOUT,<MSG>    ; SEND BYTE TO MASTER
                                ENDM
```

'SLAVE BOOTSTRAP LOADER REV 1.0' MACRO-80 3.44 09-Dec-81 PAGE 1-1

```
                        ;-------
                  +     RECEIV  MACRO   REG     ; RECEIVE MESSAGE FROM MASTER
                  +             WAIT    REGI0,5,1       ; WAIT FOR MESSAGE FROM MASTER
                  +             GETFIO  REGIN,A         ; READ MESSAGE
                  +             COND    '&REG' NE 'A'
                  +             LD      REG,A
                  +             ENDC
                                ENDM
                        ;-------
                  +     JUMP    MACRO   LBL             ; JUMP RELOCATABLE ON ZERO FLAG
                  +             EXX
                  +             LD      IX,LBL-START
                  +             ADD     IX,DE
                  +             EXX
                  +             JP      (IX)
                                ENDM
                        ;----------------------
                  +     JUMPZ   MACRO   LBL             ; JUMP RELOCATABLE ON ZERO
                  +             LOCAL   L1
                  +             JR      NZ,L1
                  +             JUMP    LBL
                  +     L1:
                                ENDM
                        ;----------------------
                        ; FIO-REGISTER
0000                    REGC0   EQU     0       ; KONTROLLREGISTER 0
0001                    REGC1   EQU     1       ;       "          1
0009                    REGC2   EQU     9       ;       "          2
000A                    REGC3   EQU     10      ;       "          3
0002                    REGI0   EQU     2       ; INTERRUPT STATUS REGISTER 0
0003                    REGI1   EQU     3       ;       "       "       "   1
0004                    REGI2   EQU     4       ;       "       "       "   2
0005                    REGI3   EQU     5       ;       "       "       "   3
0006                    REGIV   EQU     6       ; INTERRUPT VECTOR REGISTER
0007                    REGN    EQU     7       ; BYTE COUNT REGISTER
000D                    REGP    EQU     13      ; PATTERN MATCH REGISTER
000E                    REGPM   EQU     14      ; PATTERN MATCH MASK REGISTER
000F                    REGLST  EQU     15      ; DATA BUFFER REGISTER (LAST DATA BYTE)
0008                    REGBC   EQU     8       ; BYTE COUNT COMPARISION REGISTER
0008                    REGOUT  EQU     11      ; MESSAGE OUT REGISTER
000C                    REGIN   EQU     12      ; MESSAGE IN REGISTER
                        ;----------------------
                        ; MESSAGES FOR OUTPUT
0000                    READYB  EQU     00H     ; READY TO ACCEPT COMMANDS
0001                    EXECUT  EQU     01H     ; EXECUTING A COMMAND
00FF                    BERROR  EQU     0FFH    ; ERROR (FATAL IF NOT FOLLOWED BY READYB)
                        ;----------------------
                        ; MESSAGES IN INPUT
0000                    NOCOM   EQU     00H     ; NO COMMAND / COMPLETE LOAD
0001                    SNDREV  EQU     01H     ; SEND REVISION (REV1 BEFORE REV2)
0002                    SNDLDR  EQU     02H     ; SEND LOADER LENGTH(LO BEFORE HI)
0003                    SNDEPR  EQU     03H     ; SEND EPROM LENGTH(LO BEFORE HI)
0004                    RUNAT   EQU     04H     ; ROLOCATE YOURSELF TO (LO BEFORE HI)
0005                    EPROFF  EQU     05H     ; SWITCH OFF EPROM
0006                    EPRJMP  EQU     06H     ; SWITCH ON EPROM AND JUMP TO START
0007                    JUMPTO  EQU     07H     ; JUMP TO ADRESS (LO BEFORE HI)
```

'SLAVE BOOTSTRAP LOADER REV 1.0' MACRO-80 3.44 09-Dec-81 PAGE 1-2

```
0008                    JUMPIN  EQU     08H     ; LOAD AT FIRST ADRESS(LO BEFORE HI) 4 BYTE CODE
                                                ; TO SWITCH OFF EPROM AND JUMP TO SECOND ADRESS
0009                    LOADAT  EQU     09H     ; LOAD AT ADRESS (LO BEFORE HI) ALL BYTES
                                                ; COMING IN VIA BUFFER UNTIL MASTER GIVES
                                                ; NEXT COMMAND. DUE TO A FIO-MASK-ERROR THE
                                                ; BUFFER MAY NOT BECOME EMPTY WHILE THE MASTER
                                                ; WRITES, SO READING WILL BE COMPLETED WHEN
                                                ; THE NEXT COMMAND FOLLOWS. THAT COMMAND MUST NOT
                                                ; BE FOLLOWOWED BY WRITING INTO THE BUFFER!
000A                    MEMTST  EQU     0AH     ; MEMORY TEST (USE BEFORE RELOCATION)
                        ;-------------------------------------------------------------------
                        ; RESET-ENTRY
0000    31 4000                 LD      SP,4000H
0003    11 007C                 LD      DE,START
0006    D9                      EXX
0007    76                      HALT                    ; WAIT FOR NMI
                                                        ; MASTER WILL ISSUE NMI WHEN HE HAS INITIALIZED
                                                        ; HIS SIDE OF FIO
                      +         JUMP    START
                                .LIST
                        ;-------------------------------------------------------------------
                        ; NON-MASKABLE-INTERRUPT
                        ;       ASSUME THAT MASTER HAS INITIALIZED HIS PART OF THE FIO
0066                    NMI:    ; LOCATION IS 66H
                                ; INITIALIZE FIO
0066    3E 00                   LD      A,0             ; RESET OFF
0068    D3 F9                   OUT     (FIO+1),A
                      +         PUTFIO  REGC0,00010000B ; NO INTERRUPTS, VECTOR INCLUDES STATUS
                      +         PUTFIO  REGC1,00000001B ; WAIT ON FULL/EMPTY
007A    ED 45                   RETN
                        ;-------------------------------------------------------------------
                        ; LOADER
                        ;       BOTH SIDES OF FIO ARE INITIALIZED
                        ;       COMMUNICATION EXCEPT LOADING IS DONE THROUGH MESSAGE REGISTERS OF
                        ;       FIO IN THE FOLLOWING SEQUENCE:
                        ;               1.      SLAVE:  READYB
                        ;               2.      MASTER: (ANY COMMAND)
                        ;               3.      SLAVE:  EXECUTING
                        ;               ...     ...
                        ;
007C                  + START:  SEND    READYB
                      +         RECEIV  B               ; FETCH COMMAND
                      +         SEND    EXECUT
00B1    78                      LD      A,B     ; A= COMMAND NUMBER
00B2    B7                      OR      A
                      +         JUMPZ   START
00BF    FE 01                   CP      SNDREV
                      +         JUMPZ   SNDR
00CD    FE 02                   CP      SNDLDR
                      +         JUMPZ   SNDL
00DB    FE 03                   CP      SNDEPR
                      +         JUMPZ   SNDE
00E9    FE 04                   CP      RUNAT
                      +         JUMPZ   RUNA
00F7    FE 05                   CP      EPROFF
                      +         JUMPZ   EPRO
```

```
0105  FE 06                    CP      EPRJMP
                      +         JUMPZ   EPR1
0113  FE 07                    CP      JUMPTO
                      +         JUMPZ   JUMPT
0121  FE 08                    CP      JUMPIN
                      +         JUMPZ   JUMPI
012F  FE 09                    CP      LOADAT
                      +         JUMPZ   LOAD
013D  FE 0A                    CP      MEMTST
                      +         JUMPZ   MEMT
                      +         SEND    BERROR
                      +         JUMP    START           ; DO NOTHING COMMAND
                          ;----------------------
                          ; SEND REVISION OF BOOT
0167                  +   SNDR:  SEND    REV1
                      +         SEND    REV2
                      +         JUMP    START
                          ;----------------------
                          ; SEND LENGTH OF LOADER
0195                  +   SNDL:  SEND    (LOW LDRLEN)
                      +         SEND    (HIGH LDRLEN)
                      +         JUMP    START
                          ;----------------------
                          ; SEND LENGTH OF EPROM
01C3                  +   SNDE:  SEND    (LOW ROMBYT)
                      +         SEND    (HIGH ROMBYT)
                      +         JUMP    START
                          ;----------------------
                          ; RELOCATE LOADER AND CONTINUE EXECUTION AT NEW ADRESS
01F1  D9                      RUNA:  EXX                     ; DE' CONTAINS ACTUAL START
01F2  EB                             EX      DE,HL           ; SOURCE
                      +              RECEIV  E               ; GET LOW BYTE DESTINATION
                      +              RECEIV  D               ; GET HIGH BYTE DESTINATION
0215  01 02A5                        LD      BC,LDRLEN       ; BYTE COUNT
0218  ED B0                          LDIR                    ; MOVE LOADER
021A  EB                             EX      DE,HL           ; RESTORE DE'
021B  11 FD5B                        LD      DE,-LDRLEN
021E  19                             ADD     HL,DE
021F  EB                             EX      DE,HL
0220  DD 21 0000                     LD      IX,0            ; JUMP TO START DONE USING IX
0224  DD 19                          ADD     IX,DE
0226  D9                             EXX
0227  DD E9                          JP      (IX)
                          ;----------------------
                          ; SWITCH OFF EPROM
0229  DB FB                   EPR0:  IN      A,(FIO+3)
                      +              JUMP    START
                          ;----------------------
                          ; SWITCH ON EPROM AND JUMP TO START
0235  DB FA                   EPR1:  IN      A,(FIO+2)
0237  D9                             EXX
0238  11 007C                        LD      DE,START
023B  D9                             EXX
023C  C3 007C                        JP      START   ; NOTICE THE ABSOLUTE JUMP
                          ;----------------------
                          ; TRANSFER CONTROL TO USER PROGRAM
```

'SLAVE BOOTSTRAP LOADER REV 1.0' MACRO-80 3.44 09-Dec-81 PAGE 1-4

```
023F                    +     JUMPT:  RECEIV  L
                        +             RECEIV  H
0261    E9                            JP      (HL)
                              ;----------------------
0262                          JUMP1:                  ; JUMP FROM EPROM TO CUNCURRENT AREA
                                                      ;       STEP 1: PUT CODE BEYOND EPROM TO SWITCH IT OFF
                                                      ;               AND JUMP TO THE SPECIFIED ADRESS
                                                      ;       STEP 2: JUMP INTO THAT CODE
                                                      ; BOTH ADRESSES ARE TO BE GIVEN BY THE MASTER-CPU
                        +             RECEIV  L       ; GET JUMP CODE ADRESS
                        +             RECEIV  H
0284    5D                            LD      E,L
0285    54                            LD      D,H
0286    36 DB                         LD      (HL),0DBH       ; IN-COMMAND
0288    23                            INC     HL
0289    36 F8                         LD      (HL),F10+3      ; PORT FOR EPROM-OFF
028B    23                            INC     HL
028C    36 FD                         LD      (HL),0FDH       ; JP (IY) - COMMAND
028E    23                            INC     HL              ; .
028F    36 E9                         LD      (HL),0E9H       ; .
0291    EB                            EX      DE,HL
                        +             RECEIV  E               ; LOW BYTE OF DEST. ADRESS
                        +             RECEIV  D               ; HIGH BYTE OF DEST. ADRESS
02B4    FD 21 0000                    LD      IY,0            ; IY=DESTINATION
02B8    FD 19                         ADD     IY,DE           ; .
02BA    E9                            JP      (HL)            ; NOW JUMP INDIRECT
                              ;----------------------
                              ; LOAD AT LOCATION GIVEN IN MESSAGE REG
02BB                    +     LOAD:   RECEIV  L
                        +             RECEIV  H
02DD    0E F8                         LD      C,F10+0         ; DEFINE PORT NUMBER
02DF                    +     LOAD1:  PUTFIO  REGC1,01000001B ; REPEAT FREEZE BYTE COUNT
                        +             GETFIO  REGN,A
02ED    B7                            OR      A
02EE    28 06                         JR      Z,LOAD2         ; IF COUNT=0 OR
02F0    3D                            DEC     A
02F1    28 03                         JR      Z,LOAD2         ;    COUNT=1 THEN OMIT READING
02F3    47                            LD      B,A             ; GET BYTES EXCEPT LAST ONE
02F4    ED B2                         INIR
02F6                    +     LOAD2:  GETFIO  REGI0,A
02FC    CB 6F                         BIT     5,A             ; MESSAGE FROM MASTER?
02FE    28 0F                         JR      Z,LOAD1         ; NO: CONTINUE
                        +             PUTFIO  REGC1,01000001B ; FREEZE BYTE COUNT AGAIN
                        +             GETFIO  REGN,B          ; GET BYTE COUNT
030F    ED B2                         INIR                    ; READ ALL OF THE BYTES
                        +             GETFIO  REGIN,A ; IGNORE TERMINATION BYTE
                        +             JUMP    START           ; NOW EXECUTE COMMAND FOUND IN MESSAGE REGISTER
                              ;-------------------------------------------------------------------------------
0321                          LDREND:         ; PARTS BEYOND THIS LABEL ARE NOT TO BE RELOCATED
                              ;------------------------------
0321                          MEMT:                   ; MEMORYTEST X(N):=(X(N+C1)+X(N+C2)) MOD 2**16; N DEC.
0018                          MEMTC1  EQU     24D     ; DONT CHANGE CONSTANTS RANDOMLY
0037                          MEMTC2  EQU     55D     ; (CF. KNUTH: SEMINUMERICAL ALGORITHMS)
                                                      ; PART OF BOOT CODE USED FOR INITIAL VALUES
0321    01 FFFF                       LD      BC,-1           ; DESTINATION X(N) (0FFFFH)
0324    11  0017                      LD      DE,MEMTC1-1     ; X(N+C1)
```

'SLAVE BOOTSTRAP LOADER REV 1.0' MACRO-80 3.44 09-Dec-81 PAGE 1-5

```
0327  21 0036                    LD      HL,MEMTC2-1     ; X(N+C2)
032A  1A                 MEMT1:   LD      A,(DE)          ; X(N) <- X(N+C1) + X(N+C2)
032B  86                          ADD     A,(HL)          ; .
032C  02                          LD      (BC),A          ; .
032D  2B                          DEC     HL              ; N <- N-1
032E  1B                          DEC     DE              ; .
032F  0B                          DEC     BC              ; .
0330  3E 10                       LD      A,HIGH ROMBYT   ; LAST N ?
0332  A8                          XOR     B               ; .
0333  20 F5                       JR      NZ,MEMT1        ; NO --> MEMT1
0335  3E 00                       LD      A,LOW ROMBYT    ; .
0337  A9                          XOR     C               ; .
0338  20 F0                       JR      NZ,MEMT1        ; NO --> MEMT1
033A  01 FFFF                     LD      BC,-1           ; NOW START COMPARISON
033D  11 0017                     LD      DE,MEMTC1-1
0340  21 0036                     LD      HL,MEMTC2-1
0343  0A                 MEMT2:   LD      A,(BC)          ; X(N) - X(N+C1) - X(N+C2) SHOULD BE 0
0344  96                          SUB     (HL)
0345  EB                          EX      DE,HL
0346  96                          SUB     (HL)
0347  EB                          EX      DE,HL
0348  20 29                       JR      NZ,MEMT3        ; ISN'T ZERO --> MEMT3 (ERROR)
034A  2B                          DEC     HL              ; N <- N-1
034B  1B                          DEC     DE              ; .
034C  0B                          DEC     BC              ; .
034D  3E 10                       LD      A,HIGH ROMBYT   ; LAST N ?
034F  A8                          XOR     B
0350  20 F1                       JR      NZ,MEMT2        ; NO --> MEMT2
0352  3E 00                       LD      A,LOW ROMBYT
0354  A9                          XOR     C
0355  20 EC                       JR      NZ,MEMT2        ; NO --> MEMT2
                    +             SEND    READYB          ; HAPPY END: SEND MESSAGE AND FINISH
                    +             JUMP    START           ; .
0373                +    MEMT3:   SEND    BERROR          ; MEMORY FAILED: SEND MESSAGE AND FINISH
0385  C3 007C                     JP      START           ; . (ABSOLUTE JUMP!)
                         ;-----------------------------
02A5                     LDRLEN   EQU     LDREND-START            ; REAL LENGTH OF LOADER
0388                     LAST:                                    ; LENGTH OF BOOTSTRAP
                                  .LIST
                                  END
```

'SLAVE BOOTSTRAP LOADER REV 1.0' MACRO-80 3.44 09-Dec-81 PAGE 5

Macros:

GETFIO JUMP JUMPZ PUTFIO RECEIV
SEND WAIT

Symbols:

0080	..0000	0092	..0001	00A3	..0002
00BF	..0003	00CD	..0004	00DB	..0005
00E9	..0006	00F7	..0007	0105	..0008
0113	..0009	0121	..000A	012F	..000B
013D	..000C	014B	..000D	014F	..000E
016B	..000F	017D	..0010	0199	..0011
01AB	..0012	01C7	..0013	01D9	..0014
01F7	..0015	0208	..0016	0243	..0017
0254	..0018	0266	..0019	0277	..001A
0296	..001B	02A7	..001C	02BF	..001D
02D0	..001E	035B	..001F	0377	..0020
00FF	BERROR	0229	EPR0	0235	EPR1
0006	EPRJMP	0005	EPROFF	0001	EXECUT
00F8	FIO	0262	JUMPI	0008	JUMPIN
023F	JUMPT	0007	JUMPTO	0388	LAST
0321	LDREND	02A5	LDRLEN	02BB	LOAD
02DF	LOAD1	02F6	LOAD2	0009	LOADAT
0321	MEMT	032A	MEMT1	0343	MEMT2
0373	MEMT3	0018	MEMTC1	0037	MEMTC2
000A	MEMTST	0066	NMI	0000	NOCOM
0000	READYB	0008	REGBC	0000	REGC0
0001	REGC1	0009	REGC2	000A	REGC3
0002	REGI0	0003	REGI1	0004	REGI2
0005	REGI3	000C	REGIN	0006	REGIV
000F	REGLST	0007	REGN	000B	REGOUT
000D	REGP	000E	REGPM	0001	REV1
0000	REV2	1000	ROMBYT	01F1	RUNA
0004	RUNAT	01C3	SNDE	0003	SNDEPR
0195	SNDL	0002	SNDLDR	0167	SNDR
0001	SNDREV	007C	START		

No Fatal error(s)

7. SLAVE-Down-Load-Programm:

Das Programm SDOWNLOD wird ohne Parameter gerufen und lädt eine anzugebende Datei auf eine anzugebende Adresse innerhalb des Slave-Rechners und startet dieses Programm bei abgeschaltetem EPROM auf einer ebenfalls anzugebenden Adresse.

Das Programm compiliert unter PASCAL MT+ von Digital Research.

```
(*$E-*) PROGRAM slave_down_load;
(*/*)
(*/ EXAMPLE-PROGRAM: EXECUTE COM-FILE IN SLAVE *)
(*/*)

  CONST
    revision_needed = 1 (* To check compatibility *) ;
  TYPE
         sector = ARRAY [ 0 .. 127 ] OF byte;
  VAR
           rev1,
           rev2 : byte (* Slave Boot Revision *) ;
    size_of_eprom,
    size_of_loader : word;
        progname : STRING [ 14 ] ;
               f : FILE OF sector;
     load_adress : word;
    start_adress : word;

  EXTERNAL PROCEDURE master_init;
  EXTERNAL PROCEDURE no_command;
  EXTERNAL PROCEDURE get_revision( VAR part1, part2 : byte);
  EXTERNAL FUNCTION loader_length : word;
  EXTERNAL FUNCTION eprom_length : word;
  EXTERNAL PROCEDURE relocate_to(adress : word);
  EXTERNAL PROCEDURE eprom_off;
  EXTERNAL PROCEDURE eprom_again;
  EXTERNAL PROCEDURE jump_to(adress : word);
  EXTERNAL PROCEDURE jump_indirect(four_byte_adress : word; adress : word);
  EXTERNAL PROCEDURE load( VAR location : word; VAR contents : sector);
  EXTERNAL FUNCTION memory_ok : boolean;

  PROCEDURE abort(msg : STRING );
    BEGIN (*'abort'*)
      WRITELN ;
      WRITELN (msg);
      INLINE ('JMP/0/0)
    END (*'abort'*) ;


  BEGIN
(*/*) (*'slave_down_load'*)
    WRITELN ;
    WRITELN ('SLAVE-CPU LADER' : 50);
    WRITELN ;
(*/*)
```

```
    masterinit;
    no_command;
    get_revision(rev1, rev2);
    WRITELN ('Verbindung zum Bootstrap Rev. ', rev1 : 1, '.', rev2 : 1,
    ' aufgenommen.');
    IF revision_needed <> rev1
      THEN abort('Lader nicht Kompatibel');
(*/*)
    WRITE ('Speicher im Slave-Rechner ');
    IF NOT memory_ok
      THEN abort(' ******* FEHLERHAFT ! *******');
    WRITELN ('ok (64K)');
    WRITELN ;
(*/*)
    WRITE ('Welches Programm ist zu laden? ' : 50);
    READLN (progname);
    assign(f, progname);
    reset(f);
    IF ioresult = 255
      THEN abort(concat('Datei ', progname, ' nicht gefunden!'));
    IF eof(f)
      THEN abort(concat('Datei ', progname, ' leer!'));
(*/*)
    WRITE ('Ladeadresse (hex): ' : 50);
    readhex(input, load_adress, 2);
    READLN ;
    WRITE ('Startadresse (hex): ' : 50);
    readhex(input, start_adress, 2);
    READLN ;
    IF load_adress > start_adress
      THEN abort('Start aus nichtgeladenem Teil???');
(*/*)
    size_of_loader := loader_length;
    size_of_eprom := eprom_length;
(*/*)
    relocate_to(wrd($FFFF)-size_of_loader+wrd(1)) (* exec in high mem *) ;
    eprom_off;
    IF load_adress >= size_of_loader
      THEN (* run at low memory *)
        BEGIN (*"then"*)
          relocate_to(wrd(0));
          WHILE NOT eof(f) DO
            BEGIN (*"while"*)
              IF load_adress -wrd(1) > wrd($FF7F)
                THEN abort('Programm zu gross(L)');
              load(load_adress, f^);
              get(f)
            END (*"while"*);
jumpto(start_adress)
        END (*"then"*)
      ELSE (* run at high memory and perhaps later in eprom again *)
        BEGIN (*"else"*)
          WHILE NOT (eof(f) OR (load_adress > wrd($FF80)-size_of_loader)) DO
            BEGIN (*"while"*)
              load(load_adress, f^);
              get(f)
            END (*"while"*) ;
```

```
        IF eof(f)
          THEN jumpto(start_adress)
          ELSE
            BEGIN (*"else"*)
              eprom_again;
              REPEAT
                IF load_adress ) wrd($FF7C)
                  THEN abort('Programm zu gross(E)');
                load(load_adress, f^);
                get(f)
              UNTIL eof(f);
              jump_indirect(wrd($FFFC), start_adress)
            END (*"else"*)
      END (*"else"*);
        WRITELN ('Program loaded and started')
  END (*"slave_down_load"*) .
```

7.1. SLAVE-CPU-Bootstrap-Interface (SINTERF):

SINTERF ist ein PASCAL MT+ Modul zur Ansteuerung des Bootstrap SLAVBOOT von Pascal-Programmen von der Masterseite aus. Es beinhaltet die folgenden Einsprungpunkte:

PROCEDURE master_init
- Reset des Slave-Rechners
- Initialisieren der Masterseite der FIO
  - Master Controls anything
  - Clear Buffer
  - No Interrupts
  - Hardwarebeschreibung
- Auslösen eines NMI auf dem Slave-Rechner.

Alle nun folgenden Kommandos warten zunächst die Fertigmeldung des vorhergehenden Kommandos ab und geben die Kontrolle an das rufende Programm zurück ohne die Fertigmeldung des Slave-Rechners abzuwarten. Falsche und extrem verzögerte Meldungen des Slave führen zum Abbruch außer beim Speichertest.

PROCEDURE no_command
Prüft die Verständigung und kann auch zur Synchronisation verwendet werden, da ein Aufruf von no_command nach einem anderen Aufruf zum Warten auf die Fertigmeldung des letzteren führt.

PROCEDURE get_revision(VAR part1, part2 : byte)
Liefert die Revision "part1.part2". Programme sollten stets auf ein festes "part1" prüfen.

FUNCTION loader_length : word
Liefert die Länge des relokatiblen Laderteils in Byte. Der Programmlader kann auf beliebige RAM-Adressen beliebig oft verschoben werden. Alle hier genannten Routinen außer memory_ok können weiter benutzt werden.

FUNCTION eprom_length : word
Liefert die Länge des EPROMs in Byte. Das EPROM liegt ab Adresse 0. Die Länge des EPROMs sollte nur auf diese Art ermittelt werden!

PROCEDURE relocate_to(adress : word)
Verschiebt den relokatiblen Lader auf die genannte Adresse und läßt den Lader dort weiterarbeiten.

PROCEDURE eprom_off
Blendet das EPROM aus. Der Lader muß vorher aus dem EPROM kopiert werden.

PROCEDURE eprom_again
Wenn der Lader nicht im mit dem EPROM adreßgleichen Speicher liegt, so kann mit diesem Befehl die Ausführung im EPROM fortgesetzt werden.

PROCEDURE jump_to(adress : word)
Direktes Anspringen eines Benutzerprogrammes. Das u.U. eingeschaltete EPROM wird dabei nicht ausgeblendet.

PROCEDURE jump_indirect(four_byte_adress : word; adress : word)
In Fällen, die mit dem Aufruf von jump_to nicht zu bewältigen sind, kann meist mit diesem Aufruf gearbeitet werden. Es werden auf der Adresse four_byte_adress, die zum Zeitpunkt des Aufrufes eingeblendet sein muß, 4 Byte benötigt, um ein Abschalten des EPROMs und einen anschließenden Sprung auf die Adresse adress zu verwirklichen. Mit diesem Verfahren kann z.B. der gesamte Speicher von 0000H-0FFFBH geladen werden und trotzdem noch jede Adresse angesprungen werden.

PROCEDURE load(VAR location : word; VAR contents : sector)
Laden eines CP/M-Sektors von 128 Byte. Der SLAVBOOT erlaubt zwar auch andere Größen, jedoch sind Dateiendebedingungen bei dieser Sektorgröße am leichtesten zu ermitteln.

FUNCTION memory_ok : boolean
Prüfen des Slave-Rechner-Speichers oberhalb der Adressen des EPROMs. Der Speicher wird mit Zufallsbits geladen und wieder ausgelesen. Diese Funktion darf nur während des Laufs im EPROM gerufen werden.

Das Programm compiliert unter PASCAL MT+ von Digital Research.

```
(*$S-*)
(*$E-*) MODULE master_slave_interface;
(*/*)
(*/ EXAMPLE INTERFACE TO SLAVE-CPU-BOOT 1.X FOR PASCAL MT+ *)
(*/*)

  CONST
    slave_base_port = $40 (* 4 slave cpu ports *) ;
(*/ offsets to base port: *)
            data = 0;
          status = 1;
         restart = 2;
             nmi = 3 (* Non maskable interrupt *) ;
(*/ some bit numbers: *)
     msg_in_flag = 5 (* inside interrupt status register 0 *) ;
    msg_out_flag = 5 (* inside control register 1 *) ;
  TYPE
    fio_register = (ctrl0, ctrl1, intsta0, intsta1, intsta2, intsta3, intvec,
                    byte_count, count_comparison, ctrl2, ctrl3, msg_out, msg_in
                    , pattern, mask_pattern, data_buffer);
         sector = ARRAY [ 0 .. 127 ] OF byte;
  VAR
              b : byte;

  PROCEDURE abort(msg : STRING );
    BEGIN (*'abort'*)
      WRITELN ;
      WRITELN ('ABBRUCH: ', msg);
      INLINE ('JMP/0/0)
    END (*'abort'*) ;
(*/ ------------------------------ *)
(*/ FIO Register IO *)

  PROCEDURE set_reg(reg : fio_register; value : byte);
    BEGIN (*'set_reg'*)
      out [ (slave_base_port+status) ] := ord(reg);
      out [ (slave_base_port+status) ] := value
    END (*'set_reg'*) ;

  FUNCTION register(reg : fio_register) : byte;
    BEGIN (*'register'*)
      out [ (slave_base_port+status) ] := ord(reg);
      register := inp [ (slave_base_port+status) ]
    END (*'register'*) ;
(*/ ------------------------------------ *)
(*/ FIO Message Register Communication *)

  PROCEDURE send(value : byte);
    VAR
                i : integer;
```

```
    BEGIN (*"send"*)
      i := 0;
      REPEAT
        i := succ(i);
        IF i = maxint
          THEN abort('Slave-CPU verweigert Empfang');
      UNTIL NOT tstbit(register(ctrl1), msg_out_flag);
      set_reg(msg_out, value)
    END (*"send"*) ;

  FUNCTION receive : byte;
    VAR
              i : integer;

    BEGIN (*"receive"*)
      i := 0;
      REPEAT
        i := i+1;
        IF i = maxint
          THEN abort('Slave-CPU antwortet nicht');
      UNTIL tstbit(register(int_sta_0), msg_in_flag);
      receive := register(msg_in)
    END (*"receive"*) ;
(*/ ----------------------------------*)
(*/ Slave Command Protocoll - common part *)

  PROCEDURE command(cmd : byte);
    BEGIN (*"command"*)
      IF receive <> 0
        THEN abort('Bootstraplader der Slave-CPU meldet sich falsch');
      send(cmd);
      IF receive <> 1
        THEN abort('Bootstraplader der Slave-CPU bestaetigt Empfang falsch')
    END (*"command"*) ;
(*$E+*)
(*/ ---------------------------------- *)
(*/ PUBLIC Procedures *)
(*/ ---------------------------------- *)
(*/ Initialize FIO and Slave using RESET and NMI *)

  PROCEDURE master_init;
    BEGIN (*"master_init"*)
      b := inp [ (slave_base_port+restart) ] ;
      out [ (slave_base_port+status) ] := 1 (* reset fio *) ;
      set_reg(ctrl0, 0) (* clear reset bit *) ;
      set_reg(ctrl2, 0) (* disable slave side *) ;
      set_reg(ctrl3, 0) (* master controls everything *) ;
      set_reg(ctrl3, 0) (* clear buffer *) ;
      set_reg(ctrl3, $40) (* clear buffer bit reset (inverted) *) ;
      set_reg(ctrl0, $14) (* disable interrupts, slave non-z-bus *) ;
      set_reg(ctrl1, 1) (* enable wait on buffer limit *) ;
      set_reg(int_sta_0, $20) (* clear message interrupt flag *) ;
      set_reg(int_sta_0, $E0) (* disable message interrupts *) ;
      set_reg(ctrl2, 1) (* enable slave side *) ;
      b := inp [ (slave_base_port+nmi) ]
    END (*"master_init"*) ;
```

```
(*/ ------------------------------ *)
(*/ SLAVE BOOT COMMANDS *)

  PROCEDURE no_command;
    BEGIN (*"no_command"*)
      command(0)
    END (*"no_command"*) ;

  PROCEDURE get_revision( VAR part1, part2 : byte);
    BEGIN (*"get_revision"*)
      command(1);
      part1 := receive;
      part2 := receive
    END (*"get_revision"*) ;

  FUNCTION loader_length : word;
    BEGIN (*"loader_length"*)
      command(2);
      b := receive;
      loader_length := wrd(receive*256+b)
    END (*"loader_length"*) ;

  FUNCTION eprom_length : word;
    BEGIN (*"eprom_length"*)
      command(3);
      b := receive;
      eprom_length := wrd(receive*256+b)
    END (*"eprom_length"*) ;

  PROCEDURE relocate_to(adress : word);
    BEGIN (*"relocate_to"*)
      command(4);
      send(lo(adress));
      send(hi(adress))
    END (*"relocate_to"*) ;

  PROCEDURE eprom_off;
    BEGIN (*"eprom_off"*)
      command(5)
    END (*"eprom_off"*) ;

  PROCEDURE eprom_again;
    BEGIN (*"eprom_again"*)
      command(6)
    END (*"eprom_again"*) ;

  PROCEDURE jump_to(adress : word);
    BEGIN (*"jump_to"*)
      command(7);
      send(lo(adress));
      send(hi(adress))
    END (*"jump_to"*) ;
```

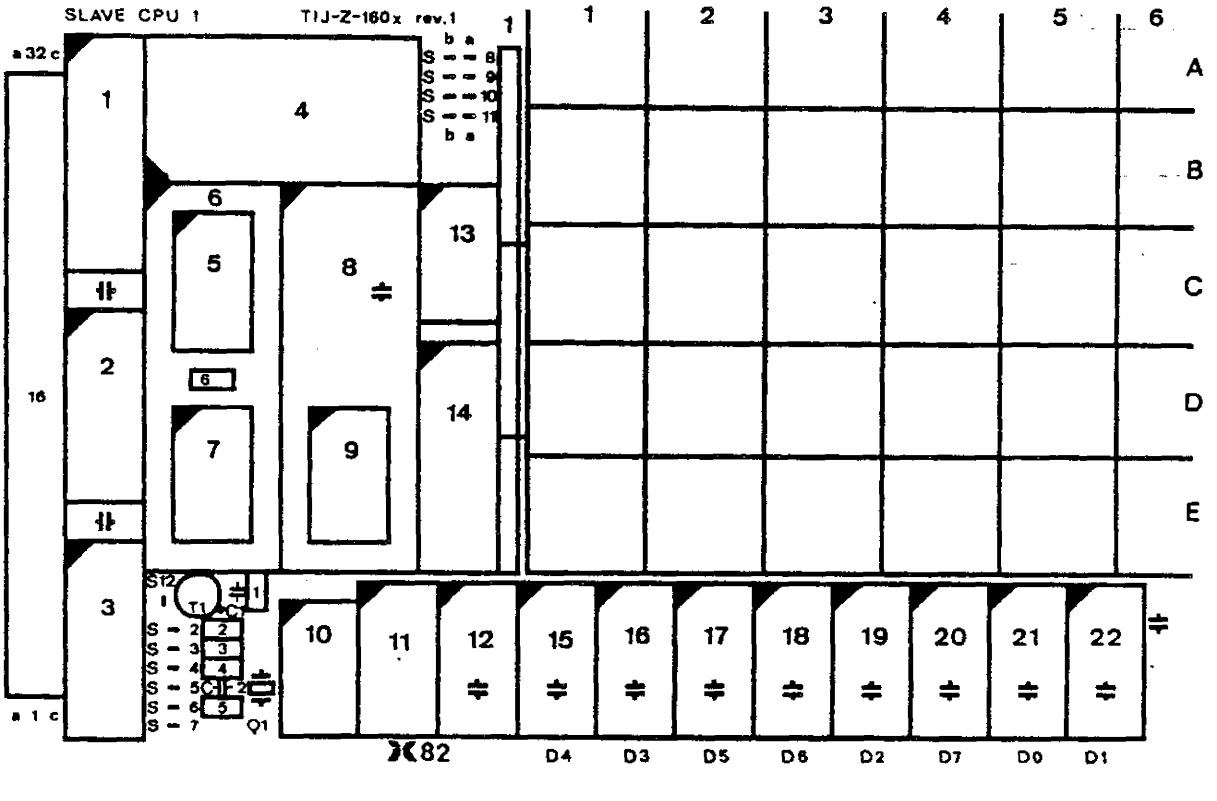
8. Stückliste:

IC 1	PAL16L6 (1608)	R 1	62K
IC 2	74LS245	R 2, 3	2K2
IC 3	74LS682	R 4, 5	330R
IC 4	BYTEWIDE Sockel	R 6	4K7
IC 5	74LS04		
IC 6	Z8038-FIO	C 1	33pF 2,5mm
IC 7 - 9	74LS74	C 2	100nF 2,5mm
IC 8	Z80B-CPU	C 12 Stück	100nF 2,5mm
IC 10	**74S04**		
IC 11 - 12	74LS157	Q 1	12MHz Grundwellenquarz
IC 13	74LS27		
IC 14	PAL16L6 (1609)		
IC 15 - 22	4164 dyn. RAM (128 R.C. Pin 1 NC.)	T 1	2N2906 o.ä.

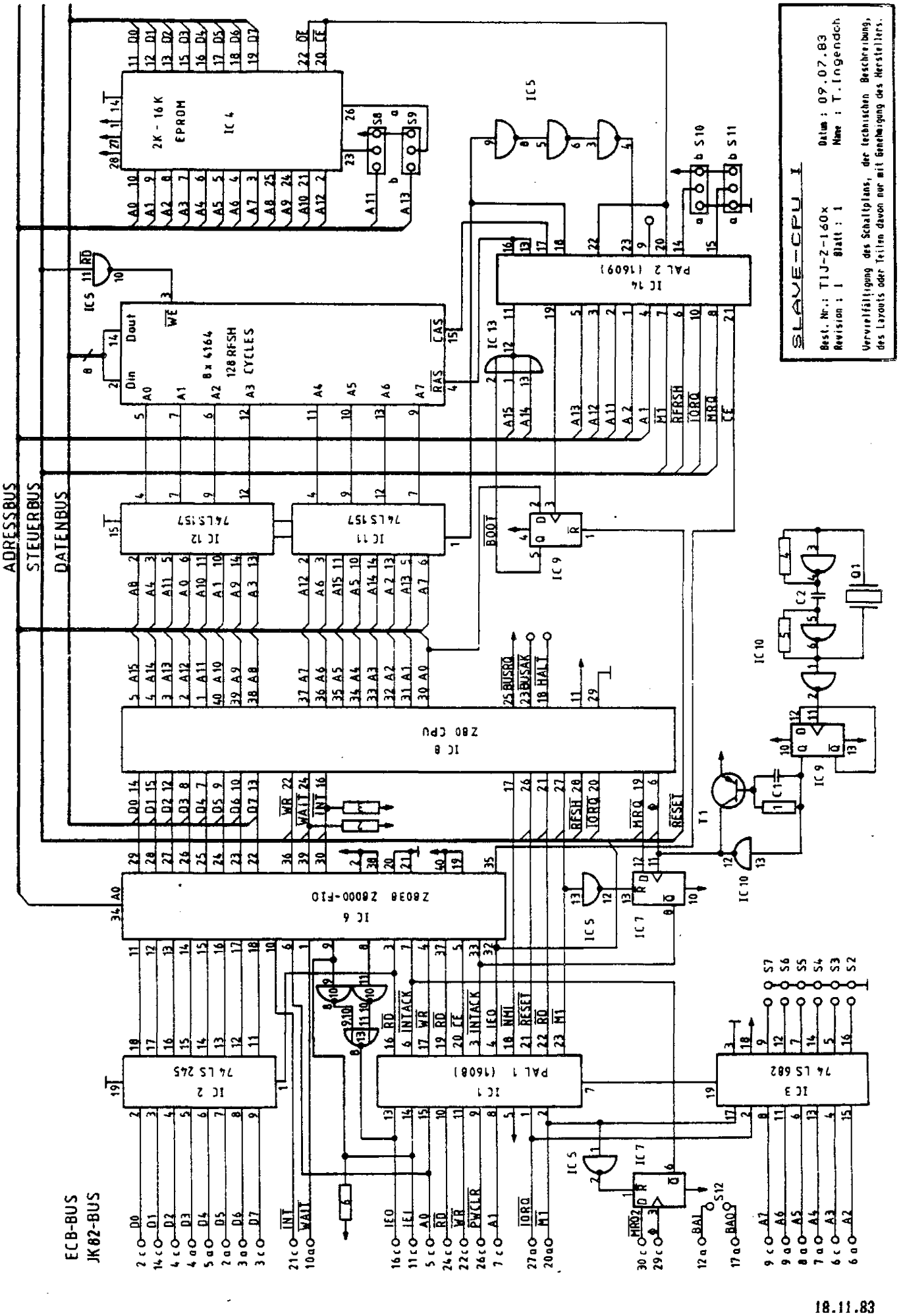
Sockel 14polig	5 Stück (300mil.)
Sockel 16polig	10 Stück (300mil.)
Sockel 20polig	2 Stück (300mil.)
Sockel 24polig	2 Stück (300mil.)
Sockel 28polig	1 Stück (600mil.)
Sockel 40polig	2 Stück (600mil.)

Pfostenreihe 52pol.
Steckjumper 12 Stück
VG64 Messerleiste

9. Bestückungsdruck:



10. Schaltplan:



jk82 SLAVE-CPU's

Softwareänderung:

Die SLAVE CPUs werden mit dem Eprom SLAVE-BOOT 1.1 statt 1.0 ab dem 28.03.85 ausgeliefert. Der BOOT ist wie folgt geändert:

1) In der Routine PUTFIO sind vor "ENDM" 35 NOP-Befehle eingefügt.
2) In der Routine GETFIO sind vor "COND '&REG' NE 'A'" 35 NOP-Befehle eingefügt.
3) In der Routine WAIT sind vor "COND VALU EQ 0" 35 NOP-Befehle eingefügt.

Änderungsgrund:

Das FIO wird grundsätzlich in zwei Schritten adressiert. Zuerst wird in den Kontroll-Port die Adresse des Registers geschrieben. Das FIO befindet sich dann im Status 1. Im Status 1 werden grundsätzlich keine IP-Bits (Interupt Pending) gesetzt, da ein Interrupt an dieser Stelle fatale Folgen haben kann. Die Interrupt Service Routine beschreibt üblicherweise selbst einige Kontrollregister, was dann dazu führt, daß die Adresse des Kontrollregisters aus der Int. Service Routine als zu schreibender Inhalt des bereits adressierten Registers interpretiert werden kann.

Grundsätzlich sollte das FIO daher nur so lang wie nötig im Status 1 verbleiben. Ein Assembler-Programm wird z.B. nach dem OUT-Befehl mit der Registeradresse sofort den IN-Befehl zwecks Auswertung des Inhalts dieses Registers (z.B. INT. Status-Register) ausführen. Der IN-Befehl schaltet das FIO sofort wieder in den Status 0. Weitere IN-Befehle (Polling) beeinflussen den Zustand nicht. Ein bestimmtes IP-Bit wird also höchsten um die Zeit zwischen dem OUT- und dem IN-Befehl verzögert.

Ist die Zeit im Status 1 nun sehr groß, was insbesonders bei dem in PASCAL geschriebenen Programm SDOWNLOD der Fall ist, kann folgendes geschehen:
Der Slave beschreibt sein Message-Out-Register während der Master seine Seite des FIOs gerade in den Status 1 gebracht hat. Das Message-Interrupt-Pending-Bit auf der Masterseite kann daher noch nicht gesetzt werden. Das Message-Register-Full-Bit auf der Slaveseite ist aber dasselbe Bit. Es wird demnach ebenfalls noch nicht gesetzt.
Der Slave seinerseits fragt dieses Bit kurz nach dem Beschreiben des Message-Out-Registers ab um festzustellen, ob der Master die Message empfangen hat. Da dieses Bit aber noch nicht gesetzt ist, nimmt die Slave-Software fälschlicherweise an, daß die Message abgeholt wurde. Dies führt u.U. zum Aufhängen des Systems.

Der Einfachheit halber wurden in die entsprechenden Routinen des SLAVE-BOOT 35 NOP-Befehle eingefügt, die eine ausreichende Wartezeit garantieren.

Bei der Erstellung eigener Software ist dieses Problem am einfachsten dadurch zu umgehen, daß die Unterprogramme in Assembler geschrieben werden und das FIO nicht länger als nötig im Status 1 belassen wird.

Stand vom: 28.03.85