

Laborbericht HWA

Fach Hardwarearchitektur

Tilo Pinkert
Andreas Romeyke
Marcus Schubert
Laborbericht REV. 1.1 β

7. Mai 2002

Zusammenfassung

Dieses Dokument beschreibt die Realisierung einer Distanzanzeige unter Verwendung des Sensors GP2D02 von Sharp, der Anzeigeeinheit HDLx 2416 von HP und eines Protoboards mit Motorola 68302-Prozessor.

Die vorliegende Arbeit wurde eigenständig im Rahmen eines Laborversuches an der FH Leipzig im SS 2002 erstellt. Als Satz-Software wurde das freie \LaTeX -System verwendet.

Inhaltsverzeichnis

1	Aufbau des Systems	3
1.1	Voraussetzung	3
1.2	Initialisierung des Boards	4
1.2.1	Adressplan	4
1.2.2	Festlegung der Chipselect	4
1.2.3	Festlegung der Steuerregister	4
1.2.4	RAM-Test	4
2	interner Parallelport	6
2.1	Initialisierung der internen Parallelports	6
2.2	Funktionsüberprüfung durch Programmbeispiel Lauflicht	6
3	Anwendungsaufgabe: Digitaler Distanzsensor	7
3.1	Aufgabenstellung	7
3.2	Hardwareanbindung	7
3.3	Programmierung	7
4	Anhang	8
4.1	Schaltplan	8
4.2	Quellcode	9

Kapitel 1

Aufbau des Systems

1.1 Voraussetzung

Das MC68302 Versuchsboard war die Grundlage für die spätere Anwendungsentwicklung. Die Mikrorechnerkonfiguration sollte unter Nutzung der folgenden Komponenten erfolgen:

- 8K x 16 Bit EPROM
- 8K x 16 Bit RAM
- 4K x 16 Bit externe Peripherie

Für die Lösung der Aufgabe war die Teilnahme an der Vorlesung Hardwarearchitektur [2] notwendig. Zur Verfügung standen eine Kurzreferenz der Assemblerbefehle [1], sowie eine Kurzbeschreibung [3] der wichtigen Konfigurationsregister des 68302 und die Datenblätter für die Anzeigeeinheit HDLx 241C und den Distanzsensor GP2D02¹.

¹Unter den URL <http://www.sharp.com> (GP2D02) und <http://www.hewlettpackard.com> (HDLx241C) erhältlich. Die Funktionsweise des Sensors und der Anzeigeeinheit ist für das Verständnis dieser Beschreibung notwendig.

1.2 Initialisierung des Boards

1.2.1 Adressplan

Für die Inbetriebnahme des Mikroprozessors ist es essentiell notwendig einige Basisregister zu konfigurieren. Aus dem Aufbau des Boards heraus ergaben sich ff. Bereiche, mit denen der Mikroprozessor interagieren musste:

8K x 16 Bit	EPROM
8K x 16 Bit	externes RAM
4K x 16 Bit	externe Peripherie
4K x 16 Bit	internes RAM

Da der Mikroprozessor über das ChipSelect 0 ab Adresse 0x0000 sein Startprogramm legt, ergab sich dann der Adreßplan nach Tabelle 1.1 (auf Seite 4).

Tabelle 1.1: Adressplan

Bereich		Start	Ende	Adressltg. A23..A0
8K x 16 Bit	EPROM	0x00000	0x03fff	0000 0000 0000 0000 0000 0000
8K x 16 Bit	extern. RAM	0x04000	0x07fff	0000 0000 0100 0000 0000 0000
4K x 16 Bit	Peripherie	0x08000	0x09fff	0000 0000 1000 0000 0000 0000
4K x 16 Bit	intern. RAM	0x10000	0x11fff	0000 0001 0000 0000 0000 0000

1.2.2 Festlegung der Chipselect

Aus dem Adressplan (Tabelle 1.1) ergibt sich die Programmierung der BRx- und ORx-Register nach Tabelle 1.2 (auf Seite 4).

Tabelle 1.2: Belegung BRx- und ORx-Register

x	Bereich	BRx	ORx
0	8K x 16 Bit EPROM	0000 0000 000	1111 1111 110
1	8K x 16 Bit extern. RAM	0000 0000 010	1111 1111 110
2	4K x 16 Bit Peripherie	0000 0000 100	1111 1111 111

Das BAR-Register bezieht sich auf das interne RAM

4K x 16 Bit intern. RAM	0000 0001 0000
-------------------------	----------------

1.2.3 Festlegung der Steuerregister

Das BAR-Register legt nicht nur die Anfangsadresse des internen RAMs fest, sondern auch, ob und wie die Functioncodes FC0-FC2 ausgewertet werden sollen, die die Sichtbarkeit der RAM-Bereiche im User- und Supervisormodus festlegen. Das CFC wurde für den Versuch auf 0 gesetzt, so daß die FC0-FC2 nicht ausgewertet wurden.

Das Statusregister SCR dient zur Einstellung von Interrupt-, Watchdog- und Timerverhalten, es wurde so eingestellt, daß der 68000-Modus benutzt wurde. Da der Bus nicht an einen externen Arbitrer abgegeben wurde und auch kein Watchdog benötigt wurde, wurden die Flags BCLM und HW DEN gelöscht. Für die restlichen Parameter wurden Defaultflags (sh. [3]) gesetzt.

Das CCR wurde in der Defaulteinstellung belassen.

1.2.4 RAM-Test

Das Testen des RAMs verfolgt zwei Zwecke, erstens soll so überprüft werden, ob der Adressplan und die Konfigurationsregister richtig gesetzt wurden, zweitens ob der externe RAM in Ordnung ist.

Die Prüfung auf korrekte Konfiguration ist einfach, da jeder Schreib-Lese-Zugriff auf nicht konfigurierte Adressbereiche einen Busfehler provoziert.

Die Prüfung, ob das externe RAM in Ordnung ist, ist weitaus schwerer. Vereinfacht muß ein Testpattern in jede Speicherzelle geschrieben werden, und dann jede beschriebene Speicherzelle wieder ausgelesen und mit dem Pattern verglichen werden. Die Auswahl der richtigen Pattern ist eine Wissenschaft für sich (sh. [4]), so daß sich hier auf die einfachen Testpattern 0xAA und 0x55 beschränkt wurde, die eine alternierende Bitfolge beinhalten, und so die Aufdeckung einzelner defekter Bit-Zellen ermöglichen.

Auf eine Prüfung des internen RAMs wurde verzichtet, da in diesem RAM-Bereich die wichtigsten Register liegen und der frei belegbare Bereich nicht genutzt wurde.

Kapitel 2

interner Parallelport

2.1 Initialisierung der internen Parallelports

Zuerst wurden alle Leitungen des Parallelports durch die Kontrollregister PCNTA für den A-Port und PCNTB für den B-Port als Portleitung geschaltet. Danach wurden über die Datenrichtungsregister PDDRA und PDDRB den Leitungen die Ein- und Ausgabe zugeordnet. Anschließend wurden die Datenregister PDATA und PDATB gelöscht.

2.2 Funktionsüberprüfung durch Programmbeispiel Lauflicht

Die Aufgabenstellung sah es vor, das am Parallelport A mittels externer Schalter eine Bitmuster angelegt werden sollte und sich diese dann durch die restlichen Port-Pins der Parallelports A und B als Lauflicht bewegen. Dazu wurden die ersten 8 Bit des 16-stelligen Ports A als Eingang belegt. Zur Ausgabe dienen die boardinternen Kontroll-LEDs. Die Aufgabe wurde durch Rotieren eines Datenregisters realisiert, das vorher mit der o.g. Bitkombination geladen wurde.

Im eigentlichen Programm wurde dann ein Ripple-Counter realisiert.

Kapitel 3

Anwendungsaufgabe: Digitaler Distanzsensor

3.1 Aufgabenstellung

Der Distanz-Sensor GP2D02 von SHARP liefert ein serielles Signal als Ergebnis einer optischen Entfernungsmessung zu reflektierenden Objekten. Es sind Untersuchungen über das Verhalten dieses Sensors am Rechnermodell ¹ vorzunehmen. Zur Anzeige dient das alphanumerisches LED-Display HDLx 2416 von Hewlett Packard.

3.2 Hardwareanbindung

Der Distanzsensor wurde unter Zuhilfenahme eines Treiberbausteines über das Protoboard angesteuert. Das LED-Display wurde direkt am Prozessorbus angeschlossen. Details sind aus dem Schaltplan in der Anlage 4.1 auf Seite 8 ersichtlich.

3.3 Programmierung

Die Initialisierung wurde so vorgenommen, wie unter 1.2 auf Seite 4 beschrieben.

Für das Auslesen des Sensors war es erforderlich mindestens zwei verschiedene Timer bereitzustellen, einmal 4µs für die Gewährleistung von stabilen Flanken bei der Taktgenerierung für den Sensor, sowie einmal 70ms für die Initialisierung des Sensors. Für die Prüfung der Funktionsweise wurde im Meßtakt über die restlichen, ungenutzten Portleitungen ein Ripplecounter realisiert.

Das Auslesen des Sensors, wie auch die Ausgabe der entsprechenden Werte auf die LED-Anzeige wurde in einer Endlosschleife aufgerufen. Daher ermöglicht das Programm einen dauerhaften Meßvorgang.

Die Bitdaten des Sensor wurden über eine ausgerollte Schleife ² in ein Datenregister eingelesen. Nach dem Einlesevorgang wurde der Wert des Datenregisters mittels einer Wertetabelle in die von der LED-Anzeige erwarteten ASCII-Werte direkt umgesetzt. Dieser Weg wurde gewählt, da durch die Sensorbedingte Ungenauigkeit³ nur 12 diskrete Werte ermittelt werden konnte. Andere Verfahren, wie zum Beispiel Interpolation der Sensorfunktion oder im RAM abgelegte Tabellen wären dadurch unzuweckmäßig gewesen. Die Wertetabelle wurde empirisch im Versuch ermittelt.

¹M68302-Protoboard

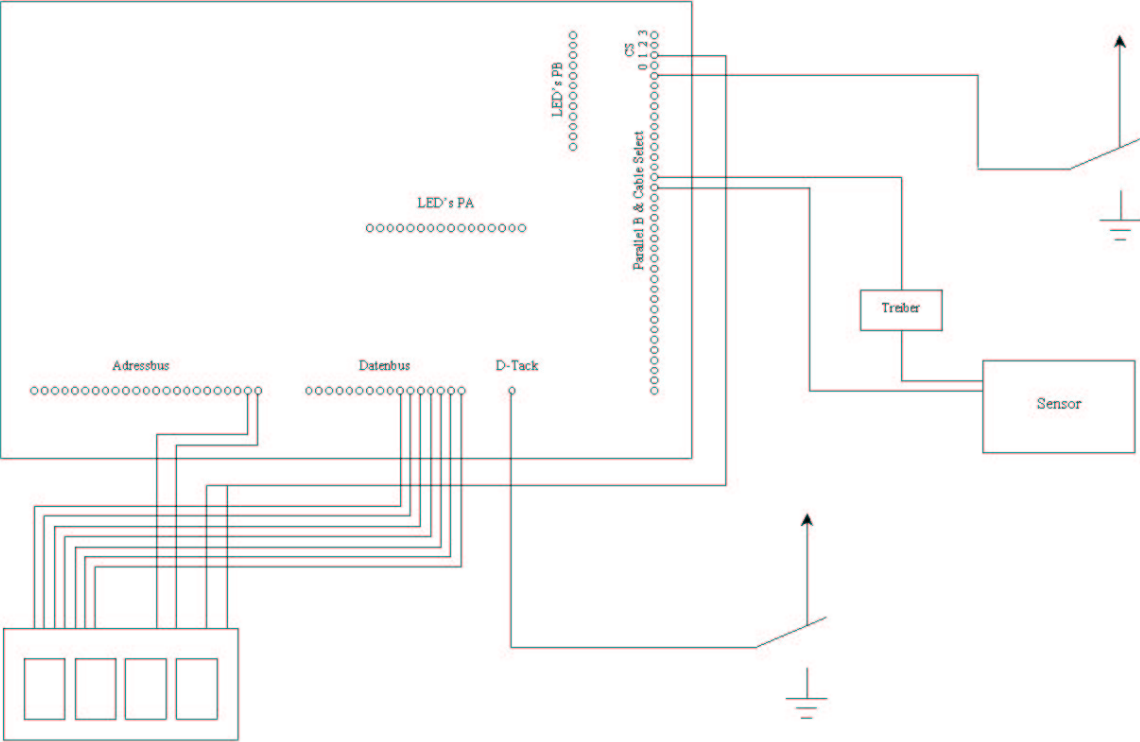
²loop unrolling

³Der GP2D02 ermittelt die Entfernung durch die Reflexion von Licht. Effektiv kann man nur im Bereich von 10-80cm messen. Im Bereich 10 bis 40cm kann man eine Auflösung von 5cm erreichen, sonst nur von 10cm. Der Sensor ist damit eher ein besseres Schätzzeisen

Kapitel 4

Anhang

4.1 Schaltplan



4.2 Quellcode

```
; Distanzsensor
; basierend auf M68302
;
; distance sensor based on a m68302 protoboard
;
; Labor HWA Seminar 003
; Gruppe: 3
; Namen:
;     Tilo Pinkert,
;     Andreas Romeyke,
;     Marcus Schubert
;
; this source is public use- and changeable,
; but comes without any warranty nor support
;
; Adressbereich:
;     EPROM $000000 - $003FFF
;     RAM    $004000 - $007FFF
;     Peripherie $008000 - $009FFF
;     intern. RAM $010000 - $011FFF
10

EPROM_START: EQU $000000
EPROM_END: EQU $003FFF
RAM_START: EQU $004000
RAM_END: EQU $008000
PER_START: EQU $008000
PER_END: EQU $009FFF
IRAM_START: EQU $010000
IRAM_END: EQU $010240
PROGSIZE: EQU $200
30
BAR: EQU $F2
SCR: EQU $F4
OR0: EQU IRAM_START+$832
OR1: EQU IRAM_START+$836
OR2: EQU IRAM_START+$83A
BR0: EQU IRAM_START+$830
BR1: EQU IRAM_START+$834
BR2: EQU IRAM_START+$838
PCNTA: EQU IRAM_START+$81E
PCNTB: EQU IRAM_START+$824
40
PDDRA: EQU IRAM_START+$820
PDDRB: EQU IRAM_START+$826
PDATA: EQU IRAM_START+$822
PDATB: EQU IRAM_START+$828
STACK: EQU RAM_END
VAR_START: EQU $0000
PRG_START: EQU $0400
CLOCKRATE: EQU 8000000
TMR1: EQU IRAM_START+$840
TRR1: EQU IRAM_START+$842
50
TCR1: EQU IRAM_START+$844
TCN1: EQU IRAM_START+$846
TER1: EQU IRAM_START+$849

.symbols ; Erzeugen Symbol-Tabelle
60
.pw 110 ; Druckbreite festlegen
; Vereinbarungen
org VAR_START ;ff. Programm ab Adr. VAR_START
dl STACK ;Stackpointer festlegen (Lage)
dl PRG_START ;Befehlszähler auf Programm setzen
org PRG_START ;ff. Programm ab Adresse $400

;#####
;# set initial registers #
;#####
70

; BAR setzen: 4.Bit von Register auf 1 setzen =A16
; Registerbereich b0 - b11 entspricht A12 - A23
; CFC - Flag = 0 ; Status FC2..FC0 = 101 (Supervisor)
; zu setzender Wert: %1010 0000 0001 0000
```

```

move.w #0101000000010000, BAR ;Lade BAR an Adresse $F2 mit Wert
      ; SCR setzen: %0000 1111 0000 0000 0000 0000 0000, die 1en löschen ev. Statusflags

move.l #0f000000, SCR; lade SCR an Adresse $F4
      ; OR0 setzen: %111 1111111110 1 0 1.Block EPROM Maske
move.w #0111111111111010, OR0
      ; OR1 setzen: %111 1111111110 0 0 2.Block ext. RAM Maske
move.w #0111111111111000, OR1
      ; OR2 setzen: %111 1111111111 0 0 3.Block ext. Peripherie
move.w #0111111111111100, OR2
      ; BR0 setzen: %000 0000000000 0 1 1.Block EPROM
move.w #0000000000000001, BR0
      ; BR1 setzen: %000 0000000010 1 1 2.Block ext. RAM
move.w #000000000001011, BR1
      ; BR2 setzen: %000 0000000100 1 1 3.Block ext. Peripherie
move.w #000000000010011, BR2

;#####
;# testing RAM ext. $4000-$8000 #
;#####
; pattern AAAA
move.l #RAM_START,a0;a0 mit effektiver Adresse laden (Start RAM-Bereich)
move.l #RAM_END,d1;d1 mit eff. (Ende RAM-Bereich)
move.w #0aaaa,d0 ;#AAAA nach d0
RAMTST0 move.w d0,(a0) ;Inhalt d0 in RAM schreiben
cmp.w (a0)+,d0 ;Vergleich, Inhalt RAM korrekt?
bne RAMERR ;nein: Sprung zur Marke
cmpa.l d1,a0
bhs RAME5
bra RAMTST0 ;ja: naechster Durchlauf
RAME5 ; pattern 5555
move.l #RAM_START,a0;a0 mit effektiver Adresse laden (Start RAM-Bereich)
move.l #RAM_END,d1;d1 mit eff. (Ende RAM-Bereich)
move.w #05555,d0 ;#5555 nach d0
RAMTST1 move.w d0,(a0) ;Inhalt d0 in RAM schreiben
cmp.w (a0)+,d0 ;Vergleich, Inhalt RAM korrekt?
bne RAMERR ;nein: Sprung zur Marke
cmpa.l d1,a0
bhs RAMIN
bra RAMTST1 ;ja: naechster Durchlauf

;#####
;# testing RAM int. $10000-$12000 (10240) #
;#####
; pattern AAAA
RAMIN
move.l #IRAM_START,a0;a0 mit effektiver Adresse laden (Start RAM-Bereich)
move.l #IRAM_END,d1;d1 mit eff. (Ende RAM-Bereich)
move.w #0aaaa,d0 ;#AAAA nach d0
RAMTST2 move.w d0,(a0) ;Inhalt d0 in RAM schreiben
cmp.w (a0)+,d0 ;Vergleich, Inhalt RAM korrekt?
bne RAMERR ;nein: Sprung zur Marke
cmpa.l d1,a0
bhs RAMI5
bra RAMTST2 ;ja: naechster Durchlauf
; pattern 5555
RAMI5
move.l #IRAM_START,a0;a0 mit effektiver Adresse laden (Start RAM-Bereich)
move.l #IRAM_END,d1;d1 mit eff. (Ende RAM-Bereich)
move.w #05555,d0 ;#5555 nach d0
RAMTST3 move.w d0,(a0) ;Inhalt d0 in RAM schreiben
cmp.w (a0)+,d0 ;Vergleich, Inhalt RAM korrekt?
bne RAMERR ;nein: Sprung zur Marke
cmpa.l d1,a0
bhs START
bra RAMTST3 ;ja: naechster Durchlauf

START

;#####
;# do other initial things #
;#####

```

```

bsr PARINIT

eor.l d7,d7 ; loesche d7

;#####
;# main loop #
;#####
160

MAINL
move.w d7,PDATA ; Lauflicht, um Aktivitaet anzuzeigen
bsr READ ; Lese Sensor aus
addq.w #1,d7
bra MAINL

;#####
;# endless end loop by logical program-error #
;#####
170

ENDE bra ENDE

;#####
;# endless end loop by ram-error #
;#####

RAMERR move.w #bad2,bad2 ; Gebe "bad2" auf Adressbus aus
bra RAMERR
180

;#####
;# Init PARPORT #
;#####

PARINIT
move.w #0000,PCNTA ; Alle Portleitungen als Port setzen (nicht als
; Peripherie)
190
move.w #0000,PCNTB ; ditto.
move.w #ffff,PDDRA ; 0-15 Ausgabe Port A
move.w #07fe,PDDRB ; 1,2-10 Ausgabe, 0,11 Eingabe Port B
eor.l d0,d0 ; d0 loeschen
move.w d0,PDATA ; Port A loeschen
move.w d0,PDATB ; Port B loeschen
move.w #2,PDATB
move.w #03,d0
rts
200

;#####
;# loop for blinking lights #
;#####

LOOP
cmp.w #0800,PDATB ; Vergleichen wenn Port B eins auf 0 liegt
blo LOOP
ror.w #1,d0 ; rotieren
move.w d0,PDATA
rts
210

;#####
;# time to wait 70 ms #
;#####

WAIT70: ;Timeraufloesung=125ns -> Timer bis 0x88c, Teiler 256 zuehlen lassen
;TMR= %0000 0000 00 0 0 1 01 1
;lesen aus TCR
;TRR ist Referenzregister -> 0x088b
;TER loeschen
220
move.w #%1111111100001011,TMR1
move.b #3,TER1 ; Events loeschen
move.w #088c,TRR1; Referenzregister laden
move.w #0,TCR1 ; Laden des TCR mit 0
move.w #%1111111100001011,TMR1

WAIT70L:
btst #1,TER1 ; Event aufgetreten, sprich TRR und TCR gleich?

```

```

beq WAIT70L
rts

```

```

;#####
;# time to wait 4 us for stable flanks
;#####

```

```

WAIT04:
;Timeraufloesung=125ns -> Timer bis 0x0004, Teiler 0 zaehlen lassen
;TMR= %0000 0000 00 0 0 1 01 1
;lesen aus TCR
;TRR ist Referenzregister -> 0x0004
;TER loeschen
move.w #%00000000000001011,TMR1
move.b #3,TER1 ; Events loeschen
move.w #$0010,TRR1; Referenzregister laden
move.w #0,TCR1 ; Laden des TCR mit 0
move.w #%00000000000001011,TMR1

```

```

WAIT04L:
btst #1,TER1 ; Event aufgetreten, sprich TRR und TCR gleich?
beq WAIT04L
rts

```

```

;#####
;# read sensor and write data to the LED-terminal #
;#####

```

```

; Konvertiere
; Wert cm Char
; c2 -> 5 -> 30
; d9 -> 10 -> 31
; ab -> 15
; 94 -> 20 -> 32
; 84 -> 25
; 7b -> 30 -> 33
; 75 -> 35
; 6f -> 40 -> 34
; 6d -> 45
; 67 -> 50 -> 35
; 65 -> 55
; 62 -> 60 -> 36
; 60 -> 65
; 5f -> 70 -> 37
; 5d -> 75
; 5c -> 80 -> 38
; 59 -> 85

```

```

READ:
; 70 ms LOW auf Port -> Initialisierung Sensor
move.w #$0,PDATB ;
bsr WAIT70
eor.l d0,d0 ; Loeschen
eor.l d1,d1 ; Loeschen

```

```

move.w #$2,PDATB; Takt high
bsr WAIT04
move.w #$0,PDATB; Takt low
bsr WAIT04
move.w PDATB,d0; Lese Bit 7
or.w d0,d1;
add.w d1,d1; Shiften nach links

```

```

move.w #$2,PDATB; Takt high
bsr WAIT04
move.w #$0,PDATB; Takt low
bsr WAIT04
move.w PDATB,d0; Lese Bit 6
or.w d0,d1;
add.w d1,d1; Shiften nach links

```

```

move.w #$2,PDATB; Takt high
bsr WAIT04
move.w #$0,PDATB; Takt low
bsr WAIT04
move.w PDATB,d0; Lese Bit 5
or.w d0,d1;

```

```

add.w d1,d1; Shiften nach links

move.w #$2,PDATB; Takt high
bsr WAIT04
move.w #$0,PDATB; Takt low
bsr WAIT04
move.w PDATB,d0; Lese Bit 4
or.w d0,d1;
add.w d1,d1; Shiften nach links

move.w #$2,PDATB; Takt high
bsr WAIT04
move.w #$0,PDATB; Takt low
bsr WAIT04
move.w PDATB,d0; Lese Bit 3
or.w d0,d1;
add.w d1,d1; Shiften nach links

move.w #$2,PDATB; Takt high
bsr WAIT04
move.w #$0,PDATB; Takt low
bsr WAIT04
move.w PDATB,d0; Lese Bit 2
or.w d0,d1;
add.w d1,d1; Shiften nach links

move.w #$2,PDATB; Takt high
bsr WAIT04
move.w #$0,PDATB; Takt low
bsr WAIT04
move.w PDATB,d0; Lese Bit 1
or.w d0,d1;
add.w d1,d1; Shiften nach links

move.w #$2,PDATB; Takt high
bsr WAIT04
move.w #$0,PDATB; Takt low
bsr WAIT04
move.w PDATB,d0; Lese Bit 0
or.w d0,d1;

; warte 70 ms mit High, da die Angabe im Typenblatt nicht zu
; stabilem zyklischen Auslesen fuehrte.
move.w #$2,PDATB; Takt high
bsr WAIT70
move.w #$0,PDATB; Takt low
; in d1 steht Wert vom Sensor
; konvertiere Wert und gebe ihn auf Dezimalanzeige aus

cmpi.b #$c0,d1;
bmi DEC15;
move.w #$31,$8006;
move.w #$30,$8004;
bra READL;

DEC15: cmpi.b #$9d,d1;
bmi DEC20;
move.w #$31,$8006;
move.w #$35,$8004;
bra READL;

DEC20: cmpi.b #$8c,d1;
bmi DEC25;
move.w #$32,$8006;
move.w #$30,$8004;
bra READL;

DEC25: cmpi.b #$80,d1;
bmi DEC30;
move.w #$32,$8006;
move.w #$35,$8004;
bra READL;

DEC30: cmpi.b #$79,d1;
bmi DEC35;
move.w #$33,$8006;
move.w #$30,$8004;

```

```

bra READL;

DEC35: cmpi.b #$72,d1;
      bmi DEC40
      move.w #$33,$8006;
      move.w #$35,$8004;
      bra READL;
                                           390

DEC40: cmpi.b #$6d,d1;
      bmi DEC45
      move.w #$34,$8006;
      move.w #$30,$8004;
      bra READL;

DEC45: cmpi.b #$69,d1;
      bmi DEC50
      move.w #$34,$8006;
      move.w #$35,$8004;
      bra READL;
                                           400

DEC50: cmpi.b #$67,d1;
      bmi DEC60
      move.w #$35,$8006;
      move.w #$30,$8004;
      bra READL;

DEC60: cmpi.b #$63,d1;
      bmi DEC70
      move.w #$36,$8006;
      move.w #$30,$8004;
      bra READL;
                                           410

DEC70: cmpi.b #$60,d1;
      bmi DEC80
      move.w #$37,$8006;
      move.w #$30,$8004;
      bra READL;
                                           420

DEC80: cmpi.b #$5c,d1;
      bmi DEC90
      move.w #$38,$8006;
      move.w #$30,$8004;
      bra READL;

DEC90: move.w #$3f,$8006;
      move.w #$3f,$8004;
                                           430

READL move.w #$63,$8002
      move.w #$6d,$8000
      rts

```

Tabellenverzeichnis

- 1.1 Adressplan 4
- 1.2 Belegung BRx- und ORx-Register 4

Literaturverzeichnis

- [1] Motorola Inc.: *Motorola M68000 Family Programmer's Reference Manual*. Motorola Inc. 1992
- [2] Dr. du Puits: *Vorlesungsscripte HWA der FHL* . FH Leipzig 2002
- [3] Fuchs, Liess: *M68300 - Mikrokontroller*. Franzis Verlag GmbH München 1994
- [4] Schubert: *Vorlesungsscript Text- und Datenkommunikation*. FH Leipzig 2002