MP16

68331\2 Controller Module

1. FUNKTION

1.1. Datenblatt

1.1.1. Anwendung

Universelles Controller-Module für Steuerungen.

1.1.2. Daten

Parameter	Wert	Dimension
Controller	68332 oder 68331	
RAM	64	KByte
EPROM	128	KByte

1.1.3. Besonderheiten

Durch einen residenten Monitor kann mit dem Modul unmittelbar über eine serielle Schnittstelle kommuniziert werden und neue Programme geladen und gestartet werden.

1.1.4. Aufbau

Scheckkartengröße 53 * 83 mm mit seitlichen Busstiften.

1.1.5. Stromversorgung

Spannung	Strom	Leistung
+5V	ca. 300mA	1,5W
-5.2V	-	
+12V	-	
-12V	-	
Gesamt		1,5W

1.2. Blockdiagramm

1.3. Beschreibung

2. BETRIEB

2.1. Konfigurierung

2.1.1. Jumper

Betriebsart	Jumper	Bemerkung
Internes ROM benutzen	J1: 2-3	z.B. 8052AH
Externes ROM benutzen	J1: 1-2	z.B. 8032
Onboard RS232 Converter benutzen	J2: close	
Betrieb mit 2764/27128	J3: 1-2 J4: 2-3	onboard programming!
Betrieb mit 27256	J3: 1-2 J4: 1-2	nur lesen
Betrieb mit 27512	J3:2-3 J4:1-2	nur lesen

2.2. Bedienung

2.2.1. BASIC

Zum Start des BASIC-Interpreters auf dem Terminal mehrmals die SPACE-Taste drücken, bis der Controller mit der Systemmeldung:

```
*MCS-51(tm) BASIC V1.1*
READY
>
```

antwortet.

Alle weiteren Befehle sind in dem BASIC-MANUAL beschrieben!

2.3. Programmierung

2.3.1. EPROM

ACHTUNG: Bei der EPROM-Programmierung muß die Busleitung **HLT** auf LOW gelegt werden! Beim normalen Betrieb sollte diese HIGH sein, da damit das Port1 frei zur Verfügung steht (siehe auch PAL-Listing).

An den Pin **INT** muß die Programmierspannung (z.B. 12,5V) + ca. 1V gelegt werden!

2.3.2. Speicherbelegung (nicht Neumann-Mode!)

Start	PSEN	RD	WR	
$F000				M0
$E000				M1
$D000				M2
$C000				M3
$B000				M4
$A000		…		M5
$9000		EPROM		M6
$8000		EPROM		M7
$2000		RAM	RAM	
$0000	BASIC ROM			

ACHTUNG: falls die vorselektierten Moduladressen (M0..M7) mit RD benutzt werden, muß auf diesen Adressen gegebenenfalls das EPROM ausgeblendet werden. Dies kann durch Beschaltung mit jeweils einer DIODE (Schottky) von Mn nach MS erreicht werden (siehe auch PAL-Listing!)

2.3.3. Maschinenroutinen

Beim Einbinden von Maschinenprogrammen muß der Speicher, der das Programm enthält im PSEN- Adressraum liegen. Dies ist durch High an C0 erreicht (Neumann Mode, siehe PAL-Listing!).

Start	PSEN	RD	WR	
…$FFFF	…	…	…	…
$9000	EPROM	EPROM		M6
$8000	EPROM	EPROM		M7
$2000	RAM	RAM	RAM	
$0000	BASIC ROM			

3. KOMMUNIKATION

3.1. Allgemein:

Die Kommunikation mit dem MP16 erfolgt **seriell**. Jede Kommunikation zwischen einem HOST und dem MP16 ist über **Messages** (Datenpakete) definiert. Jede Message beginnt mit einem **MessageByte** und kann durch weitere **Parameter** (Daten) ergänzt werden.

Falls die Messages fehlerhaft sind sendet der Empfänger eine spezielle Message (**Interrupt**) als Fehlermeldung.

3.1.1. Serielle Parameter:

- Geschwindigkeit: 9600 Baud
- Wortlänge: 8 Bit
- Parity: None
- Frame: 1 Startbit, 1 Stopbit

3.2. Message:

Jede Kommunikation zwischen HOST und TROLLEY erfolgt über festgelegte Datenpakete (Messages), die immer durch ein spezielles Commandowort (MessageByte) eingeleitet werden.

Message=MessageByte [& Daten]

MessageByte:

Das MessageByte gibt an, welche **Bedeutung** die Message hat (in jeder Gruppe 32 Möglichkeiten!) und ob und mit welcher **Länge** Daten folgen. Dabei gibt es folgende Definitionen:

1) **MSG0**: Code = 0..31 {000X XXXX}

Jede Message besteht nur aus **MSG0** (**keine** weiteren Daten!).

```
GoMsg = Msg0 + 0;                  {Continue after Stop}
SystemMsg = Msg0 + 1;              {get System Ptr}
IdleMsg = Msg0 + 2;                {Escape tasks}
StartMsg = Msg0 + 3;               {start subroutine at ADDRESS}
RestartMsg = Msg0 + 4;             {Restart complete}
Peek1Msg = Msg0 + 5;               {peek byte from ADDRESS}
Peek2Msg = Msg0 + 6;               {peek word from ADDRESS}
Peek3Msg = Msg0 + 7;               {peek triple from ADDRESS}
Peek4Msg = Msg0 + 8;               {peek LongInt from ADDRESS}
ResetMsg = Msg0 + 9;               {generate RESET}
BufferMsg = Msg0 + 10;             {request number of  bytes free in buffer}
IDMsg = Msg0 + 11;                 {get system ID}
ModuleMsg = Msg0 + 12;             {get module number}
DBugMsg = Msg0 + 13;               {enter DBUG}
TaskMsg = Msg0 + 14;               {get currentTask}
ContinueMsg = Msg0 + 17;           {BufferHandshake}
WaitMsg = Msg0 + 19;               {BufferHandshake}
```

2) **MSG1**: Code = 32..63 {001X XXXX, 1byte param}

Jede Message besteht aus **MSG1** und **einem** weiteren Byte!

```
IntMsg = Msg1 + 0;                 {interrupt errorCode}
Addr1Msg = Msg1 + 1;               {add 1 byte to ADDRESS}
Poke1Msg = Msg1 + 2;               {poke 1 byte to ADDRESS}
```

```
Data1Msg = Msg1 + 3;                    {data 1 byte}
StopMsg = Msg1 + 4;                     {Stop according mode}
```

3) **MSG2**: Code = 64..95 {010X XXXX, 2bytes param}

Jede Message besteht aus **MSG2** und **zwei** weiteren Bytes!

```
Addr2Msg = Msg2 + 1;                    {add 2 bytes to ADDRESS}
Poke2Msg = Msg2 + 2;                    {poke word to ADDRESS}
Data2Msg = Msg2 + 3;                    {data word}
PeekMsg = Msg2 + 4;                     {peek size bytes from ADDRESS}
GetStatusMsg = Msg2 + 5;                {get size bytes from STATUS}
```

4) **MSG3**: Code = 96..127 {011X XXXX, 3*bytes}

Jede Message besteht aus dem **MSG3** und **drei** weiteren Bytes!

```
Addr3Msg = Msg3 + 1;                    {add triple to ADDRESSS}
Poke3Msg = Msg3 + 2;                    {poke triple to ADDRESS}
Data3Msg = Msg3 + 3;                    {data triple}
```

5) **MSG4**: Code = 128..159 {100X XXXX, 4*bytes}

Jede Message besteht aus **MSG4** und **vier** weiteren Bytes!

```
PortMsg = Msg4 + 0;                     {set up Port}
AddrMsg = Msg4 + 1;                     {set ADDRESSS for Peek/Poke/JSR}
Poke4Msg = Msg4 + 2;                    {poke long to ADDRESS}
Data4Msg = Msg4 + 3;                    {data long}
```

6) **MSGS1**: Code = 160..191 {101X XXXX, size, size*bytes}

Jede Message besteht aus **MSGS1**, **einem** Byte für die Anzahl der folgenden Bytes und den entsprechenden **weiteren** (0..255) Datenbytes!

7) **MSGS2**: Code = 192..223 {110X XXXX, sizeH, sizeL, size*bytes}

Jede Message besteht aus **MSGS2**, **zwei** Bytes für die Anzahl der folgenden Bytes und den entsprechenden **weiteren** (0..65535) Datenbytes!

```
PokeMsg = MsgS2 + 2;                    {poke size bytes to ADDRESSS}
DataMsg = MsgS2 + 3;                    {data size bytes}
SetStatusMsg = MsgS2 + 5;               {set size bytes to Status}
```

8) **MSGS4**: Code = 224..255 {111X XXXX, size1, size2, size3, size4, size*byte}

Jede Message besteht aus **MSGS4**, **vier** Bytes für die Anzahl der folgenden Bytes und den entsprechenden **weiteren** (0..$FFFFFFFF) Datenbytes!

Interface Routinen:

Zur Vereinfachung der Beschreibung werden folgende Softwareroutinen (PASCAL) vorgeschlagen, die die Kommunikation realisieren. Diese Routinen senden bzw. empfangen Daten oder Parameter entsprechend dem Protokoll.

procedure LoadAddress (address:LongInt);

setzt im Empfänger eine Adresse z.B. für DownLoad

Protokoll: **AddrMsg** & address3, address2, address1, address0

z.B. setze auf $8000: sendet **$81,** $00, $00, $80, 00

procedure DownLoad (size:Integer; code:Ptr);

sendet daten ab **code** zum Empfänger

Protokoll: **PokeMsg** & sizeH & sizeL & code0 & code1 & …

z.B. lade code (code: $FF, …) mit 3100 bytes: sendet **$C2**, $0C, $1C, $FF, ...

procedure SendByte (data:Byte);

sendet daten (1 byte) zum Empfänger

Protokoll: **Data1Msg** & data

z.B. sende Zahl 10: sendet **$23**, $0A

procedure SendWord (data:Integer);

sendet daten (2 bytes) zum Empfänger

Protokoll: **Data2Msg** & dataH & dataL

z.B. sende Zahl 10: sendet **$43**, $00, $0A

procedure SendLong (data:LongInt);

sendet daten (4 bytes) zum Empfänger

Protokoll: **Data4Msg** & data3 & data2 & data1 & data0

z.B. sende Zahl 10: sendet **$83,** $00, $00, $00, $0A

function RecvByte :Byte;

empfängt daten (1 byte) vom Sender

Protokoll: **Data1Msg** & data

z.B. **$23**, $0A im Buffer: empfängt Zahl 10!

function RecvWord :Integer;

empfängt daten (2 bytes) vom Sender

Protokoll: **Data2Msg** & dataH & dataL

z.B. **$43**, $00, $0A im Buffer: empfängt Zahl 10!

function RecvLong :LongInt;

empfängt daten (4 bytes) vom Sender

Protokoll: **Data4Msg** & data3 & data2 & data1 & data0

z.B. **$83,** $00, $00, $00, $0A im Buffer: empfängt Zahl 10!

Trolley:

Folgende Befehle werden vom Trolley bearbeitet:

Download:

Das Laden des eigentlichen Trolley-Programs erfolgt z.B. mit folgender Sequenz:

- LoadAddress($8000);
- DownLoad(3100, TrolleyCode^);

Reset:

Alle Aktivitäten im Trolley werden unterbrochen und alle Parameter in Grundeinstellung (default) gebracht:

- SendByte(1)

SetParameters:

Parameter für die Zeitsteuerung werden geladen.

- SendByte(2)
- SendWord(DelayTransmit)
- SendWord(DurationTransmit)
- SendWord(DelayPreamp)
- SendWord(DelaySM)

Sleep:

Trolley wird in lowpower Modus gebracht.

- SendByte(3)

GetNMR25:

Trolley sendet NMR Frequenzen (Time & Periods) von allen 25 Proben.

- SendByte(4)
- for channel=1 to 25 do

```
begin
        Time(channel) := RecvLong;
        Periods(channel) := RecvLong;
end;
```

GetNMR:

Trolley sendet NMR Frequenz (Time & Periods) von bestimmter Probe.

- SendByte(5)
- SendByte(channel)
- Time(channel) := RecvLong;
- Periods(channel) := RecvLong;

Position:

Trolley sendet Positionscode

- SendByte(6)
- Position:= RecvLong;

Threshold Envelope:

setzt Schwelle für Envelope

- SendByte(7)
- SendByte(threshold)

Temperatur:

Trolley sendet Temperaturwert

- SendByte(8)
- SendLong(numberOfCycles)
- Temperature := RecvLong;

Gate:

Trolley generiert Gate

- SendByte(9)
- SendWord(timeInUS)

3. FERTIGUNG

3.1. Mechanik

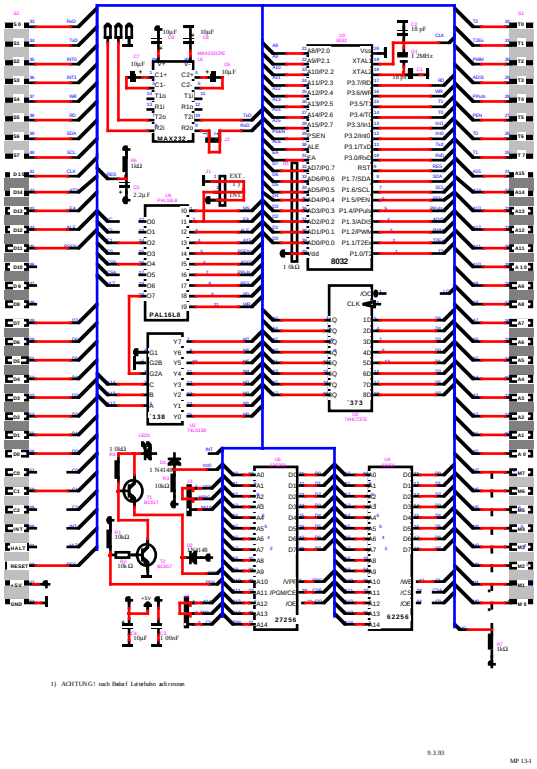
3.1.1. Stecker (S1)

Carrier-Stifte: Zu empfehlen bei Einbau in Geräten ohne Sandwich-Aufbau.

Wire-Wrap Buchsen/Stifte (Fischer: BL9): Sandwich-Aufbau

3.2. Elektronik

3.2.1. Schaltbild



3.2.2. Bestückungsplan

3.2.3. Stücklisten

3.2.4. PAL-Listing

```
    module m8052
    title 'vwa190493';
    m8052Dec device 'P16L8';
    "Inputs
MS,EA,ALE,A15,PSEN,ADIS,PPULS,RES,RD,WR,HLT,C2
PIN  1,2,3,4,5,6,7,8,9,11,18,13;
    "Outputs
LC,C1,C0,CSB,CSA,ME
PIN  12,14,15,16,17,19;
    "Modes
ProgMode = !HLT;
Prog64 = ProgMode & C2;
Prog256 = ProgMode & !C2;
NeumannMode = C2;
    EQUATIONS
!LC = !ALE # (ProgMode&!ADIS);
!C1 = !WR # (Prog64&!PPULS);
!C0 = !RD # (NeumannMode & !PSEN);
!CSB = (A15 & MS & !Prog256) # (NeumannMode & A15 & !MS & !Prog256)
     # (Prog256 & A15 & !PPULS & !ADIS) # (Prog256 & A15 & ADIS);
!CSA = !A15;
!ME = A15;
    END m8052;
```

4. MODIFIKATION

5.1. Versionen

5.1.1. Vers. 1.0:

5.1.2. Vers. 2.0:

5. ANHANG

5.1. Bausteinunterlagen

5.1.1. Controller 8052

5.1.2. Memory 62256

5.1.3. EPROM 27C256

5.1.4. MAX232