pahlbasic V1.03d 2011

Was ist pahlbasic?

pahlbasic ist eine Programmiersprache für einfache Steuerungsanwendungen. Es ist auch eine sehr einfache. Es besteht aus einer Teilmenge von Basic mit hardwarespezifischen Erweiterungen. Es orientiert sich am MCS Tiny Basic (im Intel 8052AH) sowie am Dartmouth-Basic von 1964. Also „old fashion“ Basic. Der in pahlbasic geschriebene Code wird interpretiert und ist daher nicht für Anwendungen geeignet, die hohe Geschwindigkeit erfordern. Typische Anwendungen sind einfache Steuerungen oder Regelungen zum Beispiel Alarmsysteme, Anzeigeeinheiten für Messwerte, Schaltuhren, Thermostate oder ähnl. Mit pahlbasic ist es möglich, mit wenigen Programmierschritten eine einfache Anwendung zu erzeugen, ohne sich in umfangreiche Programmiersprachen einarbeiten zu müssen.

pahlbasic wurde von einem Praktiker für kleine praktische Hobby-Projekte geschrieben, es fehlt jeder theoretische Überbau.

Die Hardware, die von pahlbasic unterstützt wird, ist ein Einplatinencomputer mit 4 Relais-, 2 PWM- sowie einem Soundausgang. Eingänge sind: 4 Digitaleingänge, 1 Zähl-(oder Impulslängen-) und 1 Frequenzmesseingang sowie 3 analoge Spannungsmesseingänge. Eine typische Mikro-SPS-Konfiguration. pahlbasic unterstützt weiter ein 4-zeiliges alphanumerisches LCD-Display, ein 20-Tasten-Keypad für Eingaben, einen Anschluss für DCF77-Funkuhrantenne sowie einen 1Wire-Temperatursensor. Siehe Hardware.

Seit der Version 1.03 wird auch ein zusätzliches EEPROM mit SPI-Anschluss und 128KB mit einem sehr einfachen Datei-System unterstützt.

Der Programmcode von pahlbasic ist sehr kompakt. Der Programmspeicher der Version für Atmega 644 fasst ca 2 KByte. Das reicht für kleine Projekte (ca 100 Programmzeilen). Eine Zugangskontrolleinheit mit Tastencode ist mit 20 Programmzeilen ruck-zuck erstellt. Für grössere Projekte kann die Version mit dem Atmega 1284 verwendet werden mit 8 Kbyte Programmspeicher. (Angaben für die Atmega1284-Version in Klammern ())

Die Programmierung wird wie bei MCS51 Basic mit einem Terminal / PC mit Terminalprogramm durchgeführt. Alternativ kann auch der frei im Internet verfügbare Editor Basic52 benutzt werden, um die Programme zu schreiben und in den Basic-Computer zu laden.

Es gibt zwei Betriebsmodi: Nach dem Einschalten befindet sich der Basic-Computer im *Direktmodus* = Anweisungseingabemodus. Hier können Anweisungen wie "print time" (wird sofort ausgeführt) oder auch Programmzeilen (beginnen mit einer Zeilennummer) wie "**10** let B = sin(A)" (wird im RAM gespeichert) eingegeben werden. Mit der Anweisung "run" wird das Programm im RAM gestartet und der Basic-Computer wechselt in den *Programm-Modus*. Es gibt Anweisungen, die dürfen nur im Anweisungsmodus und andere nur im Programm-Modus verwendet werden. Viele Anweisungen arbeiten in beiden Modi. Zur Eingabe verfügt pahlbasic über einen rudimentären "Zeileneditor" wie MCS51 Basic. (zusätzlich: Cursor links und rechts, sowie Überschreiben).

Das fertige Programm kann dauerhaft im EEPROM gespeichert werden. Der Basic-Computer wird durch eine Autostartfunktion zur Applikation. Das Programm wird dann nach Einschalten des Basic-Computers sofort gestartet.

Bedienung

Der Basic-Computer wird über ein Nullmodemkabel mit dem PC verbunden. Es wird ein Terminalprogramm wie Hyperterminal benötigt. Einstellungen:

9600 Baud	(ändern)
8N1	(Standarteinstellung)
keine Fluss-Steuerung	(ändern)
Rücktaste sendet Strg + H	(Standarteinstellung)
Emulation ANSIW	(Standarteinstellung)
Terminal VT100	(Standarteinstellung)

Nach dem Einschalten der Betriebsspannung wird die Startmeldung: pahlbasic V1.03 2042 bytes free angezeigt.

Eine Programmzeile beginnt immer mit einer Zahl, der Zeilennummer. gefolgt von den Anweisungen und ihren Parametern. Die Programmzeile wird im RAM gespeichert. Weiter passiert zunächst nichts. Ob eine Programmzeile "verstanden" wurde, lässt sich mit der "list"-Anweisung überprüfen. Zu jeder Anweisung gehört eine Syntax, die genau eingehalten werden muss. Sie ist der folgenden Anweisungsliste zu entnehmen. Fehlerhafte Eingaben führen zu falschen Programmen und zu Fehlermeldungen. Einige Fehler werden einfach ignoriert und durch sinnvolle Annahmen ersetzt (z.B.: 3.45.6 ==> 3.456). Leerzeichen zwischen den einzelnen Komponenten einer Programmzeile können weggelassen werden. Es kònnen mehrere Anweisungen in eine Programmzeile gepackt werden, durch Doppelpunkt getrennt.

Ohne Zeilennummer wird die eingetippte Anweisung (Doppelpunkt hier nicht erlaubt) sofort ausgeführt. Dies ist für das Debugging wichtig, da Variablen angezeigt werden können (print A) oder verändert (let B = 9). Die Programmausführung kann jederzeit durch Eingabe von Strg + c auf dem Terminal unterbrochen und durch Eingabe von ' cont ' fortgesetzt werden.

Basic Anweisungen

Alle Basic-Anweisungen werden klein geschrieben. Leerzeichen zwischen den Elementen erhöhen die Lesbarkeit, sind aber nicht nötig. Variablen sind Grossbuchstaben. Siehe Variablen.
Parameter in **[ ]** sind alternativ oder können weggelassen werden. Eine Basic-Befehlszeile sieht so aus:
Zeilennummer Befehl 1 [Parameter**] : [Befehl 2** Parameter**] : [Befehl 3** Parameter**]** …
Max.80 Zeichen.

ex:
10 for N = 1 to 10 **: print** N **: next**

Wo immer eine Zahl als Argument erwartet wird, kann auch ein komplexer Rechenausdruck stehen. (ausser list, lcd, inc, dec und save) ex.**:** gosub 300 + (10 * N) ist möglich!
Zeilennummern sind von 1 bis 65535 erlaubt.

clear

ex.:
clear

löscht alle Variablen und Programmschleifen. Alle Variablen erhalten den Wert 0. Alle Stringvariablen erhalten den Nullstring (“”). Die Relais werden ausgeschaltet, die PWMs auf Anfangswert gesetzt. Direkt- und Programm-Modus.

close Setzt eine “Umleitung” auf den Datenbereich einer anderen Programm-Datei durch **open** wieder zurück auf das aktuelle Programm. Siehe **open, read, write, restore** und **dptr**. Die Schreib-/Lesezeiger werden auf Anfang gesetzt.

ex.:
close

cls (clear screen)

ex.:
10 **cls**

cls löscht den Terminalbildschirm- oder LCD-Inhalt. (abhängig vom aktuellen Ausgabemedium, das mit **lcd** eingestellt wird) Kann auch von der Befehls-eingabe aufgerufen werden. Das Terminal muss das Zeichen „Formfeed“ verstehen. Siehe auch **lcd**.

cont (continue)

ex.:
cont

setzt die Programmausführung nach Programmabbruch (mit ctrl c) fort. Nur im Direktmodus. Nach einem Fehler ist die Fortsetzung mit **cont** nicht möglich.

debug **debug** schaltet den Debug-Modus ein und aus. Nach einem Reset ist der Debug-Modus standartmässig eingeschaltet, d.h. Programmabbruch (mit **Strg + c**) und Einzelschrit-Modus (mit **Strg + d**) sind erlaubt. Nach der Ausführung von **debug** sind **Strg + c** und **Strg + d** gesperrt. Dies ist evtl. sinnvoll, wenn der Basic-Computer als autonomes Gerät eingesetzt wird und auch Daten über RS232 empfängt - kann aber zu Problemen führen, wenn die Autostartfunktion (mit **save 1**) aktiviert wurde, ein Anhalten des Basic-Computers ist dann nur noch möglich, wenn ein Programmabruch im Programm eingebaut wurde. Das gilt verstärkt, wenn auch noch mit **on err** der Programmabbruch bei Fehlern abgeschaltet wurde. Wird **debug** noch einmal ausgeführt, wird der Debug-Modus wieder eingeschaltet. So ist es möglich, eine Hintertür einzubauen. Siehe auch **end**.

Nach Einschalten des Einzelschritt-Modus hält das Programm an, bis die Leertaste gedrückt wird. Es wird die jeweils nächste Instruktion auf dem Terminal gelistet und mit dem Tastendruck abgearbeitet. Instruktionen, die selbst eine Eingabe vom Terminal erwarten, arbeiten evtl. nicht korrekt. (get)

dec (decrement)
dec (System)-Variable

ex.:
dec D

10 **dec time**
20 **dec stack**

Verringert den Wert der (System-)**Variablen** um **1**. Programm und Direktmodus.

del **del ”Name”**
del Nummer

del löscht die Programm-Datei mit dem Namen **Name** oder der Nummer **Nummer**. Die Programm-Nummer kann mit **dir** ermittelt werden.

ex.:
del ”speedtest”
del 2

dir Directory
Druckt das Verzeichnis des Eeproms auf dem Terminal aus. Die Programme werden mit Namen und laufender Nummer ausgedruckt. pahlbasic hat ein sehr simples „Datei-System“. Es gibt 31 (7) „Dateien“ - jede ist 4 (16) Kbyte gross und besteht aus einem 2 (8) Kbyte Programmbereich und einem 2 (8) Kbyte Datenbereich. Fragmentierung ist unmöglich – leider wird der Platz bei kleineren Programmen verschwendet. Nur Direktmodus.

ex.:
dir

do **do Anweisungen loop [until Ausdruck]**

ex.:
10 **do**
20 inc A
30 print A
40 **loop until A = 10**

oder in einer Zeile:

10 let time = 0 **: do :** inc A **: loop until time = 10 :** print A

Der Anweisungsblock zwischen **do** und **loop** wird wiederholt bis der Ausdruck nach **until** wahr wird. Ohne **until** wird der Block immer wiederholt. Mehrere (5) **do**-**loop**-Schleifen können ineinander verschachtelt werden.

end Programm-Ende
end beendet das Programm und kehrt zum Direktmodus zurück. **end** kann auch als Stopmarke benutzt werden. Eine extra **stop**-Anweisung wurde „eingespart“. Nach **end** kann das Programm mit **cont** fortgesetzt werden, auch wenn das vielleicht nicht immer erwünscht ist.

ex.:
1000 **end**

exit Schleifen-Ausgang
exit dient zum vorzeitigen Verlassen einer Programmschleife. Bitte Schleifen niemals mit goto verlassen. Das Programm wird hinter der **loop-** oder **next-**Anweisung fortgesetzt.

ex.:
10 let A = 0 **: do**
20 if A > 5 then **exit**
30 inc A **: loop**
40 hier geht es weiter nach **exit**

for **for NumVar = Startwert to Endwert [step Schrittweite]**

ex.:
10 **for N = 2 to 24 step 2**
20 Anweisung 1
...
50 Anweisung X
60 **next**
oder:
70 **for A = 2 * pi to pi ^ 2 step pi / 100 :** print sin(A) **: next**

for / next ist eine Zähl-Schleife. Der Anweisungsblock zwischen **for** und **next** wird so oft wiederholt, wie durch Startwert, Endwert und Schrittweite bestimmt wird. Als Zähler dient eine Variable (NumVar), die bei jedem Schleifendurchlauf um den Wert Schrittweite erhöht oder erniedrigt (bei negativer Schrittweite) wird, (mit dem Startwert startet) bis der Endwert erreicht ist. Defaultwert für Schrittweite (wenn Schrittweite nicht angegeben wird) ist: +1. Schleifen dürfen jederzeit mit **gosub** verlassen werden – es muss dann mit **return** wieder in die Schleife zurückgekehrt werden. Aber niemals mit **goto!** Mehrere (5) **for / next**-Schleifen können ineinander verschachtelt werden.

get **get NumVar** Eingabe eines einzelnen Zeichens

ex.:
10 **get A :** print **chr A**
20 if buf > 0 then **get B** else return

holt ein Zeichen von der RS232 Schnittstelle und speichert es als Zahlenwert (ASCII) in der Variablen (A). **get** wartet bis ein Zeichen empfangen wurde. Es kann vorher die Systemvariable **buf** abgefragt werden, um Blockaden zu vermeiden. Oder man fragt mit **on buf** ereignisgesteuert ab. Siehe auch **Systemvariable get**.

goto **goto Zeilen-Nummer** Sprunganweisung

ex.:
goto 100
verzweigt zur Programmzeile 100. Kann auch von der Befehlseingabe aufgerufen werden z.B. um das Programm ohne das Löschen der Variablen (wie mit **run**) oder ab einer bestimmten Stelle zu starten oder fortzusetzen.

gosub **gosub Zeilen-Nummer** Sprung in ein Unterprogramm

ex.:
20 **gosub** 100 hier der Aufruf des Unterprogramms
30 Anweisung zu der zurückgekehrt wird.
...
50 **gosub** 100 wiederholter Aufruf
60 Anweisung zu der zurückgekehrt wird.
...
...
100 **Anweisung 1** Ab hier das Unterprogramm

...
110 Anweisung x
120 **return** Springt zurück zur Stelle, die der Aufrufenden folgt.

Oder:
20 **gosub** 100 **:** Anweisung **: gosub** 100 **:** Anweisung **:** Anweisung
…
...
100 Anweisung1 **:** Anweisung2 **: return**

verzweigt zur Programmzeile (hier 100) ins Unterprogramm und merkt sich die Stelle von der aus verzweigt wurde. Rückkehr mit **return**. Aus einem Unterprogramm können wieder Unterprogramme aufgerufen werden.Schachtelungstiefe max. 5 (20). Im Direktmodus wie goto (ohne Rücksprung).

inc (increment)
inc (System)-Variable

ex.:
inc F
Erhöht den Wert der Variablen (hier F) um **1**. Programm und Direktmodus.

if **if Ausdruck then [Anweisung] [:Anweisung]...[else Anweisung] [:Anw...]**

Die Anweisung für Entscheidungen im Programm: **If** prüft den **Ausdruck** und führt die auf **then** folgenden Anweisungen aus, wenn der Ausdruck wahr (<> 0) ist. Ist die **else**-Anweisung vorhanden, so werden die darauf folgenden Anweisungen bei nicht Erfüllung ausgeführt. Bei Zeichenketten ist nur "**=**", "**<>**" und "**in**" erlaubt. Ausserdem muss der **1. Vergleichsoperand eine Stringvariable** sein. **If** / **then** / **else** *müssen* in *derselben* Zeile stehen.Nur eine **If**-Anweisung pro Zeile. Nur im Programm.

ex.:
10 **if** time > 33 **then** goto 20 **else** goto 30
20 **if** A xor (B and 7) **then** print "OK"
30 **if** $1 **=** "Hallo" **then** let A = 2 **:** return **else** let A = 3 **:** return
30 **if** $0 **<>** $2 **then** gosub 50
40 **if** $5 **=** clock **then else** let $5 = "00:00:00" 'der then-Teil darf fehlen.
50 **if** $1 **in** $2 **then** return

input **input NumVar**
input StringVar

Die Anweisung für Eingaben vom Terminal. (Parameter für die Programmausführung)

ex.:
10 print "Geben Sie Ihr Alter ein: " **: input** A

Eingabe eines Zahlenwerts vom Terminal, muss mit return (Zeichen 13) abgeschlossen sein. Ergebnis in der Variablen A.

ex.:
10 print "Name: " **: input** $2
Eingabe einer Zeichenkette vom Terminal, muss mit return (Zeichen 13) abgeschlossen sein.
10 print "Eingabe Sekunden: " **: input** sec Eingabe einer Systemvariablen.

input bin **input bin (System-)Variable**
Eingabe eines numerischen Werts als (max.16bit) Binärzahl.

ex.:
input bin A

input hex **input hex (System-)Variable**
Eingabe eines numerischen Werts als (max. 4-stellige) Hexadezimalzahl.

ex.:
input hex A

input#
1. **input# NumVar**
2. **input# StringVar**

1) Eingabe eines berechenbaren Ausdrucks und das Rechenergebnis einer (System-)Variablen zuweisen. Der Ausdruck wird nicht geprüft!.

ex.:.
10 print "berechne: " **: input# A :** print " = "**; :** print **A**
Als Eingabe sind Ausdrücke wie 4 * sin(.3) möglich. Siehe auch **val**. Diese Anweisung verwandelt den Basic-Computer in einen Taschenrechner.

2) Eingabe eines berechenbaren Ausdrucks und speichern in einer String- variablen. Der Ausdruck wird nicht geprüft!.

ex.:.
10 print "Eingabe: " **: input# $1 :** print " = "**; :** print **val $1**
Der String wird in Tokens umgewandelt und ändert sich daher sofort nach der Eingabe. Dies kann mit: print $1 überprüft werden. Siehe auch **val**.

inkey **inkey NumVar [, Modus] [, Anzahl Ziffern]**
inkey StringVar [, Modus] [, Anzahl Zeichen]

Die Anweisung für Eingaben vom Keypad. Es erscheint ein Cursor auf dem LCD an der Stelle, die vorher mit **locate** festgelegt wurde. Die Eingabe endet spätestens am Zeilenende des Displays.

Modus: 1 = Eingabe muss nicht mit Return abgeschlossen werden, sondern wird nach einer Eingabepause von ca 500ms automatisch übernommen.

Modus: 2 = Bei jedem Tastendruck wird ein kurzer Beep über den Soundausgang ausgegeben.

Modus: 3 = Kombination aus Modus 1 und 2.

Wird **Modus** weggelassen oder der Wert **0** angegeben, dann wird **inkey ohne Beep** und mit **Return** (am Ende der Eingabe) ausgeführt.

Anzahl Ziffern / Zeichen: max. Länge der Eingabe. Ohne diese Angabe**:** max. 20 Zeichen / Ziffern aber höchstens bis zum Zeilenende des Displays.

Die **inkey**-Anweisung ist für das 4x20-Zeichendisplay ausgelegt. Wird ein Display mit geringerer Spaltenzahl verwendet, sollte der Parameter „Anzahl Zeichen" oder „Anzahl Ziffern" unbedingt angegeben und der Spaltenzahl angepasst sein. Sonst kann es sein, dass die Eingabe auf dem Display „zerrissen" wird.

ex.

10 lcd1 : cls : locate 2,1	'Vorbereitung des LCD.
20 inkey A, 2, 10	**'**Eingabe eines numerischen Werts
30 inkey $2, 3	**'**Eingabe einer Zeichenkette

40 inkey time, , 4 ist dasgleiche wie: **30 inkey time, 0, 4**

lcd Wahl des Ausgabemediums für print
lcd Parameter

ex.:
10 **lcd 0 :** print ”Hallo Welt”
20 **lcd 1 :** print ”Hallo Welt”

Wechselt das Ausgabemedium. **lcd1** schaltet die Zeichenausgabe von RS232 auf das LCD um. **lcd0** schaltet wieder zurück zum Terminal.

lcd 2 schaltet den blinkenden Blockcursor auf dem LCD ein. (nur für **inkey**)

lcd 3 verschiebt den Displayinhalt um 1 Position nach links.

lcd 4 verschiebt den Displayinhalt um 1 Position nach rechts.

lcd 5 schaltet das LCD aus. (lcd1 schaltet es wieder ein.)

let **let NumVar = Ausdruck**
let StringVar1 = Stringvar2, [Startpos], [Anzahl]
let Stringvar (Position) = ”Zeichen”
let Stringvar (Position) = ASCii-Wert

ex.:

10 **let A = 2 * (A + sin (pi / 8))**	rechnet Ausdrücke (Formeln) aus.
let N = 5 > 4	Kann vergleichen
10 **let C = A and (250 or B)**	Logische Verknüpfungen
30 **let $0 = "Hallo Welt"**	Kopiert eine StringKonstante in einen String.
let $1 = $2	Kopiert Strings: $2 nach $1
let $2 = $3, 6	Kopiert $3 nach $2 ab dem 6. Zeichen
let $2 = $3, , 6	Kopiert die ersten 6 Zeichen von $3 nach $2
40 **let $0 = $3, 5, 4**	Kopiert 4 Zeichen von $3 ab dem 5. Zeichen nach $0
50 **let $4 = clock**	Kopiert Uhrzeit nach $4
let rel2 = 1	Schaltet Relais 2 ein
let $4(3) = ”A”	Tauscht ein einzelnes Zeichen aus.
60 **let A = ”A”**	weist der Variablen A den Ascii-Wert des Zeichens “A” zu.

Weist einer String- oder (System-)**Variablen** einen Wert zu. Kann auch direkt (im Befehlsmodus) eingegeben werden.Negative Werte bitte in Klammern**:** **let** A = (-2.33) + (-1.4) Ausdruecke werden strikt von links nach rechts berechnet, Punkt-vor Strichrechnung ist nicht implementiert. Bitte Klammern setzen zur Änderung der Reihenfolge. **let** N = 3 + (4 * 9). Max. 5 (20) Klammerebenen. Siehe auch **Stringfunktionen**.

list **list Zeile - Zeile**

ex.:

list	listet das gesamte Programm

list 10	nur Zeile 10
list -100	bis Zeile 100
list 100 -	ab Zeile 100
list 20 - 100	die Zeilen 20 bis 100

Zeigt die gewünschten Programmzeilen an. **list** alleine listet das ganze Programm. **list 10** nur Zeile 10 . **list 20 - 100** die Zeilen 20 bis 100. Sollte nicht im Programm verwendet werden.

load **load "Name"**
load Programm-Nummer

lädt die Programm-Datei mit dem angegebenen **Namen** oder ihrer **Nummer** (mit **dir** ermitteln) in den Speicher. Von einem Programm aus, kann man Programme nachladen, die werden sofort gestartet. Siehe auch **save**, **del** und **dir**.

ex.:
load "Temperaturlogger"
load 17

locate **locate Zeile, Spalte**

ex.:
10 **locate 1, 5 : print** freq
20 **locate A, B – 2 : print** ad1

Setzt den Cursor auf dem LCD auf Zeile und Spalte und schaltet auf LCD - Betrieb um. Zeile = 1...4, Spalte = 1....20

new (alles neu)

ex.:
new

löscht das im **RAM**-Speicher befindliche Programm und alle Variablen. Nur Direktmodus. (Zum Löschen des **internen Eproms** anschliessend **save 0** ausführen.)

on **on time T-Wert gosub Zeilennummer**
on dig n [, Flanke] gosub Zeilennummer
on err gosub Zeilennummer
on key gosub Zeilennummer

ex.:
10 **on time 300 gosub 100**
20 **on dig1, 1 gosub 200**
30 **on dig4, 0 gosub 1200**
40 **on buf gosub 900**
50 **on key gosub 1000**

T-Wert = 1.... 65535 Sekunden, **n** = 1....4 und **Flanke** = 0 oder 1.

on verzweigt bei Auftreten eines Ereignisses zur angegebenen Programmzeile und merkt sich die Position (im Programm) für einen späteren Rücksprung (return). **on** Instruktionen *sollten* am Programmanfang stehen. Die aufgerufenen Unterprogramme *dürfen nicht* am Programmanfang stehen!

on ist nicht interrupt gesteuert, sondern wird nach jeder Anweisung geprüft. Daher können Anweisungen, die den Programmablauf blockieren (wait, get, input, inkey) das Erkennen der Ereignisse verzögern oder sogar verhindern.

on buf aktiviert ein Ereignis, sobald ein Zeichen im RS232 Empfangspuffer ist. z.B. wenn auf dem Terminal eine Taste gedrückt wurde.

on dig1 - on dig4 wird aktiviert, wenn der entsprechende Eingang **dig1** bis **dig4** den Wert **Flanke** annimmt. Logisch eins = + 5 Volt, logisch null = 0Volt. Defaultwert für **Flanke** ist **1**. (Ruhewert 0 Volt, aktiv bei Wechsel auf 5 Volt)

on err verzweigt bei Auftreten eines Fehlers zur angegebenen Programmzeile. Dort kann der Fehler „behandelt" werden. Der normale Programmabbruch im Fehlerfall wird ausgeschaltet. Ist in der Fehlerbehandlungsroutine selbst ein Fehler, wird das Programm abgebrochen.

on key wird aktiv, wenn die Taste «**esc**» auf dem Keypad gedrückt wurde.

on time löst ein Ereignis aus, wenn **time** den Wert **T-Wert** erreicht hat.
Bei Aufruf von **on time** wird die Systemvariable **time** neu gestartet.
time ist die Systemuhr und wird nach dem Einschalten des Basic-Computers jede Sekunde eins weiter gezählt.

Für einen zyklischen Aufruf eines Unterprogramms muss die erste Anweisung des Unterprogramms „ let time = 0 " sein.

ex.:

10 **on time 1000 gosub 100**	Am Programmstart wird das Ereignis definiert.
...	...
...	
...	
100 Anweisung 1	Ab hier wird das Ereignis **on time** abgearbeitet.
110 Anweisung 2	
....	
150 return	Mit **return** wird zu der Stelle zurückgesprungen, an der das Programm durch das Ereignis unterbrochen wurde. Siehe auch **return**.

open **open "Name"**
open Nummer

In pahlbasic gibt es keine Dateien, in denen man Daten speichern kann. Der Verwaltungsaufwand für den kleinen EEPROM-Speicher wäre gross. Statt dessen gibt es für jede Programm-Datei einen 2 (8) KByte grossen Datenbereich, der fest mit der Programm-Datei verbunden ist und nicht extra geöffnet werden muss. Es sei denn, man will von Programm A auf die Daten von Programm B zugreifen. Hier kommt **open** ins Spiel. Nach Ausführen von **open „Name"** ist der Datenbereich von Programm „Name" dem aktuellen Programm zugeteilt. Mit **close** wird wieder der eigene Datenbereich aktuell. Statt **„Name"** kann auch die Programm-**Nummer** eingesetzt werden, die man mit „**dir**" abfragen kann.

poke **poke Word-Adresse, Byte-Wert**

ex.:
10 **poke 4000, 234**

20 **poke A and 4096, 9 or C**

Schreibt den Wert *Byte-Wert* in die RAM-Speicherzelle mit Adresse *Word-Adresse*. Im Programm-Modus oder direkt.

print oder **?**

print NumVar
print StringVar
print StringKonstante
print Ausdruck
print bin Ausdruck
print hex Ausdruck

ex.:
? A
? not (A or (C and 255))
10 cls **:** locate 1**,** 1 **: print** "Hallo"
20 **print** $3 **;**

druckt den Inhalt von Variablen oder Textkonstanten oder Funktionen auf dem Terminal oder LCD aus. Print rechnet auch Ausdrücke wie A + B * C aus. Ein Semikolon am Ende der Anweisung unterdrückt den Zeilenvorschub.(Auf dem Terminal). Ein Komma setzt den Tabulator auf die nächste Position (für Tabellen).

ex.:
10 **print** " x"**, : print** "sin(x)"**, : print** "cos(x)"**, : print** "tan(x)"
20 for N = 0 to pi / 2 step pi / 20 **: print** N**, : print** sin (N)**,**
30 **print** cos (N)**, : print** tan (N) **:** next

Programm- oder Befehlsmodus.

read

read Numerische Variable
read Systemvariable
read Stringvariable

ex.:
10 **read A**
20 **read pwm1**
30 **read $2**

liest einen Wert oder eine Zeichenkette aus dem EEPROM-Speicher und weist ihn einer (System)Variablen zu. Der Wert muss zuvor mit **write** in den EEPROM-Speicher geschrieben worden sein. Achtung! Die Werte oder Zeichenketten müssen in der gleichen Reihenfolge ausgelesen werden, wie sie hineingeschrieben wurden. D.h. Man darf einer numerischen Variablen keinen String zuweisen (Und umgekehrt) Es wird kein Fehler angezeigt, es gibt nur unsinnige Ergebnisse. Natürlich darf man die Variable A speichern und als Variable B oder als Systemvariable time wieder einlesen. Siehe auch **open, close, write, restore** und **dptr.**

restore

restore
restore 0 [, Zeiger]
restore 1 [, Zeiger]

restore setzt die Datenzeiger für **read** und **write** auf Null (Anfang)

restore 0 setzt nur den Datenzeiger für **read** auf Anfang.
Wurde der Wert **Zeiger** angegeben, wird der Datenzeiger auf diesen Wert

gesetzt. (Wertebereich für **Zeiger:** 0...2043) (0....8187)

restore 1 setzt nur den Datenzeiger für **write** auf Anfang.
Wurde der Wert **Zeiger** angegeben, wird der Datenzeiger auf diesen Wert gesetzt. (Wertebereich für **Zeiger:** 0...2043) (0....8187)

ex.:
10 **restore 0 , 32 :** read A : read B

Wird der **write**-Zeiger zurückgesetzt, werden die alten Werte beim nächsten **write** überschrieben. Wird der **read**-Zeiger zurückgesetzt, kann man alle Werte (noch einmal) neu einlesen. Siehe auch: **open, close, read** und **write** sowie **dptr.**

return **return [Zeilennummer]**

Rücksprung vom Unterprogramm. Es wurde zuvor mit **gosub** in ein Unterprogramm gesprungen. Mit **return** wird an die Stelle zurückgekehrt, die dem Aufruf folgt. Ist die **Zeilennummer** angegeben, wird dorthin gesprungen und die mit **gosub** gespeicherte **Rücksprungadresse** verworfen.

ex.:
10 gosub 100 **'**Aufruf des Unterprogramms
20 Anweisung1
….
….
100 Anweisung2 **'**hier beginnt das Unterprogramm
....
150 return **'**Rücksprung (zur Zeile 20)

run **run ["Programmname"] [Programmnummer]**

ex.:
run
startet das im Speicher befindliche Programm. **run** löscht vorher alle Variablen wie clear.

run "alarmanlage"
Lädt das Programm „alarmanlage" in den Speicher und startet es.

run 3
Lädt das Programm Nr. 3 in den Speicher und startet es. Die Programmnummer mit **dir** ermitteln. **run** nur im Befehlsmodus.

save (sichern)

save 0 Speichert das im RAM befindliche Programm dauerhaft im internen EEPROM. Es muss mit run gestartet werden. Dient auch zum Löschen des EPROMS (nach **new**)

save 1 Wie save 0. Das Programm wird nach einem Reset sofort gestartet. Ist kein Programm im RAM-Speicher wird eine Fehlermeldung ausgegeben.

Achtung: Programme mit dem ATMEGA1284 können bis zu 8 Kbyte gross werden, ins interne EEPROM passen aber nur 4 Kbyte! Ist das Programm im RAM grösser als 4 Kbyte gibt es eine Fehlermeldung und wird nicht im internen EEPROM gespeichert. Mit dem ATMega 644 kann dieser Fehler nicht auftreten,

save ”Name” [, 1]

Ab Version 1.03 unterstützt pahlbasic **externes** EEPROM (ausserhalb des Controllers). Es können 31 (7) Programme unter ihrem Namen gespeichert werden. Der Zusatzparameter nach dem Komma speichert zusätzlich ein Autostart-Flag, so dass **ein** Programm automatisch nach dem Einschalten geladen und gestartet wird. Siehe auch **load**, **del**, **dir**.

ex.:
save ”Temperatur Alarm” , 1

ex.:
save (ohne Parameter)

Wurde das Programm mit **load** geladen, kann es (z.B. nach Bearbeitung) mit **save** wieder gespeichert werden, der Name muss nicht wiederholt werden.

sound **sound [Frequenz][, Dauer]**

ex.:
10 **sound 2000, 100**
20 let A = 440 'und jetzt eine Tonleiter
30 for N = 1 to12 **: sound A, 30 :** let A = A * (2 ^ (1 / 12)) **:** next

Erzeugt ein Rechtecksignal am Soundausgang mit der **Frequenz** in Hertz und der **Dauer** in 1/100 sek. (ex.**:** 2000 Hz, 1 Sekunde) Defaultwerte**:** Frequenz: 500 Hz, Dauer 100 ms. Minimale Frequenz**:** 120 Hz Maximale Frequenz**:** 15000Hz. Genauigkeit: ca. 1%.

wait **wait numer. Wert**

ex.:
wait 10 Wartet 1 Sekunde.
10 **wait A**
20 **wait ad1 / 5** auch so etwas ist möglich

Das Programm wartet die angegebene Zeit in Zehntelsekunden. Wertebereich 1 ... 65535.

write **write Numerische Variable**
write Systemvariable
write Stringvariable
write Zahlenwert (Ausdruck)
write ”Zeichenkette”

ex.:
10 **write** A
20 **write** time
30 **write** $5
40 **write** 3 * 9 + 7
50 **write** ”Uhrzeit: ” **: write** clock
60 **write #** 3 * sin(pi / 8) (siehe auch **val**)

write schreibt Werte und Zeichenketten in den ext. EEPROM-Speicher. Sie bleiben auch nach dem Ausschalten des Basic-Computers erhalten. Diese Funktion kann für das Daten-Loggen benutzt werden. Zu jeder im EEPROM-Speicher abgelegten Programm-Datei gibt es einen 2 (8) Kbyte grossen Datenspeicher. Ohne **externen** EEPROM-Speicher funktioniert **write** nicht!

Es werden nur die Werte, aber nicht die Variablen gespeichert. Siehe auch **read** und **restore** sowie **dptr**. Deshalb beim Auslesen der Werte die gleiche Reihenfolge einhalten, mit der die Werte gespeichert wurden. Nach jedem Schreibvorgang wird der Schreibzeiger um die Länge der Variablen verstellt. Ein Fliesskomma-Wert hat 4 Bytes eine Zeichenkette max. 21 Bytes. Der Speicherplatz von 2KByte reicht also für ca 500 Messwerte.
(Atmega1284**:** 8KByte)

rset (reset) **rset rel x** (x = 1...4)

ex.:
rset rel 1
Schaltet Relais 1 aus. Programm und Direktmodus.

set (setze) **set rel x** (x = 1...4)

ex.:
set rel 2
Schaltet Relais 2 ein. Programm und Direktmodus.

tgl (toggle) **tgl rel x** (x = 1...4)

ex.:
tgl rel 3

Toggelt das Relais 3. War Relais 3 eingeschaltet, wird es ausgeschaltet. War es ausgeschaltet, wird es eingeschaltet. Programm und Direktmodus.

imp (impuls) **imp rel x** (x = 1...4)

ex.:
10 let **pulse** = 20
20 **imp** rel3

Schaltet das Relais 3 für z.B. 20 Zeiteinheiten ein und danach wieder aus, die zuvor in der Systemvariablen **pulse** gespeichert wurden. Zeiteinheit ist 1/10 Sekunde. Defaultwert für pulse ist 10 (= 1 Sekunde). Programm und Direktmodus. (War das Relais vorher eingeschaltet, wird es für die Impulszeit ausgeschaltet.)

blk (blink) **blk rel x** (x = 1...4)

ex.:
blk rel1

Schaltet das Relais1 mit Blinkrythmus (1Hz) ein. Das Relais muss mit **rset rel1** wieder ausgeschaltet werden. Programm und Direktmodus.

Variablen

1.) numerische Variablen

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P

pahlbasic kennt die 16 vordefinierten Variablen A,B,C....P, die numerische Werte (Fliesskomma) aufnehmen können. Genauigkeit max. 7 Stellen / max. 5 Stellen nach dem Dez.-Punkt. Wissenschaftliche Notation ist nicht implementiert. Variablen müssen vor ihrer Verwendung nicht deklariert noch initialisiert werden. Ihr Startwert ist 0. (Die Atmega1284-Version kennt auch die Variablen Q....Z)

Beispiel für einen Wandlungsfehler: 1.2345 * 100 = 123.45001 Es werden keine „kosmetischen“ Rundungen durchgeführt.

Array:
Die Variablen **A – J** können auch als Array mit dem Namen **A** angesprochen werden. Das Array **A** besteht aus 10 Elementen mit den Indizes von **0 … 9**.

(ATMEGA1284-Version**:** zusätzlich Array B**:** Variablen K...T)

A(0) = A, A(1) = B, A(2) = C …. A(9) = J (B(0) = K ….)

ex.:
```
10 print ”x”, : print “sqr(x)”                    'Quadratwurzel-Tabelle
20 for P = 0 to 9 : let A(P) = sqr ( P + 1 ) : next
30 for P = 0 to 9 : print P + 1, : print A(P) : next
```

2.) Zeichenketten Variablen

$0, $1, $2, $2, $3, $4, $5, $6, $7 können jeweils 20 Zeichen aufnehmen. *)

ex.:
10 let **$0** = “Garage” **:** print **$0**

*) 80 mit Atmega1284

3.) Systemvariablen

ad1 ADC1
Wert des 1. Analog-Digital-Wandlers. 0 = 0Volt , 1023 = + 5 Volt. Nur lesen. Auflösung 10bit.

ex.:
10 print **ad1** / 1023 * 5; : print ” Volt“

ad2 ADC2
Wert des 2. Analog-Digital-Wandlers. 0 = 0Volt , 1023 = +5 Volt. Nur lesen. Auflösung 10bit.

ex.:
10 let A = **ad2** / 1023 * 5
20 if **ad2** < 512 then gosub 1000

ad3 ADC3
Wert des 3. Analog-Digital-Wandlers. 0 = 0Volt , 1023 = +5 Volt. Nur lesen. Auflösung 10bit.

ex.:
10 let A = **ad3** / 1023 * 5
20 if **ad3** < 512 then gosub 1000

buf Buffer
Anzahl der Zeichen im RS232 Empfangspuffer. Nur lesen.

ex.:
10 if **buf** > 0 then gosub 300
20 **on buf** gosub 1000

count Zähler
Zählerstand des Zähleinganges. Lesen und setzen.

ex.:
10 let **count** = 0
20 print **count**

clock Uhr
Uhrzeit der Systemuhr als Zeichenkette als HH**:**MM**:**SS. Nur lesen. HH = Stunden, MM = Minuten, SS = Sekunden.

ex.:
10 let $0 = **clock**
? **clock**
30 if $1 = **clock** then imp rel1 : locate 1,1 : print „Wecken“

dig1 Digitaleingang 1
Zustand des digitalen Eingangs 1. 1 = +5V , 0 = 0V. Nur lesen.

dig2 Digitaleingang 2
Zustand des digitalen Eingangs 2. 1 = +5V , 0 = 0V. Nur lesen.

dig3 Digitaleingang 3
Zustand des digitalen Eingangs 3. 1 = +5V , 0 = 0V. Nur lesen.

dig4 Digitaleingang 4
Zustand des digitalen Eingangs 4. 1 = +5V , 0 = 0V. Nur lesen.

dnr Nummer des aktuellen Dateisegments. Zu jeder Programm-Datei gehört ein Dateisegment. Mit **open** kann der Zugriff geändert werden Nur Lesen.

dow Day of week
Wochentag der Systemuhr. Werte: 1...7 . Montag = 1. Mit DCF77-Modul**:** nur Lesen (ohne Modul auch Schreiben).

dow$ **dow** als String
Statt einem Zahlenwert liefert **dow$** den Wochentag als abgekürzten String: „Mo, Di, Mi, Do, Fr, Sa, So“. Mit DCF77-Modul**:** nur lesen.

dptr Datenzeiger
Nach einer Leseoperation **(read)**, kann man mit **dptr** den Lesezeiger

abfragen, nach einer Schreiboperation **(write)** den Schreibzeiger.

ex.:
10 read A **:** let N = **dptr :** read B **:** read C
20 write A **:** let M = **dptr**
30 restore 0, N **:** read A

date Datum
Datum der Systemuhr. Mit DCF77-Modul**:** nur lesen. **date** ist ein String mit dem Format**:** Wochentag, TT**.**MM**.**20JJ

day Tag. Mit DCF77-Modul**:** nur lesen.

erl Fehlerzeile
Nummer der Programmzeile, in der der letzte Fehler aufgetreten ist.

err Fehlernummer
Nummer des zuletzt aufgetretenen Fehlers.**err** = 0 heisst**:** kein Fehler aufgetreten. Nach der Abfrage von **err** wird **err** gelöscht. Lesen und setzen möglich – setzen erzeugt künstlich einen Fehler.

ex.:
10 on **err** gosub 1000
20 print "Eingabe **:** "; **:** input A **:** let A = 1 / A 'testweise 0
.... eingeben.
....
....
1000 return 20 'falsche Eingabe wiederholen.

free freier Speicher
free gibt den freien Programmspeicherplatz in bytes wieder. Nur lesen.
ex.:
print **free**

freq Frequenz
Wert der am Frequenzmess-Eingang gemessenen Frequenz des dort anliegenden (Recheck-)Signals. Max. Frequenz: ca. 15KHz

ex.:
? **freq**
30 let A = B * **freq** / 2 * pi

get **get** ist auch eine Systemvariable. **get** liefert den Ascii-Wert der zuletzt gedrückten Taste auf dem **Terminal**. Keine Taste = 0.

ex.:
10 do **:** loop until buf > 0 **:** let A = **get**

diese Zeile ahmt die Instruktion „ **get A** " nach. Sie hat den Vorteil jederzeit durch **on** unterbrochen werden zu können.

hour Stunden
Stunden der Systemuhr. Nur 24-Stundenmodus. Lesen und setzen. (Setzen nicht mit DCF77-Modul.)

key Taste
Ascii-Wert der zuletzt gedrückten Taste auf dem **Keypad**. Ansicht

des Keypads siehe Anhang. Wurde keine Taste gedrückt, liefert **key** den Wert 0.

ex.:
10 do **:** let A = **key :** loop until A > 0 **:** print chr(A)

wartet auf Tastendruck auf dem Keypad und druckt ihr Zeichen aus.

min Minuten
Minuten der Systemuhr. Lesen und setzen. (Setzen nicht mit DCF77-Modul.)

mon Monat. Mit DCF77-Modul**:** nur lesen.

pgm Nummer der im Speicher befindlichen Programm-Datei. Siehe auch **dir.** 0 = Programm wurde nicht geladen, sondern wird gerade neu im RAM erstellt. Nur Lesen.

ex.:
print **pgm**
10 load **pgm** +1

pulse Impuls

Schreiben in pulse:
Zeitwert für die imp-Anweisung. Impulslänge in 1/10 Sekunden.

ex.:
10 let **pulse** = 30 (=3 Sekunden, kleinster Wert**:** pulse = 1)
20 imp rel2 (Relais 2 zieht für 3 Sekunden an)

Lesen von pulse:
Impulslänge des am Zähleingangs anliegenden Signals. Auflösung ist 1/225 Sekunde = ca 4,4 ms. Maximalwert 220 Stunden. (32- Bit). Es wird immer die Impulslänge des letzten eingegangenen Impulses ausgegeben. **pulse** muss also gelesen werden, bevor der nächste Impuls einläuft.

ex.:
10 print **pulse** 'liefert die Impulslänge des count-Eingangssignals
20 if **pulse** > 225 then goto 100
30 let pulse = pulse + 10 'das liefert ein anderes Ergebnis als mit anderen (System-) Variablen.

pwm1 Pulsweitenmodulator 1
Momentaner Dutycyclewert des Pulsweitenmodulatorkanals 1. Setzen und lesen.

pwm2 Pulsweitenmodulator 2
Momentaner Dutycyclewert des Pulsweitenmodulatorkanals 2. Setzen und lesen

rel1 Relais 1
Zustand des Relais 1. 1 = Ein, 0 = Aus. Lesen und setzen.

rel2 Relais 2
Zustand des Relais 2. 1 = Ein, 0 = Aus. Lesen und setzen.

rel3 Relais 3
Zustand des Relais 3. 1 = Ein, 0 = Aus. Lesen und setzen.

rel4 Relais 4
Zustand des Relais 4. 1 = Ein, 0 = Aus. Lesen und setzen.

sec Sekunden
Sekunden der Systemuhr. Lesen und setzen. (Setzen nicht mit DCF77-Modul.)

sync synchron mit Funkuhr
Gibt Auskunft über den Synchronisierungszustand der Systemuhr. Die Uhr wird nach Einschalten des Basic-Computers sowie um 3 Uhr morgens synchronisiert.
Nach dem Empfang des1. korrekten Zeittelegramms ist **sync = 1**. Wird innerhalb einer Stunde ein zweites gültiges Telegramm empfangen, nimmt **sync** den Wert **2** an. Danach ist manuelles Setzen von Zeit und Datum nicht mehr möglich.
Andernfalls liefert **sync** den Wert **0**. Setzen von **sync** startet eine neue Synchronisation: **let sync = 0** (andere Werte werden ignoriert)

temp Temperatur
Temperaturwert des 1Wire-Temperatursensors (DS18S20) in Grad Celsius. Auflösung und Genauigkeit: 0,5 Grad Celsius. (9 Bit) Messbereich -55 °C .... 120 °C. (Wandlungsdauer**:** 800ms)

time Systemuhr
time ist die Systemuhr. **time** wird nach dem Einschalten des Basic-Computers jede Sekunde um 1 erhöht. Auflösung: 32 Bit.
ex.:

? **time**	Lesen
let A = **time**	Lesen
10 let **time** = 0	Setzen
20 if **time** > A then goto 200	Lesen

year Jahr. Mit DCF77-Modul**:** nur lesen. Datum und Wochentag werden von der Systemuhr nicht fortgeschrieben. Sie werden nur vom DCF-Signal geliefert.

Zugriff auf einzelne Bits

auf ein einzelnes Bit einer **numerischen** (System-)**Variablen** greift man mit dem Dezimalpunkt zu:

(System-)Variable . Bitnummer

ex.:
let A.15 = 1
print pwm1.1
10 for N = 0 to 15 : print C.N : next
20 if J.3 = 0 then ….

Die entsprechende Variable wird **vorher** in eine **16-Bit-Wort-Zahl** gewandelt. Der Wert hinter dem Punkt muss kleiner als 16 sein. (Bit 0….15) und darf eine Konstante oder Variable sein. Das Bit kann nur die Werte 0 oder 1 annehmen. (Alle Werte > 0 werden als 1 interpretiert)

Operatoren

arithmetrisch

keine Rechenregelbeachtung, bitte die gewünschte Reihenfolge mit Klammern erzeugen.

+ Addition
- Subtraktion
***** Multiplikation
/ Division
^ Potenz - ermöglicht auch z.B. das Ziehen der 12.Wurzel:
let A = 2 **^** (1 / 12)
mod liefert den Rest einer Division ganzer Zahlen.
<< bitweises links-schieben (Das Ergebnis ist ein Wordwert, der rechte Operand (der angibt um wieviel Bits verschoben wird) darf nicht grösser als 16 sein) let A = 16 **<<** 2 print B **<<** 3
>> bitweises rechts-schieben (sonst wie **<<**)
@ pointer, liefert die physische Adresse einer Variablen. print **@A**

logisch

not bitweises **not**, die Operanden werden in 16 Bit-Wortvariablen gewandelt. **not (** 0 **)** = 65535
not wird vom Interpreter wie eine Funktion behandelt.
ex.:
10 let A = **not (** B **)**
20 print **not (** 1 + C **)**

and bitweises **and**, das Ergebnis ist ein 16 Bit-Word-Wert.
ex.:
10 print 3 **and** 255
20 let A = B **and (** 5 or 8 **)**

or bitweises **or**, das Ergebnis ist ein 16 Bit-Word-Wert.
ex.:
10 print 3 **or** 8
20 let A = B **or (** 5 and 255 **)**

xor bitweises **xor**, das Ergebnis ist ein 16 Bit-Word-Wert.

Vergleich

> gròsser
< kleiner
>= gròsser gleich
<= kleiner gleich
= gleich
<> ungleich

Math. Funktionen

abs liefert den Absolutwert eines Wertes zurück.

ex.:
print **abs ((**-1.6**))** Ergebnis: 1.6
10 let B = **abs (** B **)**

atn berechnet den Arcustangens eines Radiant-Wertes.

ex.:
? **atn (** .3 **)**

cos berechnet den Cosinus eines Radiant-Wertes

ex.:
? **cos (.**234 **)**
10 print **cos ((** pi / 8 **)** + B **)**

hi liefert das höhere **byte** einer 16-Bit-Wort-Zahl.

ex.:
? **hi** (999)

lg berechnet den natülichen Logarithmus zur Basis **e**.

ex.:
? **lg (e)**

lo liefert das niedere **byte** einer 16-Bit-Wort-Zahl.

ex.:
? **lo** (999)

log berechnet den 10-er Logarithmus eines Wertes.

ex.:
? **log (** 100 **)**
10 let B = **log (** A + 99 **)**

peek liest ein Byte von der angegeben (RAM) Adresse.

ex.:
? **peek (** 2000 **)**
10 let B = 12
20 let A = **peek (** B **)**

rnd (round) rundet den angegebenen Wert auf die nächste ganze Zahl.

ex.:
? **rnd (** 1.6 **)** liefert 2 als Ergebnis
10 let A = **rnd (** B **)**

sin berechnet den Sinus eines Radiant-Wertes

ex.:
? **sin (** .234 **)**
10 print **sin (** (pi / 8) + B **)**

sqr berechnet die Quadratwurzel eines (positiven) Wertes

ex.:
? **sqr (** 13.56 **)**
10 let A = **sqr (** pi / 2 **)**

tan berechnet den Tangens eines Radiant-Wertes

ex.:
? **tan (** .234 **)**
20 print **tan (** pi / 8 **)**

Stringfunktionen

asc Statt **print asc ("A")** schreibt man in pahlbasic: **print ("A")** (d.h.**:** asc gibt es nicht)

bin **bin Zahl**
liefert eine Zeichenkette, die den Wert **Zahl binär** darstellt. **Zahl** ist eine 16-Bit-Wort-Zahl. Das Zeichen «%» kennzeichnet eine Binärzahl.

ex.:
print **bin** 4 + 333 Ergebnis**:** %101010001
let $2 = **bin** 255
print val $2

chr **chr Zahl**
liefert ein Zeichen mit dem **Ascii-Wert Zahl. Zahl** ist ein Bytewert.

ex.:
print **chr** 13 sendet Wagenrücklaufzeichen (Return)
let $1 = **chr** 10 weist einem String ein nicht druckbares Zeichen zu.
write **chr** 13 schreibt einen 1 Zeichen String ins EEPROM (als Fliesskommazahl)

hex **hex Zahl**
liefert eine Zeichenkette, die den Wert **Zahl** hexadezimal darstellt. **Zahl** ist eine 16-Bit-Wort-Zahl. Das Zeichen «&» kennzeichnet eine Hexadezimalzahl.

ex.:
print **hex** 3 * 9 Ergebnis**:** &1B

in **Stringvariable1 in Stringvariable2**
liefert als Ergebnis die Position in **String 2**, ab welcher **String1** dort vorkommt. Kommt **String1** nicht in **String 2** vor, liefert **in** den Wert 0.

ex.:
10 let $1 = "1234567890" **:** let $2 = "67"
20 **let A = $2 in $1**
30 **if $2 in $1** then print "gefunden an Position: "**; :** print A
40 **if $2 in $1 = 6** then print "richtige Position"

str **str Ausdruck**
str verwandelt das Ergebnis eines Rechenausdrucks in einen **String.**

ex.:
10 let **$1 = str A** verwandelt den Wert der Variablen A in einen String.
20 let **$4 = str 4 + (3 * B)** Rechenergebnis in einen String.
30 print $4 Ausdruck des Ergebnisses

val **val Stringvariable**
val liefert den numerischen Wert, den die **Stringvariable** repräsentiert. Der Inhalt von **Stringvariable** muss ein berechenbarer Ausdruck sein.
(Umkehrung von **str**) Ist der Inhalt von Stringvariable nicht numerisch, ist das Ergebnis 0.

ex.:
10 let **$1 = "125.45"** numerische Werte ohne Klammern
20 print **val $1**

30 let **$2 = # sin(pi / 8)** **Rechenausdrücke** müssen dem Interpreter
40 print **val $2** durch **#** mitgeteilt werden.

Die Zeichenkette wird dann in Tokens umgewandelt. Der String sieht dadurch «komisch» aus. **#** ginge auch im oberen Beispiel.

Richtig Sinn macht **val** mit **input#**. Dadurch ist es möglich, zur Laufzeit des Programms Formelausdrücke einzugeben, die durch das Programm berechnet werden. Ebenso ist es möglich mit **write** und **read** mathematische Bibliotheken abzuspeichern und durch ein und dieselbe Routine berechnen zu lassen.

val kann selbst nicht in mathematischen Ausdrücken benutzt werden:
let A = 3 * **val** $1 geht nicht! – statt dessen: let A = **val** $1 **:** let A = 3 * A

Vergleiche:

Strings können nur auf Gleichheit „**=**“ oder Ungleichheit „**<>**“ geprüft werden. Der **erste** Vergleichsoperand muss **immer** eine Stringvariable sein.
ex.:
10 if **$1 = $2** then print ”Die Namen sind gleich”
20 print **$2 <> ”Montag“**
30 if ”pahlbasic” = $3 then.... in pahlbasic nicht erlaubt
40 if date = ”Mo, 01.12.2011” then.... geht mit pahlbasic nicht, statt dessen**:**
40 let **$1 = ” Mo, 01.12.2011” :** if **$1 = date** then...

Stringvariable(Position)

greift auf ein einzelnes Zeichen innerhalb eines Strings zu. Man kann es lesen oder verändern. Einzelne Zeichen sind Bytes also auch Zahlen – je nach Interpretation. Man kann mit ihnen auch rechnen. pahlbasic interpretiert einzelne Zeichen stets als Zahlenwert (Ascii-Wert), da alle Ausdrücke von einer Rechenroutine ausgewertet werden. (Ausgenommen sind nur Zuweisungen an eine Stringvariable.)

ex.:
10 let **A = $2(5)** hier ist „**=**“ das Zuweisungszeichen
Unterschied zu Standart Basic:
20 print **$2(5)** **<----** Druckt nicht das Zeichen selbst aus, sondern seinen **ASCII-Wert**. Daher auch keine eckigen Klammern.

40 print **chr $2(5)** Mit **chr** erhält man aber auch das Zeichen selbst. (**write** funktioniert wie print.)

50 if **$2(5) = ”A”** then.... und so prüft man auf ein Zeichen an bestimmter Position
(„**=**“ ist hier Operator - wird vom Interpreter auch anders codiert als im 1. Beispiel)

oder:
60 if **$2(5) = 65** then....

70 for A = 1 to 20 **: let $1(A) = $1(A) + $2(A) :** next 'vielleicht eine Verschlüsselung?

Innerhalb der Klammer dürfen nur Konstanten oder Variablen stehen.

Konstanten

pi Die Kreiszahl**:** 3.1415927

ex.:
?sin **(pi** / 4 **)**

e Die Euler'sche Zahl**: e** = 2.71828183.

ex.:
?lg **(e)**

true intern verwendet für das Ergebnis eines Vergleichs. **true** = 65535

ex.:
print 4 > 3 Ergebnis: 65535

false intern verwendet für das Ergebnis eines Vergleichs. **false** = 0

ex.:
print 3 > 4 Ergebnis: 0

Zahlen

ausser 10-stelligen Fliesskommazahlen kennt pahlbasic auch Hexadezimalzahlen und Binärzahlen.

Hexadezimalzahlen werden durch ein vorangestelltes „**&**“ gekennzeichnet.
ex.:
&F00C (dez. 61452) max. 4 Stellen

Binärzahlen werden durch ein vorangestelltes „**%**“ gekennzeichnet.
ex.:
%10010011 (dez. 147) max. 16 Stellen

Fehlermeldungen

1	**illegal direct**	nicht im Direktmodus (if, return,on...)
2	**syntax error**	eine Anweisung ist falsch (geschrieben) In vielen Fällen gibt der Interpreter einen Hinweis auf die Position des Fehlers in der Form: ….....x wobei x die Fehlerstelle markiert. (wie mcs51 basic) bei der Eingabe, sonst zur Laufzeit.
3	**expected ') '**	eine geöffnete Klammer wurde nicht geschlossen
4	**expected ' to '**	to oder Endwert fehlt in for
5	**next without for**	ein “for” fehlt
6	**return without gosub**	ein gosub weniger als returns
7	**expected ' = '**	= Zeichen in let oder for fehlt.
8	**line number?**	Zeilen-Nummer fehlt, ist Null oder existiert nicht.
9	**stack error**	exit wurde ausgeführt, obwohl keine Adresse auf dem Stapel war
10	**dcf77 active**	Funkuhr ist synchronisiert, Zeit-einstellung nicht mehr erlaubt
11	**not in prog mode**	Nur im Direktmodus erlaubt. (new, run, cont, list, save, dir, del)
12	**can't continue**	Nach einem Programmfehler kann nicht mit cont fortgesetzt werden.
13	**bad argument**	Falsches Argument oder Division durch 0, . z.B.: let A = $0
14	**out of memory**	zu viele Klammern, for/next-, do/loop -Schleifen, gosubs (max 5 Ebenen) oder Programmspeicher voll.
15	**loop without do**	ein do fehlt.
16	**no program**	kein Programm im RAM-Speicher bei save
17	**no ext. eeprom**	kein externes EEPROM vorhanden.
18	**prog not found**	Programm nicht gefunden
19	**number too big**	Diese Zahl ist für pahlbasic zu gross (> 10 000 000)
20	**user break**	Das Programm wurde beendet.

Unterschiede zu anderen Tiny Basic-Dialekten

Es gibt zwar Basic-Normen: ISO und ANSI für minimales und Full-Basic, aber kaum ein Softwarehersteller hält sich daran. Insbesondere bei Software für Microcontroller kocht jeder sein eigenes Basic.

pahlbasic hält sich in vielen Fällen an Tiny Basic-Vorbilder wie MCS51 – Basic. Aber bedingt durch die unterschiedliche Hardware, wurden bewusst Änderungen vorgenommen. Einer der wichtigsten Gründe für Abweichungen ist der kleine zur Verfügung stehende RAM-Speicher, da pahlbasic auf einem AVR-Controller (ATMEG644) ohne externen Ram-Speicher läuft. Die Version 1.03 läuft schon auf einem „nackten" Controller mit lediglich 14,7456 Mhz – Quarz und RS232 – Signalwandler.

Unterschiede sind:

Variablen sind statisch, sie sind schon vor ihrer Verwendung vorhanden. pahlbasic ist nicht sehr schnell, das Suchen nach Variablen im Datenspeicher bräuchte zusätzliche Zeit. Es können auch keine individuellen Namen vergeben werden. Der kleine RAM-Speicher wäre schnell voll. Es stehen lediglich 4 Kbyte RAM zur Verfügung, davon werden 2Kbyte für Variablen und Basic verwendet und 2Kbyte bleiben für das eigentliche Basic-Programm. Mit der Version mit Atmega1284 sieht es etwas günstiger aus.

Die Genauigkeit der **Fliesskommazahlen** ist nur 7 Stellen, sollte aber für einfache Steuerungen reichen. Maximaler Zahlenwert**:** 10000000. Der Fehler wächst mit jeder weiteren Stelle bleibt aber unter 0.01%.
z.B. 9 Stellen: Eingabe: 1234567.89 wird von pahlbasic zu: 1234567.75 gewandelt.

Stringvariablen sind max. nur 20 Zeichen lang. Wichtigstes Ausgabemedium im Betrieb ist das LCD-Display und das hat eine Zeilenlänge von 20 Zeichen.Gilt nicht für die 1284-Version.

Kommentare beginnen mit dem Hochkomma (') und nicht mit der Anweisung rem. Kommentare machen nur Sinn, wenn der Programmtext mit einem Editor (z.B. Basic52) erstellt wird. Ab dem Zeichen (') wird alles ignoriert bis zum Zeilenende (Zeichen 13, Return) Daher nicht für die Online-Eingabe.

Negative Zahlen (Konstanten, Literals) müssen in Klammern gefasst sein, auch wenn bereits eine Klammer (z.B. für eine Funktion) existiert: sin(-1) ergibt einen Syntaxerror, richtig ist in pahlbasic : sin((-1)). Der Interpreter unterscheidet streng, ob die Klammer zur Funktion oder zur negativen Konstante gehört. Diese Schreibregel sorgt aber für Klarheit: 3 * (-1) statt 3 * -1.

Stringfunktionen gibt es in pahlbasic nur wenige, es soll gesteuert und gerechnet werden. Textverarbeitung können PC's besser.

Beispiele für Abweichungen:

asc	gibt es in pahlbasic nicht. print asc("A") entspricht in pahlbasic : print ("A"). Mit **$x(Position)** kann man den ASCii-Wert eines einzelnen Zeichens (in einem String) ermitteln. Ausserdem kann pahlbasic mit Zeichen rechnen. let A = 3 * ("c" + $1(5))
data / read / restore	Data gibt es nicht in pahlbasic dafür gibt es in pahlbasic write, read und restore. Viel flexibler! Nur wenn man Tabellen braucht...
instr	die Funktion **instr** heist in pahlbasic **in**.

input# ist nicht auf einen Kanal oder File bezogen (wie bei Q-Basic) sondern bezieht sich auf die Eingabe von mathematischen Formelausdrücken.

inkey inkey in anderen Basic-Dialekten holt ein einzelnes Zeichen vom Terminal. pahlbasic unterstützt ein 20-Tasten Keypad, eine Eingabe vom Keypad - **in**put **key**pad - ist daher inkey.Für das Terminal gibt es **get**.

let ist in den meisten Basic-Dialekten optional und wird weggelassen – nicht in pahlbasic.

mid oder **mid$** gibt es in pahlbasic nicht – dafür gibt es **$n(N)** für einzelne Zeichen und die Kopierfunktion von **let: $n, Startposition, Anzahl** für Teilzeichenketten.

on Variable gibt es in pahlbasic nicht – dafür kann man Zeilen-Nummern berechnen, ähnliches gilt für **select case**.

ex.:

statt:	10 on A goto 100, 110, 120, 130, 140, 150	(29 Bytes)
pahlbasic:	10 if (A > 0) and (A < 7) then goto 10 * A + 90	(24 Bytes)

open / close ist in pahlbasic ein sehr spezielles Konstrukt, da es nur ein rudimentäres «Dateisystem» gibt.

while / wend gibt es in pahlbasic nicht – man kann mit **do loop**, wenn die Bedingung richtig gewählt wird, dasgleiche erreichen, ausserdem kann man mit einer **if / then / exit** Anweisung zu Beginn der Schleife prüfen und vorzeitig verlassen (nicht schön aber praktisch).

ex.:

```
10 while dig1 = 0 : wend          'wartet auf Digitaleingang 1
```

ist äquivalent zu:

```
10 do : loop until dig1 = 1
```

Ausblicke:

ab Version 1.4 wird der Atmega 644 nicht mehr unterstützt, statt dessen nur Atmega1284.

Für die Version 1.4 geplant:

1.) das alphanumerische LCD wird durch ein graphisches ersetzt. Dafür gibt es Graphikanweisungen und eine Oszilloskopfunktion.
2.) Anschlussmöglichkeit für Porterweiterung, Digital-Analog-Wandler, zweiten Temperatursensor, Empfänger für Infrarot-Fernbedienungssignale.

Für die Version 1.5 geplant:

1.) Netzwerkanschluss, Programmieren und Steuern übers Netzwerk.
2.) Datenspeicherung auf SD-Card.

Anhang 1

Keypadansicht

1	2	3	Del	+
4	5	6	(	-
7	8	9	)	*
.	0	Return	esc	/

Abb. 1

Anhang 2

Anschlussbelegung des Prozessors gemäss Version 1.03c

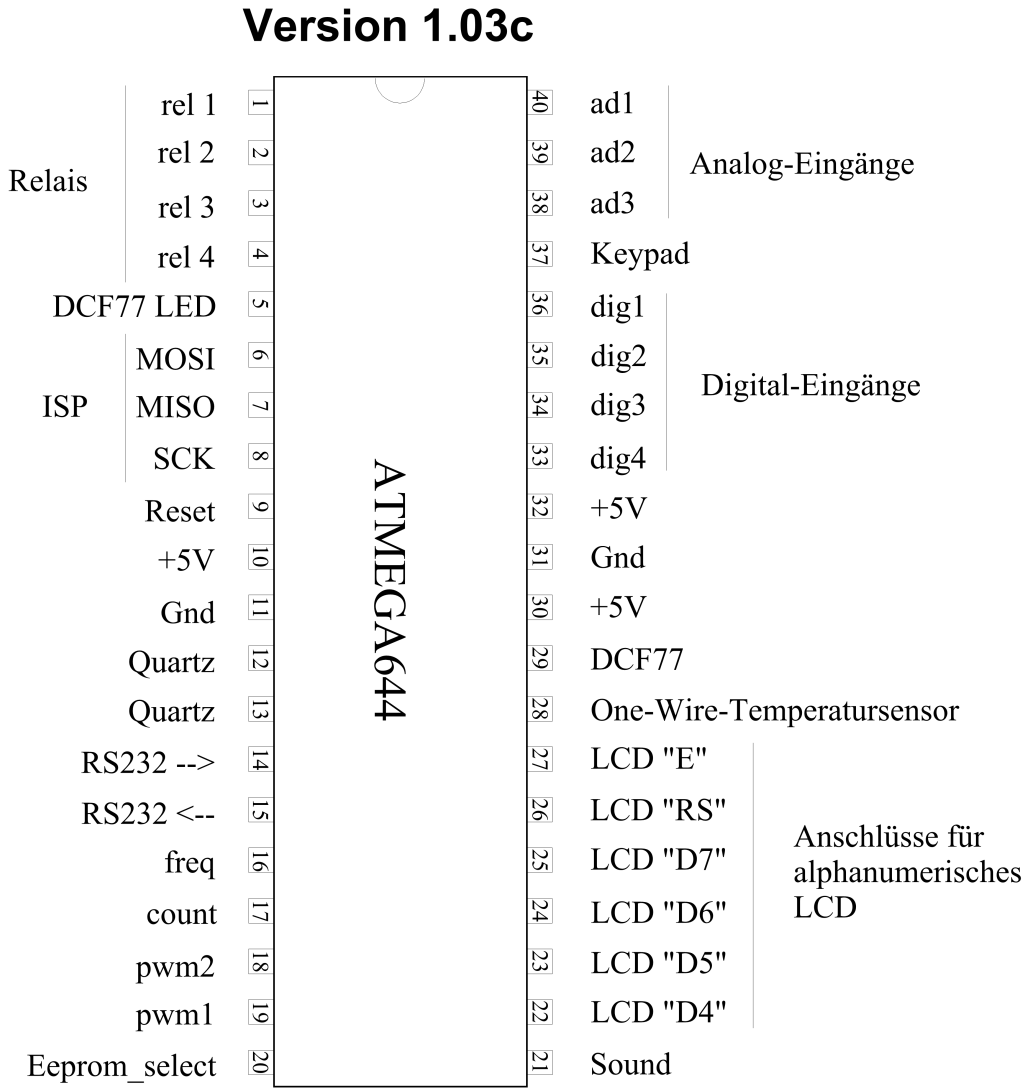


Abb. 2
Inzwischen wird der pinkompatible ATMega1284P eingesetzt. Mehr Speicherplatz mit mehr Möglichkeiten.

Anhang 3

Beschreibung der Anschlüsse

Nr.:	Name	Beschreibung
1	rel1	Hier wird das Relais 1 mittels Transistor angeschlossen.
2	rel2	Hier wird das Relais 2 mittels Transistor angeschlossen.
3	rel3	Hier wird das Relais 3 mittels Transistor angeschlossen.
4	rel4	Hier wird das Relais 4 mittels Transistor angeschlossen. Anschlussprinzip Relais: Abb.: 6
5	DCF77-LED	Hier wird eine LED angeschlossen, die die Aktivität des DCF77-Empfangs (Funkuhrantenne) anzeigt. Abb.
6	MOSI	Master Out / Slave In (SPI-Daten-Ausgang für EEPROM-Speicher)
7	MISO	Master In / Slave Out (SPI-Daten-Eingang für EEPROM-Speicher)
8	SCK	SPI Taktausgang für EEPROM-Speicher Abb.: 5
9	Reset	Reset-Eingang des Prozessors (negative Logik) Abb.: 7
10	+5V	Versorgungsspannung 5 Volt +/- 1% Abb.: 8
11	Gnd	Ground (Masse) Minuspol der Versorgungsspannung. Abb.: 8
12	Quartz	Anschluss des Schwingquarzes für die Takterzeugung
13	Quartz	Frequenz: 14,7456 Mhz Abb.:9
14	RS232 -->	Empfangsleitung der seriellen Schnittstelle (RxD)
15	RS232 <--	Sendeleitung der seriellen Schnittstelle (TxD) zur Verbindung zum Terminal. Abb.: 16
16	freq	Frequenzmess-Eingang. Hier kann ein Signal eingespeist werden, dessen Frequenz gemessen werden kann. Abb.
17	count	Zähleingang. Impulse an diesem Eingang werden gezählt und ihre Dauer gemessen. Abb.: 17
18	pwm2	Pulsweitenmodulator 2. Ausgang gibt pulsweiten moduliertes Signal aus zur Leistungs-Steuerung von Verbrauchern.
19	pwm1	Pulsweitenmodulator 1. Wie pwm2. Abb.: 15
20	Eeprom_select	Chip-select-Signal für 25LC1024 128KByte EEPROM-Speicher Aktiv low. Abb.: 5
21	Sound	Soundausgang. Hier kann z.B. ein Lautsprecher angeschlossen werden. Abb.: 10
22	LCD D4	an die „LCD“-Anschlüsse kann ein alphanumerisches LCD-Display
23	LCD D5	angeschlossen werden. Maximal 4 Zeilen x 20 Spalten
24	LCD D6	Abb.:11
25	LCD D7	
26	LCD RS	
27	LCD E	
28	OneWire	Hier kann ein (Dallas) DS18S20 Temperatursensor angeschlossen werden mit 9Bit Auflösung in nicht parasitärer Beschaltung. Abb.:12
29	DCF77	Anschluss für Funkuhrantenne (DCF77) positives Signal.
30	+5V	siehe oben Abb.: 8
31	Gnd	siehe oben Abb.: 8
32	+5V	siehe oben Abb.: 8
33	dig4	Digital-Eingang 4. Anschluss für digitale Signale. z.B. Kontakte
34	dig3	Digital-Eingang 3 5 Volt = 1; 0 Volt = 0
35	dig2	Digital-Eingang 2 Anschlüsse siehe Abb.:13
36	dig1	Digital-Eingang 1
37	keypad	Anschluss für eine kleine Tastatur (20-Tasten-Keypad). Abb.: 14
38	ad3	Spannungsmesseingang 3. Anschlussprinzip: Abb.: 13
39	ad2	Spannungsmesseingang 2.
40	ad1	Spannungsmesseingang 1.

Anhang 4

Schaltplan des Keypads

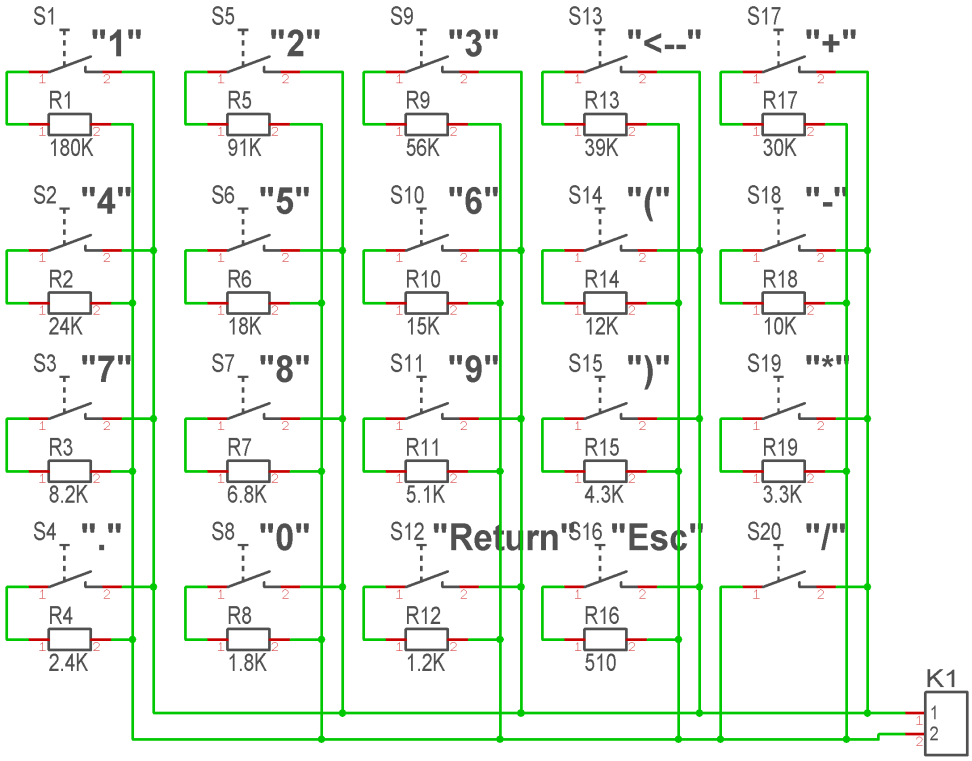
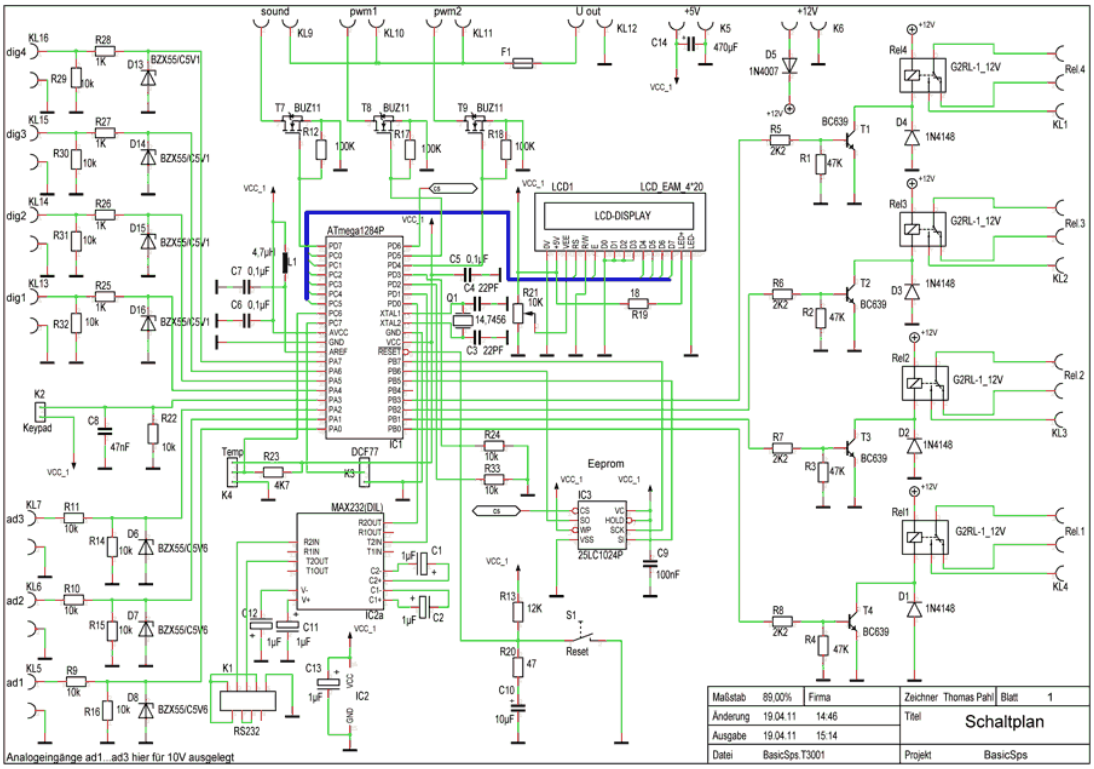


Abb. 3

Jeder Taster schaltet nur einfach einen anderen Widerstandswert an K1. Der Prozessor misst diesen Wert mit seinem Analog-Digital-Wandler 225 mal je Sekunde. (Genauer: der jeweilige Keypad-Widerstand bildet mit dem 10KOhm-Widerstand auf dem Controllerboard einen Spannung-steiler, dessen Spannung gemessen wird)

Anhang 5



Schaltplan des Controllerboards

Abb. 4

Sieht auf den ersten Blick kompliziert aus. Sind aber viele Wiederholungen drin. Teile der Schaltung, die nicht benötigt werden, können weggelassen werden (Ausgänge) oder durch einen simplen Widerstand ersetzt werden (Eingänge). Die verschiedenen Schaltungsbereiche werden einzeln besprochen. Die minimale Beschaltung benötigt die RS232-Schnittstelle, den Quarz und 5Volt Spannung sowie ein paar Widerstände. Siehe dazu die Abbildungen und Beschreibungen zu den Prozessoranschlüssen. Das EEPROM ist mittlerweile anders als in der Zeichnung ein 25LC1024P auch der Prozessor hat sich geändert: es ist jetzt der pinkompatible ATMEGA1284P mit mehr Flashspeicher und mehr RAM.

Anhang 5 Anschaltungen

Anschaltung des EEPROMS

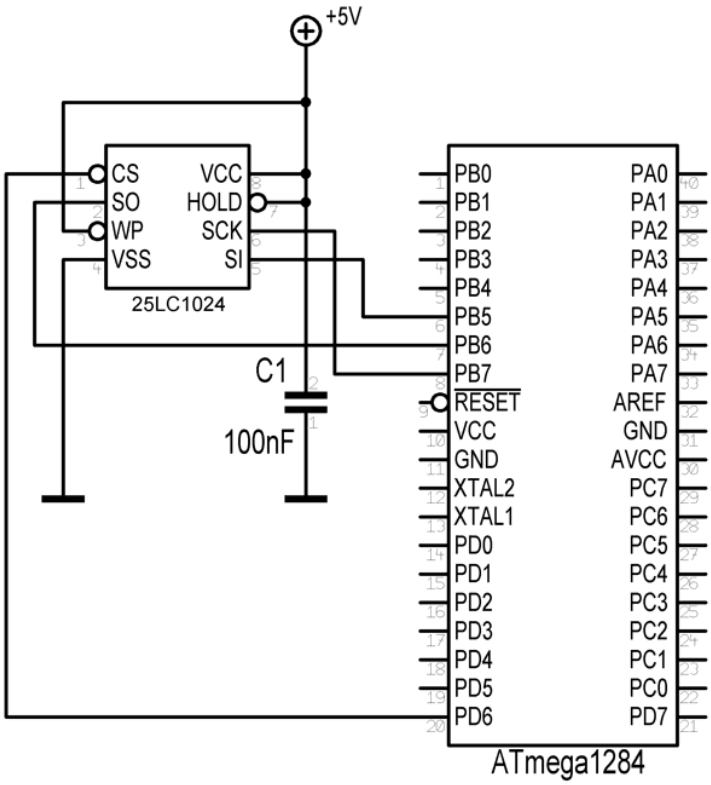


Abb. 5

Das EEPROM benötigt – ausser der Stromversorgung – eine Verbindung zum SPI des Prozessors und ein Chip-Select-Signal (PD6). Es empfiehlt sich die Verwendung eines Chipsockels mit integriertem 100nF-Kondensator. Das EEPROM dient zum Speichern von Programmen und Daten. Sozusagen die „Festplatte“ des Basic-Computers. Es fasst 128 KByte entsprechend 7 Programm-Dateien mit 7 Dateisegmenten.

Anschluss der Relais

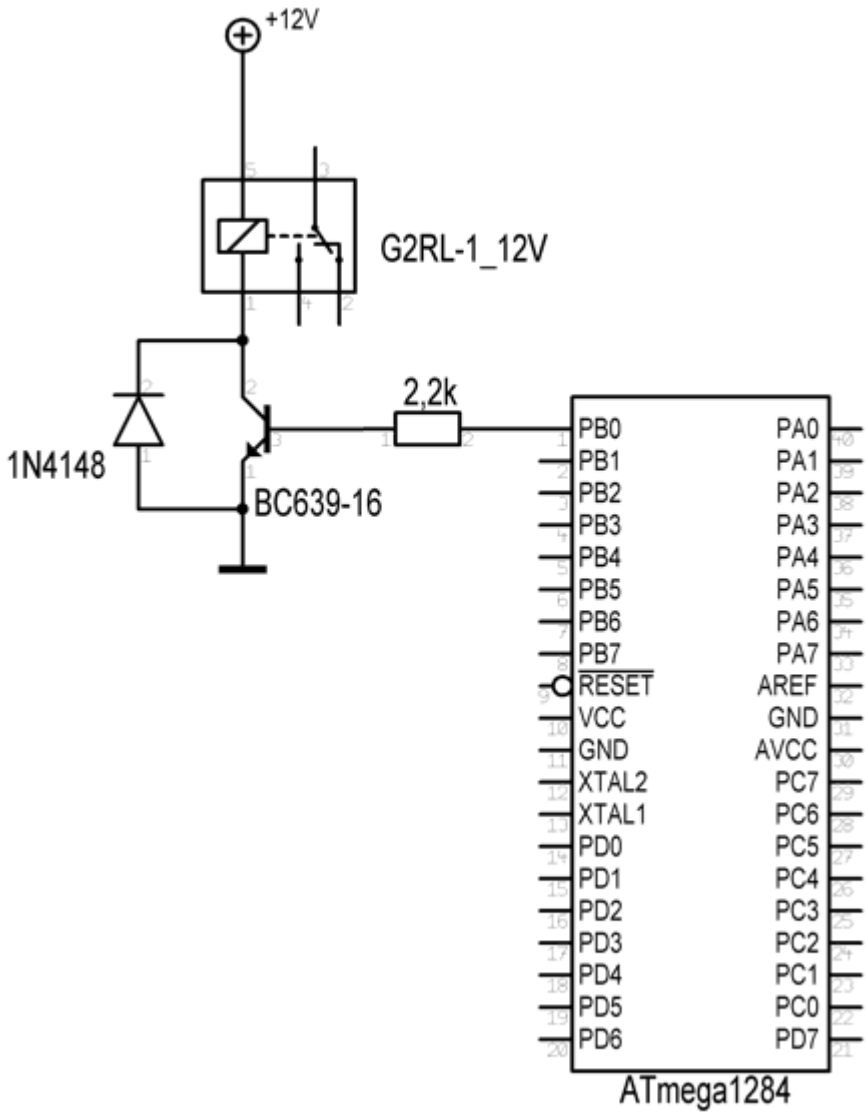


Abb. 6

Selbstverständlich können auch Relais mit anderer Spannung angeschlossen werden. Dann natürlich keine 12V Zusatzspannung! Ja es können auch andere Verbraucher, die maximal 0,5 A verbrauchen direkt angesteuert werden. (Gilt für die angegebenen Transistoren – bei Verwendung stärkerer Transistoren können auch höhere Ströme geschaltet werden.) Die Diode wird nur bei induktiven Verbrauchern benötigt. Die Relais 2...4 werden genauso an Pin 2....4 angeschlossen.

Anschaltung des Resetknopfes

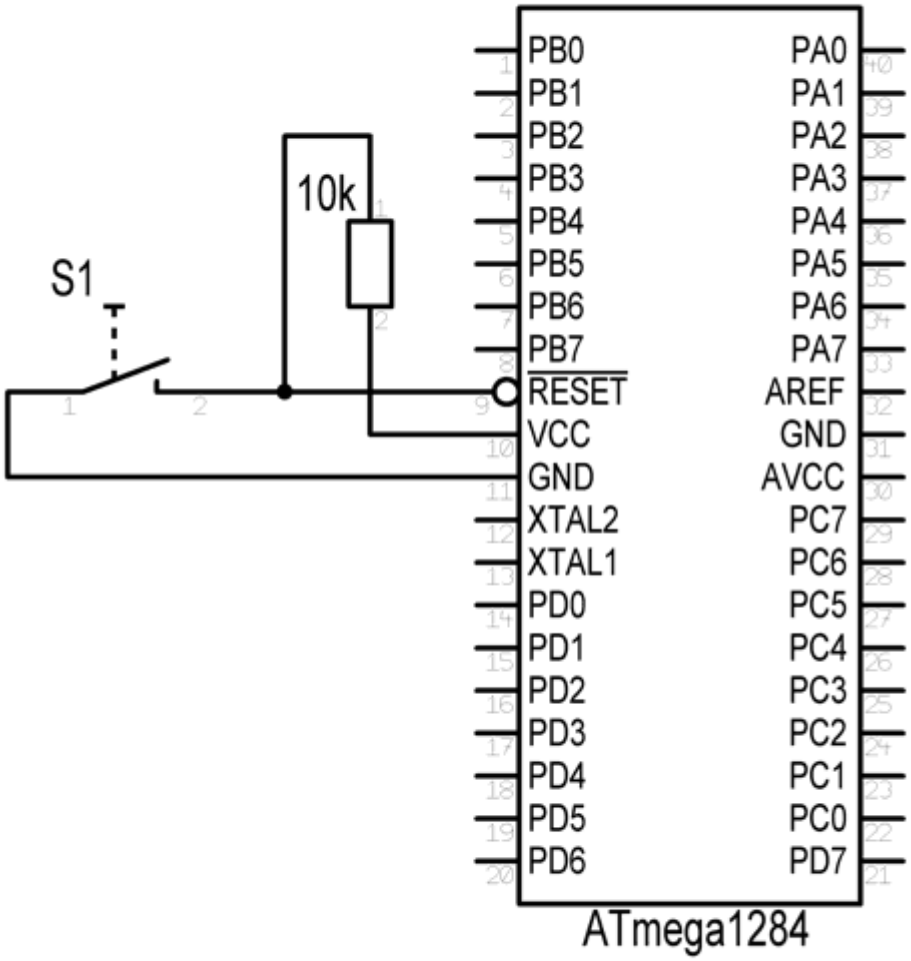


Abb. 7

Minimalbeschaltung des Reset-Eingangs. Wird kein Resettaster benötigt, den Reset-Eingang mit VCC verbinden. (bei Anlegen der Betriebsspannung wird automatisch ein Reset-Signal erzeugt.)

Stromversorgung des Prozessors

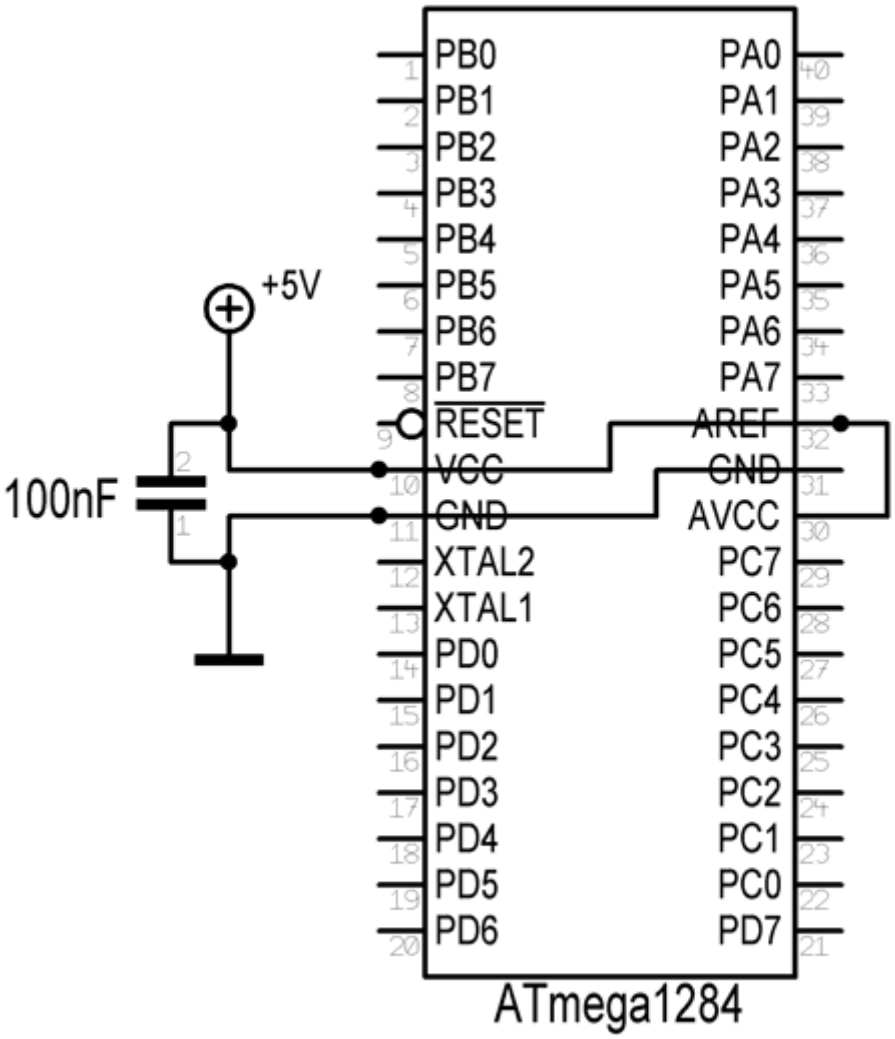


Abb. 8

Minimalbeschaltung der Stromversorgung ohne besondere Entstörungsmassnahmen. Der 100nF-Kondensator sollte direkt am Chip-Sockel angebracht werden.

Anschluss des Quarzes

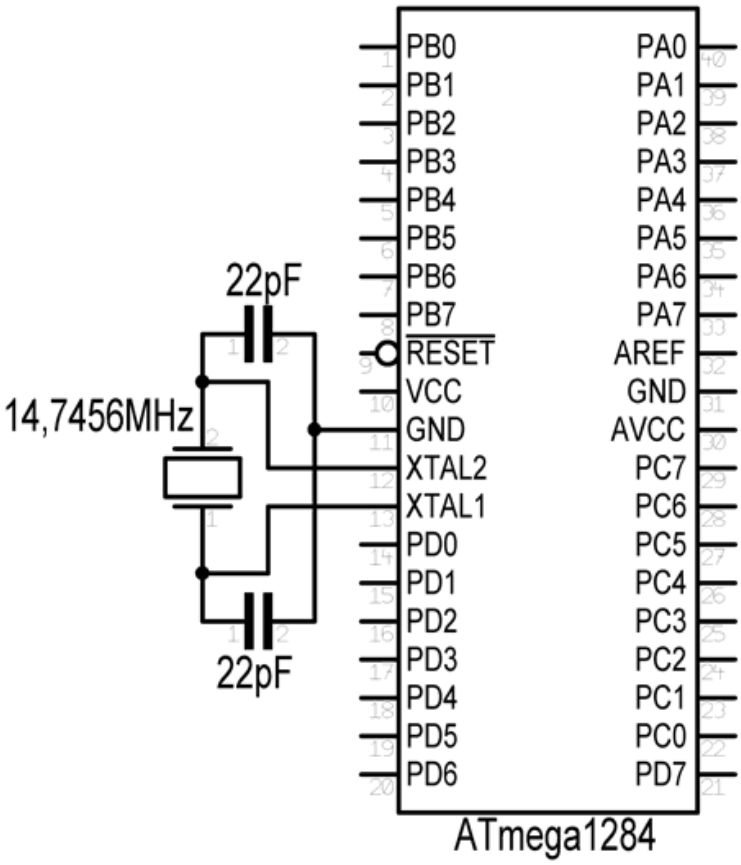


Abb. 9

Die Oszillator-Fusebit muss auf „Full Swing Oscillator“ gesetzt werden. Mit anderen Quarzfrequenzen funktionieren weder die Terminalverbindung noch stimmen die in der Anleitung angegebenen Zeit- und Frequenzangaben.

Soundausgang

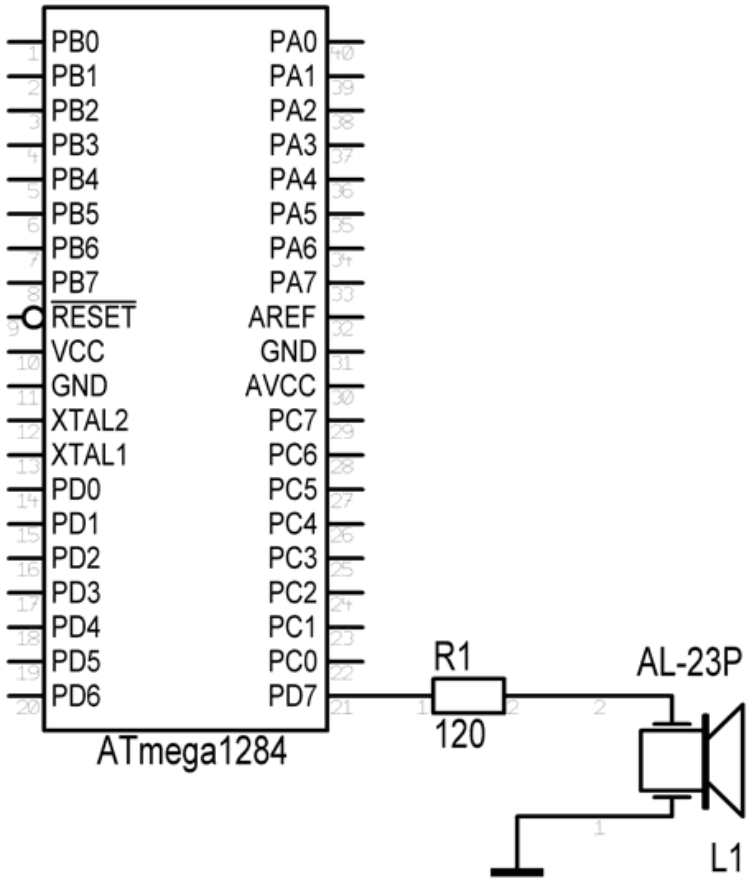


Abb. 10

Wenn keine grosse Lautstärke benötigt wird, kann man einen Lautsprecher 8 - 32 Ohm direkt wie abgebildet anschliessen. R1 darf auf keinen Fall weggelassen werden. (Ausnahme**:** der Lautsprecher hat selbst einen ohmschen Widerstand von > 100 Ohm)

Displayanschluss

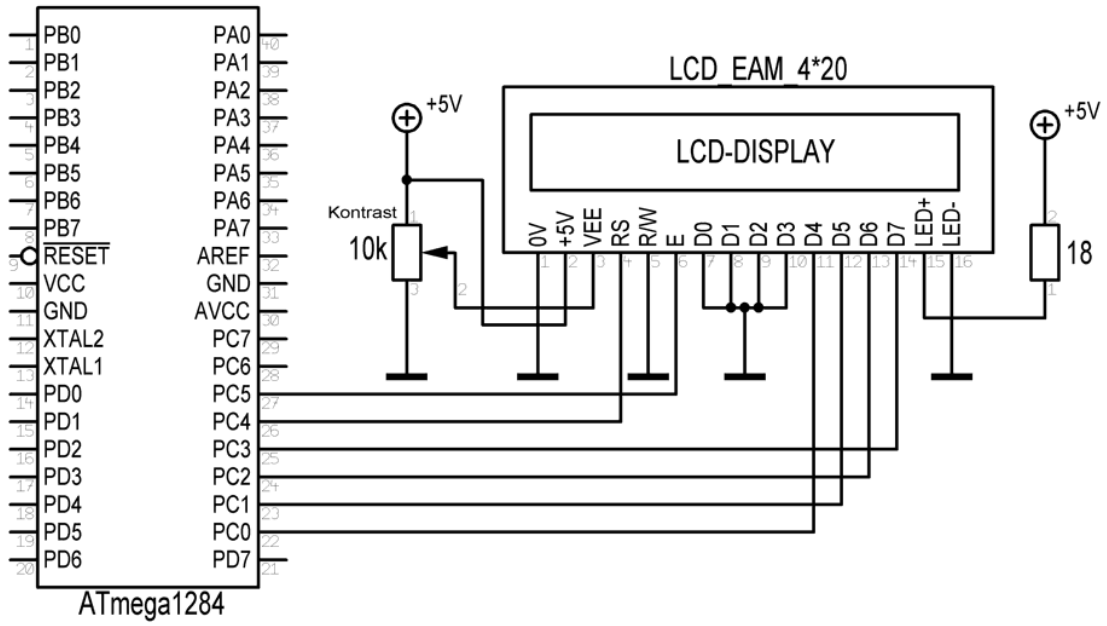


Abb. 11

Es können auch LC-Displays mit anderer Zeilen- und Spaltenzahl benutzt werden auch ohne Beleuchtung. Standard ist ein Display mit 4 Zeilen zu 20 Zeichen. Die Beleuchtung ist ständig eingeschaltet. Ein- und Ausschalten der Beleuchtung ist von der Software nicht vorgesehen. (es kann hierfür aber ein Relais- oder ein PWM-Ausgang verwendet werden)

Die inkey-Anweisung ist für das 4x20-Zeichendisplay ausgelegt. Wird ein Display mit geringerer Spaltenzahl verwendet, sollte der Parameter „Anzahl Zeichen“ oder „Anzahl Ziffern“ unbedingt angegeben und der Spaltenzahl angepasst sein. Sonst kann es sein, dass die Eingabe auf dem Display „zerrissen“ wird.

Das Display muss einen HD4470 kompatiblen Controller haben. (Industriestandart)

Anschluss Temperatursensor

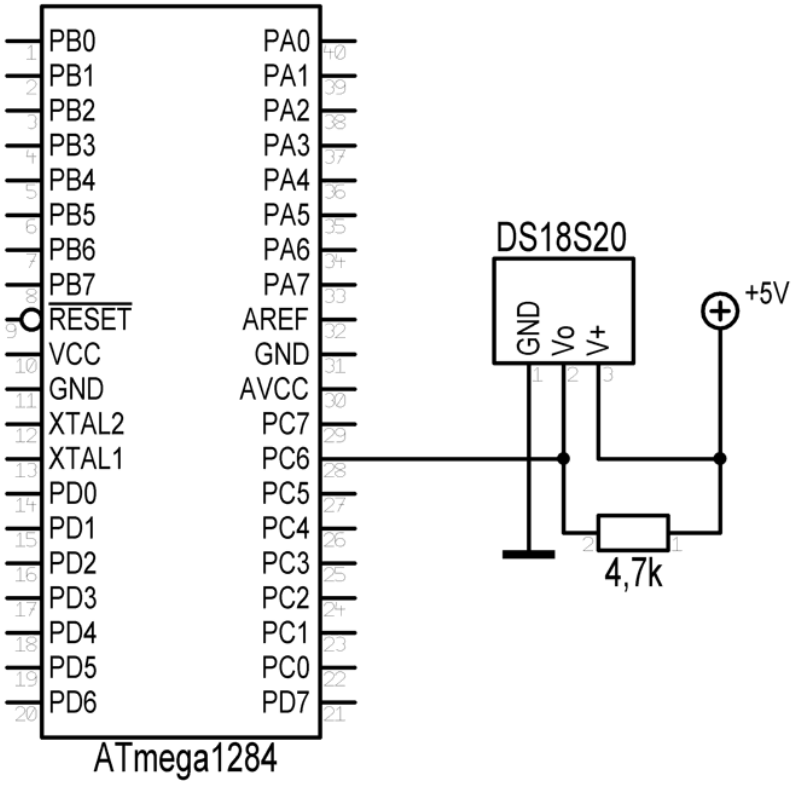


Abb. 12

Es kann nur ein Temperatursensor DS18S20 an den Pin 28 angeschlossen werden. Parasitäre Stromversorgung ist nicht vorgesehen. Wird der Temperatursensor nicht benötigt, nur den 4,7K Widerstand anschliessen.

Die Anschlussfolge des Temperatursensors im Schaltbild Abb. 12 stimmt mit seinem bedrahteten Plastikgehäuse überein. (Blick auf die flache, beschriftetete Seite von links nach rechts)

Eingangsbeschaltung

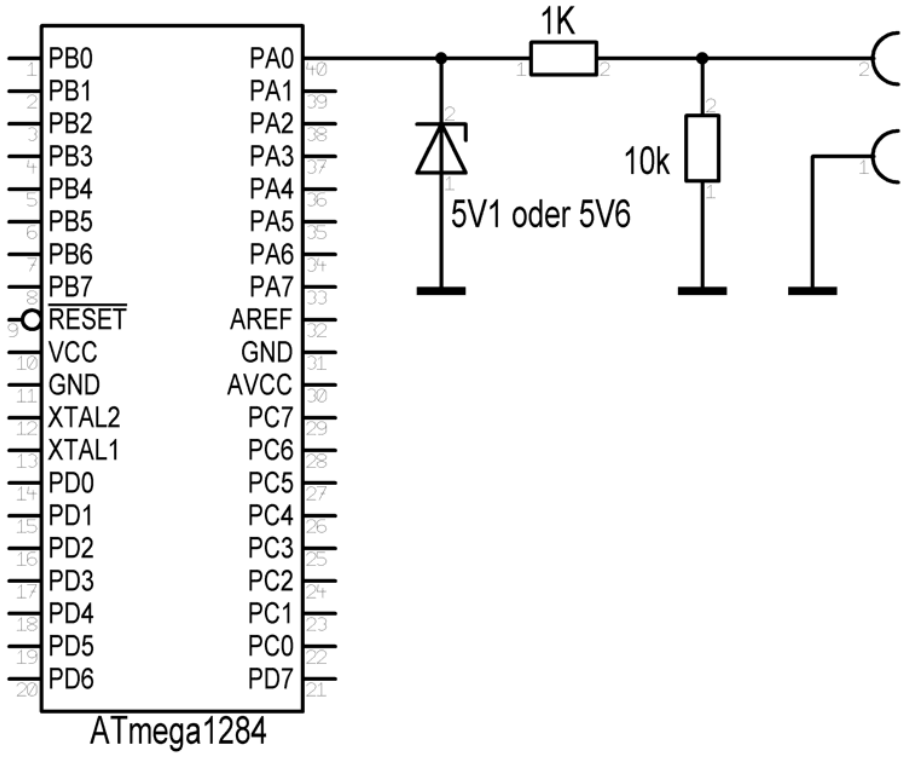


Abb. 13

Eingangsbeschaltung für die Digitaleingänge dig1...dig4 sowie für die analogen Messeingänge ad1....ad3. Die Eingangsschaltung enthält einen Überspannungsschutz mit 1KOhm und 5V1 Zenerdiode, sowie einen definierten Eingangswiderstand von 10KOhm. Bei langen Anschlussleitungen empfiehlt sich, einen Kondensator von 47nF parallel zum 10 KOhm Widerstand zu schalten. Die maximale Eingangsspannug beträgt 5 Volt. Bei den analogen Eingängen ad1....ad3 Zenerdioden 5V6 einsetzen. Nicht benötigte Eingänge mit Masse brücken oder mit obiger Schutzschaltung versehen und nicht weiter beschalten.

Ist die Eingangsspannung durch Anschluss an eine elektronische Schaltung (z.B. CMOS oder TTL-Logik mit 5 Volt Betriebsspannung) bereits begrenzt, ist die Schutzschaltung nach Abb. 13 nicht nötig. Die entsprechende Schaltung kann dann direkt verbunden werden. Im Zweifelsfall sicherheitshalber nach Abb. 13 beschalten.

Keypadanschluss

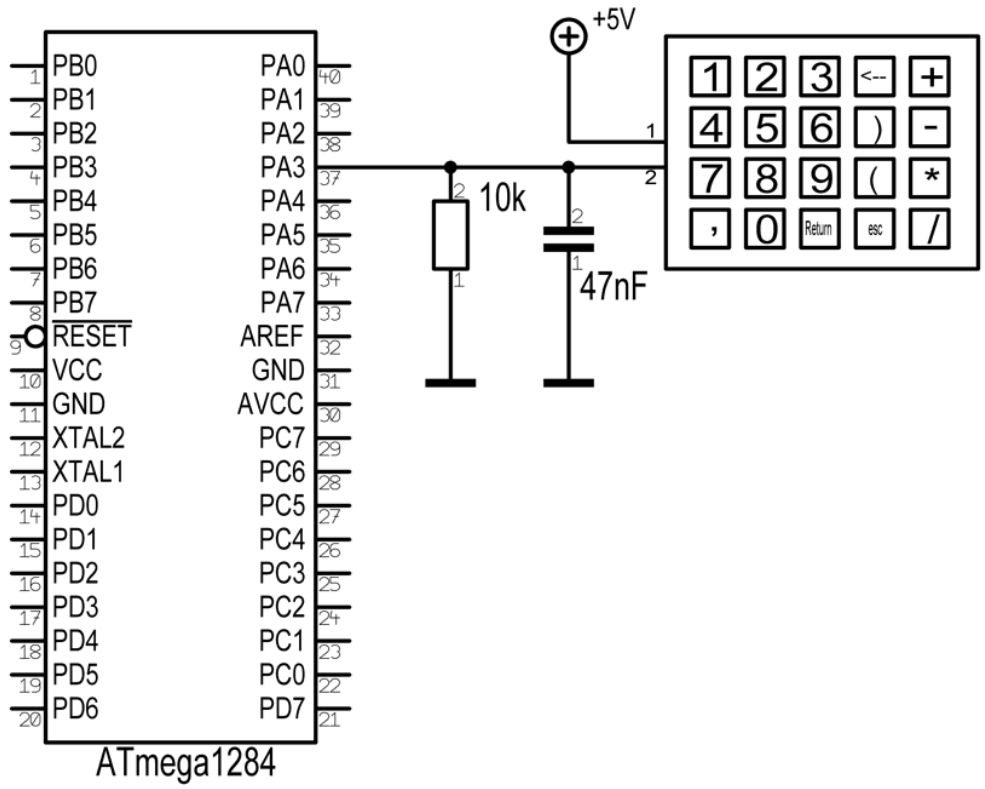


Abb. 14

Der Schaltplan des 20-Tasten-Keypads ist in Abbildung 3 zu sehen. Benötigt man weiniger Tasten, können die nicht benötigten einfach weggelassen werden. Wird das Keypad nicht benötigt, PA3 nach Masse brücken.

PWM-Ausgänge

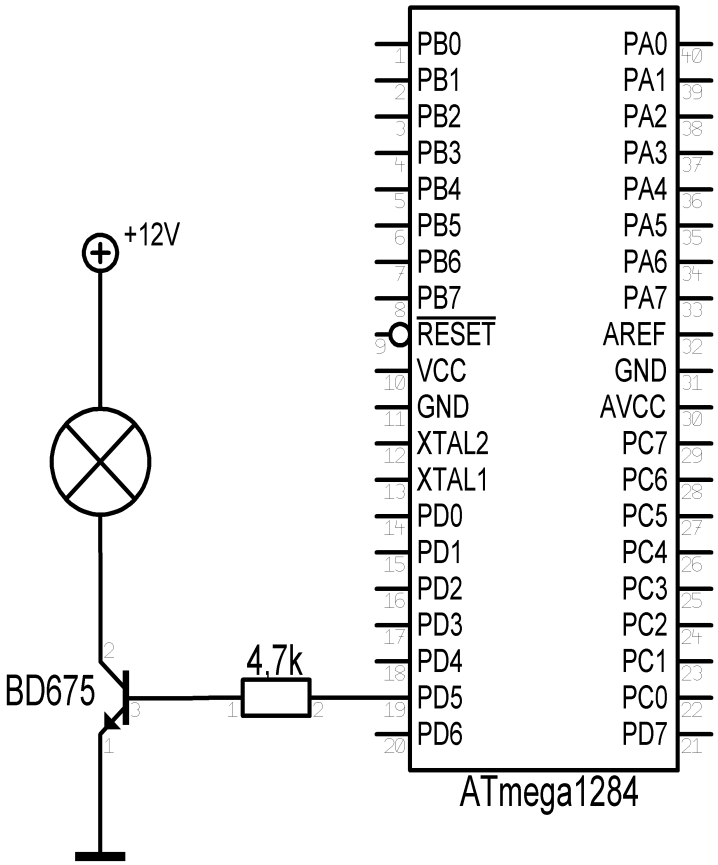


Abb. 15

In diesem Beispiel steuert der PWM-Ausgang die Helligkeit einer 12V Lampe. Der BD675 ist ein Darlingtontransistor mit 4A Belastbarkeit. Darlingtontransistoren sind zwar praktisch in der Anwendung, haben aber den Nachteil, dass die Kollektor-Emitterspannung im durchgeschalteten Zustand ca 0,7V beträgt. Bei höheren Strömen ist daher mit einer gewissen Verlustleistung zu rechnen. Auf dem Controllerboard wird daher ein MOS-FET verwendet.

Verbindung zum PC

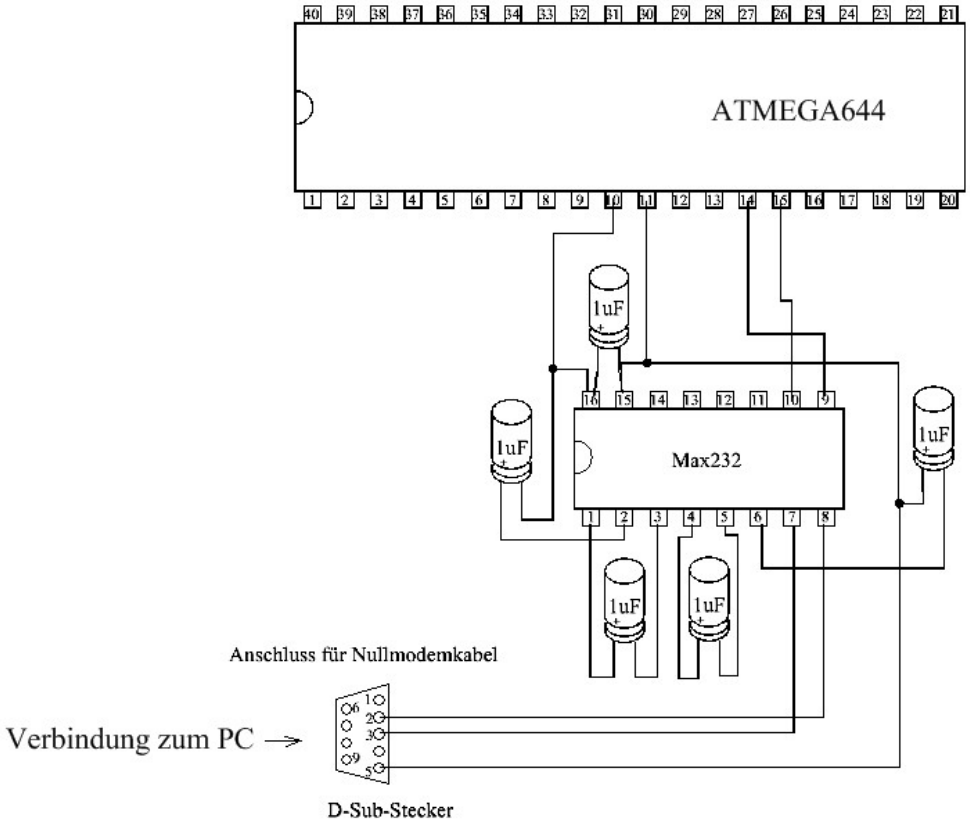


Abb. 16

RS232-Schnittstelle

Die Verbindung zum PC ist simpel. Es wird lediglich ein Schnittstellenwandler benötigt. An den abgebildeten Kondensatoren ist der Pluspol markiert. Es wird ein Nullmodemkabel benötigt. Hat der PC keine RS232-Schnittstelle (V24) mehr, so hilft ein USB-Seriell-Konverter.

Counteingang mit Taster

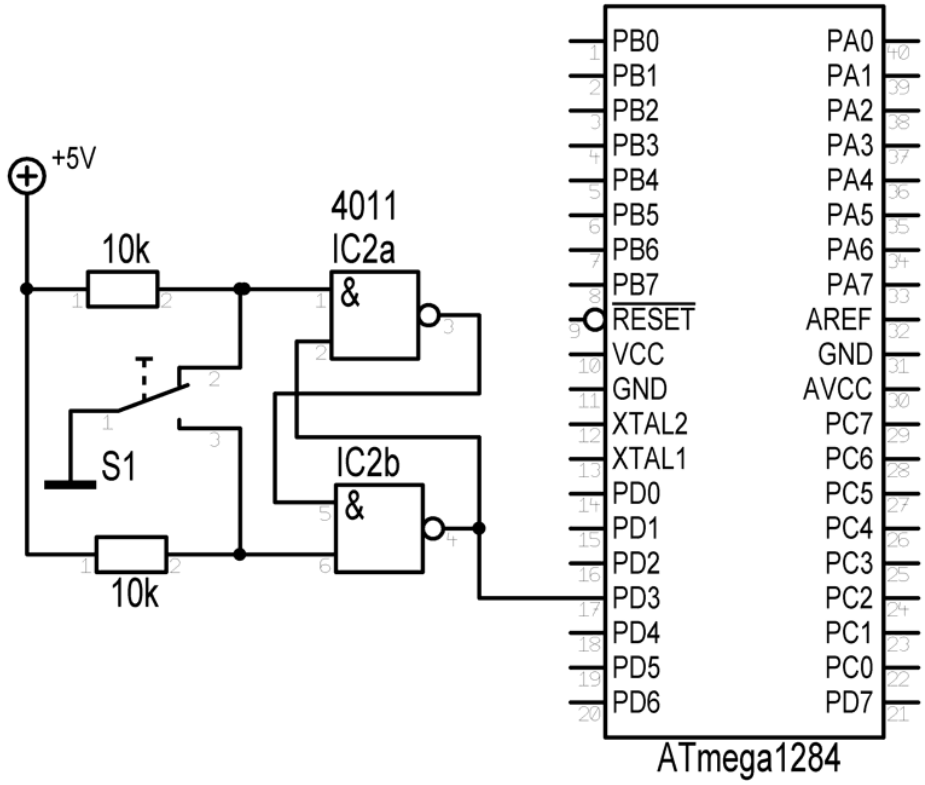


Abb. 17

Hardwareentprellung mit Nand-Gattern für Tasteneingabe

Der Zähleingang ist recht schnell, er kann mehr als 10000 Impulse je Sekunde zählen. Will man Tastendrücke zählen (oder Impulse anderer Kontakte), so ist eine Schaltung, die das mechanische Prellen unterdrückt wie hier in Abb. 17 hilfreich. Sie funktioniert mit Umschaltkontakt. Erzeugt eine elektronische Schaltung die Impulse, so ist eine Entprellung entbehrlich. Eine Beschaltung wie in **Abb. 13** ist dann auch für den Zähleingang richtig. Wird der Zähleingang nicht verwendet, Pin 17 unbedingt mit Masse verbinden. Auf dem Controllerboard findet dieser Eingang keine Verwendung, kann aber nachgerüstet werden.
Werden Zähl- und Frequenzmess-Eingang gleichzeitig mit höheren Frequenzen (> 5KHz) beaufschlagt, kann es zu Messfehlern kommen.