User Manual

Version 02/09 G BA001 ProgRef
Revision 1.35

CE

Programmier Referenz

M 2.0 / STATION 2.0
M ADV / STATION ADV

Control Units

Inhalt

EINFÜHRUNG

M 2.0 / M ADV - BASIC PROGRAMMIERBARE CONTROL UNITS

Kleine Microcontroller haben die Welt erobert und sind überall zu finden. Microcontroller werden vorwiegend in Assembler programmiert, was Erfahrung in dieser Programmiersprache und genaue Kenntnis der Prozessorarchitektur erfordert. Nicht selten ist allein das Manual zum Prozessor mehrere hundert Seiten lang und von Laien praktisch nicht zu verstehen. Auch wer Assembler programmieren möchte, ohne diese Kenntnisse bereits zu haben ist auf Bücher angewiesen deren Seitenumfang ganz sicher noch größer ist BASIC++ ist der BASIC-Dialekt, der zur Programmierung der Control Units verwendet wird und keine der genannten Spezialkenntnisse erfordert. Die Syntax entspricht in etwa der des Standard-BASIC. Um den Kern des Standard-BASIC sind zahlreiche neue Funktionen verfügbar.

Speicher
Die Computer der M 2.0 M ADV Serie haben ein komplettes Betriebssystem auf dem Chip und lassen sich in einem BASIC-Dialekt programmieren. BASIC ist nicht nur leicht zu erlernen, sondern auch sehr platzsparend. Ein BASIC- Program braucht nur etwa ein Fünftel des Speichers, das ein Asseblerprogramm gleicher Funktion benötigen würde. Grundsätzlich gilt, je länger das BASIC - Programm ist, desto grösser ist der Speicher-Vorteil gegenüber einer Assembler Programmierung. Die M 2.0 Serie bietet dem Anwender etwa 10kB Programmspeicher und 140 Byte Speicher für Variablen. Die M ADV Serie bietet neben dem zusätzlichen Floating Point Modul 22kB Programmspeicher und 240Bytes (mit kleinen Einschränkungen) für Variablen.

Geschwindigkeit
Der Ausspruch "BASIC ist langsam" stammt aus der Zeit, als BASIC auf dem Computer als Quelltext gespeichert war und der gesamte Text bei der Ausführung erst gelesen und interpretiert werden musste. Bei BASIC wird der Quelltext von einem Compiler gelesen und in die von Betriebssystem lesbaren sog. Token verwandelt, welche als eigentliches Programm in die Control Units geladen wird. Das Betriebssystem muss also nicht die vielen Buchstaben einer Programmzeile lesen sonder nur das dafür repräsentative Token. Und das geht sehr schnell. Die M 2.0 / M ADV Unit benötigt nur etwa 50us für die Ausführung eines BASIC-Befehls.

ALLGEMEIN ZU BASIC++

Programmieren in BASIC++
Wenn man bedenkt, dass ein einziger BASIC-Befehl Routinen des Betriebssystems aufruft, die meistens mehrere hundert Bytes lang sind, ist leicht einzusehen, dass es eine enorme Zeiterspartnis darstellt ein Programm in BASIC zu programmieren. Ein Programm in BASIC zu programmieren dürfte nur etwa ein Zehntel der Zeit beanspruchen, welches ein Assemblerprogramm gleicher Funktion zum Entwurf benötigt.

Nötige Programmierkenntnisse
Basic++ ist sowohl für den Einstieg (einfache Kernfunktionen) als auch für die professionelle Programmierung der M 2.0 / M ADV Control Units optimiert. Der Einstieg wird durch viele Beispiele sowohl zu BASIC++ als auch zu angebotenem Zubehör erleichtert.
Sie werden sehr schnell erkennen wie einfach die Handhabung ist und wie übersichtlich sich Ihre Programme gestalten lassen. Ausserden finden Sie in der LIBRARY eine Auswahl von Programmen die Sie in Ihren Anwendungen unterstützen
BASIC++ und die Control Units sind also auch für Entwickler und Programmierer, welche die Kenntnisse über Prozessorarchitektur und Assemblerprogrammierung haben, ein leistungsfähiges Tool, komplexe Programme in kürzester Zeit zu entwickeln und eine flexibles Controller-System für die Steuerung in Ihrer Anwendung.

Bitte besuchen Sie www.spiketronics.com um über Updates aktuell informiert zu sein

ALLGEMEIN ZU DIESER REFERENZ

Diese Entwickler Referenz beschreibt die Aspekte der Programmierung in BASIC++ speziell für die Control Units M 2.0 / M ADV und deren Hutschienen - Versionen.
Im Rahmen dieser Referenz werden die für die M 2.0 / M ADV anwendbaren Instruktionen und Definitionen beschrieben, soweit es sinnvoll ist, wird jeweils eine allgemein gültige Syntax angegeben.

Syntax: *Variable = value1 MOD value2*

Variable: Variable Byte oder Word Typ
value1 Variable, value oder Konstante Byte oder Word Typ
value2 Variable, value oder Konstante Byte oder Word Typ

Die Werte (hier *value1* and *value2*) können auch Terme in Klammern sein.

Variable= *(SQR(value*value))* MOD *value2*

Das ist gültig, solange Rechenergebnisse den Zahlenbereich der Variablen nicht überschreiten.

Zusätzlich zur reinen Syntaxbeschreibung werden in dieser Referenz jeweils auch Beispiele gezeigt, bei denen die notwendige Definition der verwendeten Variablen aber aus Platzgründen fehlt.

Definiton:

```
DEFINE MyWord as word
DEFINE MyByte1 as byte
DEFINE  MyByte2 as byte
DEFINE MyBitVar as bit
```

Example:

```
MyWord=MyByte1 MOD MyByte2
```

DATEN TYPEN

Die M 2.0 / M ADV unterstützt Bits, Bytes, Words. Floats werden zusätzlich von der M ADV unterstützt. Bits nehmen die Zustände ON und OFF bzw. TRUE und FALSE ein. Bytes können einen vorzeichenlosen Wert von 0 bis 255 annehmen, Words sind Vorzeichen behaftete Werte im bereich -32768 bis +32767. Die Zuweisung kann dabei in verschiedenen Zahlensystemen erfolgen. Details zu den Floats entnehmen sie bitte dem Kapitel FLOATING POINT MODULE

Beispiele verschiedener Zahlensysteme

01011101b	binary system
123o	oktal system
1AFh	hexadecimal system
1000	decimal system
ON	boolean true (numerical 255 at byte values)
OFF	boolean false(numerical 0 for byte and word values)

DEKLARATIONEN UND DEFINITIONEN

OPTION

Das **OPTION** Schlüsselwort ist ein Compilerpräprozessorbefehl, der dazu dient für unterschiedliche Controller Versionen einen angepassten Befehlscode zu erzeugen.

Generell sollte am Anfang des Programms immer die Zielplattform durch OPTION angegeben werden.

Wenn Sie zum Beispiel mit der MICRO arbeiten sollten Sie mit dem OPTION Befehl die Zielplattform CCMICRO wählen; da die Micro nur mit BYTE Variablen arbeitet muss die systemeigene Rückgabe-Variable vom Typ BYTE sein.

Hier eine Lister der OPTIONS:

- **CC2.0**: 64/140 Byte User Variablen, Pointer-Unterstützung
- **CCMICRO**: 24 Byte User Variablen
- **CCADV**: 240 Byte User Variablen, Gleitkomma-Unterstützung (nur ADVANCED)
- **CCADVNOFLOAT**: 240 Byte User Variablen, keine Gleitkomma-Unterstützung (nur ADVANCED)
- **FLOAT**: 140 Byte User Variablen, Gleitkomma-Unterstützung (nur ADVANCED)

```
OPTION CCADV
```

Beispiel: Zielplattform ADVANCED, 240 Bytes nutzbar

PORTS UND VARIABLEN ALLGEMEIN

Variablen und Ports sind Speicherstellen im Controller die in der Regel gelesen und beschrieben werden können. Während Anwendervariablen frei verwendbar und benennbar sind, sind Ports in ihrer Funktion festgelegt. Sie sind zu Laufzeit veränderbar und deren Inhalt abrufbar. Dazu muss der Speicherplatz oder der Port zuerst vom Anwender mit einem symbolischen Namen versehen werden

```
DEFINE MyBitport1  as PORT[1]
DEFINE MyByte as 1
```

Definition mit symbolischem Namen

Sind Speicherstellen oder Ports einmal definiert werden sie im Programm ausschliesslich mit ihrem symbolischen Namen verwendet um z.B.neue Werte zuzuweisen oder Inhalte zu lesen.

```
MyBitport1 = ON
MyByte1 = MyByte1*10
```

Zuweisungen

PROGRAM MEMORY

Die M 2.0 hat 10KB Programmspeicher, die M ADV etwa 22KB. Speicher der nicht für das Anwenderprogramm genutzt wird, steht für nicht flüchtige Datenspeicherung zur Verfügung.

USER VARIABLEN - COMPILER DEFINIERT

Die M 2.0 Control Units haben 140 Variablen die dem Anwender für sein Programm zur freien Verfügung stehen. Bei der M ADV sind weitere 100 Variablen nutzbar, Wenn die File Funktionen PRINT# und INPUT# nicht benutzt werden, oder so benutzt werden, dass die Änderung des Variableninhalts der Variablen 140 bis 240 nicht stört. Bei BASIC++ hat man bei der Variablen-Definition zwischen lokalen und globalen Varianblen zu unterscheiden die, wie der Name sagt nur innerhalb einer Function oder innerhalb des ganzen Programms Gültigkeit haben. Weiterhin gibt es referenzierte Variablen welche jeweils eine einzige Speicherstelle im definierten Format Word,Byte,Float darstellen, aber unter verschiedenen symbolischen Namen benutzt werden können.

Diese Variablen müssen definiert werden, bevor sie im Programm verwendet werden können. In der Regel überlässt man dem Compiler die Aufteilung der Variablen auf den Speicherbereich. Für spezielle Anwendungen kann aber der Anwender die Variablen auf bestimmte Speicherplätze selbst verteilen.

BIT Variablen
Acht BIT Variablen (Zustand ON oder OFF) belegen 1 Byte im Speicher.

```
define MyBit as bit
```

BYTE Variablen
Bytes (Wertebereich 0 ... 255) ist der kleinste numerische Datentyp 1 Byte = 8 Bit

```
define MyByte as byte
```

WORD Variablen
Words (Wertebereich -32768 ... 32767) belegen 2 Byte = 16 Bit

```
define MyWord as word
```

FLOAT Variablen
Sie belegen 4 Byte

```
define MyFlaot as float
```

Referenzierte Variablen
Um Variablenspeicher zu sparen kann man verschiedene Variablen auf eine einzige Variable referenzieren, d.h einer einzigen Variablen verschiedene Namen geben. Das ist für Bit, Byte,Word und Float Variablen zulässig

```
define MyWord as word
define Word1 Ref Myword
define Word2 Ref Myword
```

Word1 und Word2 benutzen den Variablenspeicher von MyWord

Mit der Referenzierung hat man auch leicht Zugriff auf die einzelnen Bits innerhalb einer Variable

```
define DATA  as byte

define DB0   ref DATA at bit[1]
define DB1   ref DATA at bit[2]
define DB2   ref DATA at bit[3]
define DB3   ref DATA at bit[4]
define DB4   ref DATA at bit[5]
define DB5   ref DATA at bit[6]
define DB6   ref DATA at bit[7]
define DB7   ref DATA at bit[8]
```

Im Programm manupiliert dann z.B.

DB7=OFF

Das höchste Bit der Variablen DATA. Wichtig ist in diesem Zusammenhang dass die Referenzierung auf bit [1] bis bit [8] und nicht auf bit [0] bis bit [8] erfolgt

USER VARIABLEN - USER DEFINIERT

In der Regel überlässt man dem Compiler die Aufteilung der Variablen auf den Speicherbereich. Für spezielle Anwendungen kann aber der Anwender die Variablen auf bestimmte Speicherplätze selbst verteilen. Der Anwender hat selbst Sorge zu tragen dass gemeinsame Speicherzellen nicht ungewollt verändert werden, d.h BIT[8] teilt sich z.B.eine Speicherzelle mit BYTE[1] und WORD[1]

In der Regel ist das nur erforderlich wenn Systemtreiber geladen werden welche bestimmte Variablen als Schnittstelle zum BASIC Teil des Programms benötigen. Dann werden sie Speicherzellen für den Treiber vom Anwender dafür reserviert und der verbleibende Rest dem Compiler zu Vergabe überlassen.

{BYTEOFFSET}

Der Präprozessor Befehl {BYTEOFFSET} legt den Startpunkt zur automatischen Vergabe des Index für Byte und Word Variablen fest. Bei Byte und Word Variablen wird der Index immer inkrementiert.

Beispiel für die Vergabe von Variablen Für einen Systemtreiber und Compileranweisung zur Variablen-reservierung. Byte [1] und Byte [2] sind hier für einen Treiber reserviert

```
define MyByte1   as Byte[1]
define MyByte2   as Byte[2]
{BYTEOFFSET 3}
define MyByte3 as Byte
```

Der Compiler beginnt hier mit der Vergabe der Variablen ab dem dritten Byte. MyByte3 wäre also Byte[3]

BIT Variablen
Die Speicherplätze können auf BIT Variablen verteilt werden. Da aber maximal 256 BIT Variablen zulässig sind, kann nicht der ganze Variablenspeicher an BIT Variablen vergeben werden. Die höchste BIT Variable ist BIT[256] und liegt im Speicher im höchsten Bit des Memory Bytes 32

Acht BIT Variablen (Zustand ON oder OFF) belegen 1 Byte im Speicher.

```
define MyBit as bit[1]
```

zulässig ist BIT[1] bis BIT[256]

BYTE Variablen
Bytes (Wertebereich 0 ... 255) ist der kleinste numerische Datentyp 1 Byte = 8 Bit

```
define MyByte as byte[1]
```

zulässig ist BYTE[1] bis BYTE[140] (240 bei der M ADV)

WORD Variablen
Words (Wertebereich -32768 ... 32767) belegen 2 Byte = 16 Bit

```
define MyWord as word[1]
```

zulässig ist WORD[1] bis WORD[70] (120 bei der M ADV)

EXTERNE USER VARIABLEN

Mit den Control Units kann auch auch auf ein externes I²C-Bus EEPROM direkt zugegriffen werden. Das EEPROM ist als ausgelagerter Speicher zu verstehen und wird in BASIC++ mit CHIPRAM deklariert. Anders als bei beim internen Variablen-Speicher werden FLOAT, WORD und BIT-Variablen nicht unterstützt. Das externe EEPROM muss immer auf die I²C-Bus Adresse 160 eingestellt werden. Es wird jeweis ein Byte geschrieben oder gelesen.

Define MyByte as CHIPRAM	Benutzer Variable externer Speicher, vom Compiler verwaltet
Define MyByte as CHIPRAM[1]	Benutzer Variable externer Speicher, vom Anwender verwaltet

KONSTANTEN

Mit Konstanten werden Zahlen, implementierte Konstanten und durch den Benutzer erstellte Konstanten bezeichnet. Konstanten (außer in BASIC++ schon vorimplementierte Konstanten) können nur während der Entwicklungsphase verändert werden. Während der Laufzeit ist der Wert im Gegensatz zu Eigenschaften fest definiert.

Numerische Konstanten werden in BASIC++ in der Regel im Dezimalsystem angegeben. Zusätzlich unterstützt BASIC++ auch das Binär- , Oktal- , und Hexadezimalsystem. Um eine solche Zahl zu markieren muss die Zahl entweder durch Endung "b", "o" oder "h" hervorgehoben werden

```
const MyConstant = 122
```

POINTER

INTERRUPT VECTOR

Die M 2.0 / M ADV Units haben einen INTERRUPT-Vektor der auf einen Programmteil zeigt, dessen Aufgabe es ist, unmittelbar auf eine externe Anforderung (eben das Interrupt Signal am Port IRQ) zu reagieren.Der IRQ Eingang ist auf fallende Flanke getriggert. Der Interrupt ist abgeschaltet, wenn kein Interrupt Vektor definiert wurde.

```
INTERRUPT MyInterruptService
```

Wahlweise zum Interrupt IRQ, kann der Interrupt auch durch den internen 20 ms Systemtimer ausgelöst werden, oder auch wenn vom IR-Objekt oder vom RF-Objekt gültige Daten empfangen wurden. Die Definition des INTERRUPT Vektors ändert sich dabei nicht, Es wird nur ein entsprechendes Bit im CONFIG Register gesetzt. Im Kapitel CONFIG REGISTER finden Sie weitere Informationen zu diesem Thema.

ACHTUNG:
Als Anwender müssen Sie dafür sorgen, dass Ihre INTERRUPT-SERVICE ROUTINE kein OBJECT aktiviert, öffnet ohne ein anderes aktives Objekt vorher deaktiviert zu haben.
Ausserdem müssen Sie sich bewusst sein, dass ein Interrupt zu jedem x-beliebigen Zeitpunkt (- beliebige Stelle im Programm) auftreten kann. Interrupts können sehr vorteilhaft sein, müssen aber mit viel Sorgfalt bei der Programmierung des Anwenderprogrammes eingesetzt werden

VARIABLEN POINTER

Die Control Units erlauben dass auf Variablen mittels Pointer (Zeiger) zugegriffen werden kann
Pointer sind eine spezielle Art von Variablen. Alle Pointervariablen sind ein Byte groß. Mit Ihnen kann genauso umgegangen werden wie mit normalen Bytevariablen.

Mit einer Pointervariable kann variabel auf den Speicher zugreifen werden. Über bestimmte Operatoren kann man mittels einer solchen Variable den Inhalt einer Speicherstelle, auf die der Pointer zeigt auslesen und verändern. Durch Pointer kann man Strukturen wie Arrays und Zeichenketten programmieren.

```
DEFINE ^p AS WORD
```

Definiert eine Variable welcher der Inhalt einer anderen Variablen zugewiesen wird. Diese andere Variable wird mit einem Pointer adressiert

EXTERNE DATEIEN

IMPORT

Es ist möglich externe Dateien in Form von Tabellen oder Programmen zu importieren. BASIC++ stellt für den Anwender eine ganze Reihe von Bibliotheksdateien zur Verfügung, die es dem Anwender ersparen allgemein gebräuchliche Funktionen (wie z.B. Tastatureingabe) selbst zu programmieren. Darüber hinaus machen sie ein Programm wesentlich besser lesbar. Es kann jedes beliebige File importiert werden, allerdings muss es natürlich vom Compiler lesbar sein.

Syntax: Import [File]

```
IMPORT "lib\MYFILE.BAS"
```

SYSCODE

Die Control Units sind in der Lage Teiber oder auch Erweiterungen des Betriebssystems zu laden. Dafür stehen zwei Seiten (je 256 Bytes) am Ende des Programmspeichers zur Verfügung. Die Files haben immer die Extension
.S19

Syntax: SYSCODE [File]

```
SYSCODE "lib\MYFILE.S19"
```

SPECIAL PORTS

Special digital ports
Alle Speziellen Ports sind mit 10k Pull Up Widerständen nach HI gezogen. Spezielle Ports sind hinsichtlich der Funktion in der Regel einmalig und brauchen nicht definiert werden

START und RESET

Diese Ports sind in der Regel an Taster angeschlossen um das Anwenderprogramm manuell zu starten oder anzuhalten. Drücken sie die an START angeschlossene Taste um ein Anwenderprogramm zu starten. RESET hält das Programm an und aktiviert den DOWNLOAD MODE.

RESET=DOWNLOAD-MODUS !!!
Es dürfen dann *keinerlei* Daten von der seriellen Schnittstelle kommen, da ansonsten das Anwender-programm überschrieben wird. Der Jumper JP2 auf der Unit aktiviert den Autostart, d.h. die Unit started nach Anlegen der Betriebsspannung das Anwenderprogramm. Ist dieser Jumper nicht gesteckt, verweilt die Unit im RESET und wartet auf einen Download, oder das Startsignal.

START und RESET verhält sich bei den RAIL Versionen geringfügig anders. Die RAIL Versionen starten das Programm nach einem Reset oder dem Anlegen der Betriebsspannung automatisch (Betriebsart Autostart). Für einen Programm-Download in die RAIL Versionen muss zuerst die RESET dann zusätzlich die PROG Taste gedrückt werden. Lassen sie dann die RESET und anschließend die PROG Taste los, starten Sie den Download Ihres Anwenderprogramms auf Ihrem Computer. Nach beendetem Download startet das Programm automatisch. Sehen Sie dazu bitte auch in das Manual zu den RAIL Versionen.
Der START Port kann bei allen M 2.0 / M ADV Units gelesen werden. Nach Programmstart steht damit also ein zusätzlicher Porteingang (oder eine Taste bei den RAIL Versionen) zur Verfügung. Der Port Status ist in CONFIG REGISTER gespiegelt. Sehen sie hierzu bitte das zugehörige Kapitel.

START und RESET benötigen keine Definition, es sind keine Schlüsselwörter dafür reserviert.

FREQ1 und FREQ2

FREQ1 und FREQ2 sind immer Eingänge

Frequenzzählung
FREQ1 und FREQ2 sind Systemvariablen die nicht separat angelegt werden müssen. Das Betriebssystem misst an beiden Eingängen die Frequenz (mit einer TORZEIT von 1s) und stellt den ermittelten Wert in diesen beiden Variablen zu Verfügung. FREQ1 und FREQ2 haben eine alternative Funktion als Ereigniszähler.

Ereigniszählung
Im CONFIG-REGISTER bestimmen Bit 1 und Bit 2 die Funktion von FREQ1 u. FREQ2. Beide Eingänge können getrennt konfiguriert werden, d.h. während einer der Eingänge ein Ereigniszähler ist kann der andere auch ein Frequenzzähler sein. Die Zähler können gelöscht werden, wenn den Systemvariablen jeweils der Wert 0 zugewiesen wird. Ausserdem können Sie den Zählern beliebige Anfangswerte zuweisen, z.B. einen Zählerstand vom Vortag Der Ereigniszähler kann einen max. Wert von 32767 annehmen und zählt dann nicht mehr weiter.

Synchronisierung der RTC
FREQ1 ist ausserdem dafür vorgesehen ein DCF77 Empfangsmodul anzuschliesen um die Systemuhr (RTC) automatisch zu stellen. Dieser Programmteil des Betriebssystems läuft im Hintergrund, der Anwender braucht keine Einstellungen vorzunehmen. Die RTC ist synchronisiert, sobald ein Empfangsmodul angeschlossen ist und gültige Daten an FREQ1 decodiert wurden.

FREQ1 und FREQ2 sind vordefiniert, der Trigger ist auf fallende Flanke eingestellt. Diese Ports sind immer Eingang.FREQ1 und FREQ2 sind Schlüsselwörter.

IRQ

Der IRQ Port ist immer ein Eingang, er ist negativ Flanken getriggert. Ein Signal am IRQ Port veranlasst das Programm die aktuelle Abarbeitung zu unterbrechen und auf ein (mittels dem INTERRUPT Vector definiertes) Sprungziel zu verzweigen.Der IRQ Port kann bei allen M 2.0 / M ADV Units gelesen werden. Nach Programmstart steht damit also ein zusätzlicher Porteingang zur Verfügung. Der Port Status ist in CONFIG REGISTER gespiegelt. Sehen sie hierzu bitte das zugehörige Kapitel.

Der IRQ selbst bedarf keiner Definition, wohl aber muss ein POINTER, eben der Interrupt Vektor definiert werden. Es kein Schlüsselwort dafür reserviert.

BEEP

Der BEEP Port ist immer ein Ausgang und in der Regel an einen Piezo Buzzer angeschlossen. Der Audio Frequenzbereich beträgt 10kHz bis 100Hz.

Der BEEP Port ist vordefiniert, Beep ist ein Schlüsselwort

RXD/TXD

Das serielle interface (RXD=input TXD=output) verarbeitet 5V digitalen Logikpegel.Es ist immer ein Pegelwandler erforderlich, wenn diese Ports an ein RS232 Interface angeschlossen werden. Eine Kommunikation zwischen Units ist auf 5V Ebene natürlich ohne zusätzliche Hardware möglich.

RXD und TXD bedürfen keiner Definition. RXD ist ein Schlüsselwort

STANDARD DIGITAL I/O PORTS - P1 bis P16

Die standard digital Input/Output - Funktion steht dem Anwender an allen 16 Ports der M 2.0 / M ADV zur Verfügung Nach Anlegen der Betriebsspannung (oder Reset) sind alle Ports im Modus Digitalport - Eingang. Jeder Digitalport wird zu Anfang des Programms mit einem Symbolischen Namen versehen. Sehen Sie im Kapitel INSTRUKTIONEN UND SCHLÜSSELWÖRTER wie ein Digitalport gesetzt, gelesen oder deaktiviert wird. Die Analogports AD1 bis AD8 können unabhängig voneinander als Digitalport betrieben werden. Siehe Kapitel CONFIG REGISTER. Sie sind dann PORT[17] bis PORT[24] oder BYTEPORT[3] BYTEPORT[3] ist als default ein EXTENDED PORT. Er ist abgeschaltet, wenn die AD-Ports als Digitalports betrieben werden

define MyBitPort1	**as PORT[1]**
define MyByteport1	**as BYTEPORT[1]**

Zulässig ist die Definition für PORT[1] bis PORT[16] und PORT[17] bis PORT[24] wenn die AD-Ports digital betrieben werden.
Zulässig ist die Definition für BYTEPORT[1] bis BYTEPORT[3] und BYTEPORT[3] wenn die AD-Ports digital betrieben werden.

DIGITAL INPUT PORT

Digitaleingänge werden zur Abfrage von Schaltzuständen verwendet. Wird ein Digitalport als Eingang benutzt, führt er im unbeschalteten Zustand einen nicht definierten Pegel, der vom Anwender mittels eines Pullup Widerstandes festgelegt werden sollte. Ist beispielsweise ein Reedkontakt an diesem Port angeschlossen, wird dann bei offenem Schalter eine logische Eins (TRUE) vom Port gelesen, bei geschlossenem Schalter eine logische Null (FALSE). Achten Sie bitte unbedingt darauf, dass je nach Beschaltung des Ports und der logischen Aussage, die Ihr Programm beinhalten soll, der eingelesene Wert eventuell invertiert werden muss (NOT- Operator).
Nach dem Zuschalten der Betriebsspannung oder nach einem Reset verhalten sich alle Digitalports zunächst elektrisch als Eingang, sie führen also über den Pullup- Widerstand High- Pegel oder undefinierten Pegel, wenn nichts angeschlossen ist.

DIGITAL OUTPUT PORT

Wird ein Digitalport als Ausgang verwendet, können daran nachfolgende ICs, Transistoren oder Low-Current- Leuchtdioden über Widerstände betrieben werden. Der maximal zulässige Laststrom für einen einzelnen Port beträgt 10 mA. In jedem Fall ist eine ausreichende Strombegrenzung, zum Beispiel durch einen Widerstand, zu gewährleisten, da es sonst zur Zerstörung des Mikrocontrollers kommen kann! Innerhalb des Mikrocontrollers erfolgt die interne Beschaltung eines Digitalports als Ausgang oder Eingang beim Ausführen des Anwenderprogramms. Eine Schreibanweisung auf einen Port veranlasst diesen die Funktion eines Ausgangs anzunehmen.

OPTIONAL DIGITAL INTERFACE PORTS

Das Betriebssystem unterstützt einige spezielle Funktionen die bestimmte Ports in Anspruch nehmen. Diese Ports stehen dann nicht mehr zur freien Verfügung, wenn deren spezielle Funktion genutzt werden soll. Spezielle Deklarationen brauchen nicht gemacht zu werden, die Ports übernehmem diese Funktionen automatisch sobald die entsprechenden Objekte bzw. Schlüsselwörter benutzt werden.

LCD INTERFACE (PORT 9 bis 16)

Wird das LCD Interface mit dem Schlüsselwort LCD.INIT initialisiert, werden die Ports 9 bis 16 vom LCD Treiber des Betriebssystems verwaltet und ermöglichen ein konfortables Schreiben auf das LCD. Anschliessbar sind alle LCD welche einen HD 44100 kompatiblen LCD Controller verwenden.
Sie können diese Ports auch benutzen, wenn ein LCD angeschlossen ist und nicht benutzt wird. Dazu muss jedoch der Port 15 fest auf LO geschaltet werden. Port 15 darf dann natürlich nicht benutzt werden.

I²C-BUS INTERFACE (PORT 9 and 10)

Jede Benutzung der I²C Bus Funktionen (Kapitel I²C OBJECT) belegt diese ports als SDA und SCL für den I²C Bus. Ausserdem wird der I²C Bus auch bei einigen Systemfunktionen wie CHIPRAM und CHIPCARD BOOT aktiviert. Der I²C Bus teilst sich den Port 9 und 10 mit dem LCD Interface, was aber zu keinerlei Beeinfussungen führt.

RC5 IR INTERFACE (PORT 2 und 3)

Die Benutzung des RC5 IR Interfaces (Kapitel IR OBJECT) belegt diese Ports als RC Coder/Decoder Interface. Diese Ports stehen n nicht mehr zur freien Verfügung.

RF INTERFACE (PORT 2 und 3)

Die Benutzung des RF Interfaces (Kapitel RF OBJECT) belegt diese Ports als RF Coder/Decoder Interface. Diese Ports stehen dann nicht mehr zur freien Verfügung.

EXTENDED PORTS

Die EXTENDED PORTS sind eigentlich nicht direkter Bestandteil der Computer Ressourcen, da sie ja externe I²C Bus Bausteine sind. Allerdings werden sie vom Betriebssystem so behandelt als wären sie Bestandteil der Unit und müssen dann auch als solche definiert werden.

```
define MyBitPort17          as PORT[17]
define MyBytePort4      as BYTEPORT[4]
```

Zulässig ist Port[17] to Port [144]
Zulässig ist Byteport[3] to Byteport[18]

Der PCF 8574 hat als Teilnehmer am I²C.Bus eine bestimmte Adresse die aus einem festen (vom Hersteller) Teil und einem einstellbaren Teil besteht. Der feste Teil bildet die BASIS-Adresse, d.h. Ab dieser Basis kann eine von einer bestimmte Anzahl von Adressen vergeben werden.

PCF 8574 0100$A_2A_1A_0$R BASISADRESSE 64
PCF 8574A 0101$A_2A_1A_0$R BASISADRESSE 80

$A_2A_1A_0$ stellt den vom Anwender einzustellenden variablen Teil der Adresse ein. Bei den Conrad Erweiterungsmodulen sind diese Adresseingänge mit Widerständen gegen +5V gezogen und bilden die Adresse 111 wenn kein Jumper gesteckt ist. Ein Jumper zieht den entsprechenden Adresseingang auf LO. Ein Jumper auf A_0 stellt also eine Adresse 110 (also dezimal 6) ein.

R ist das niedrigste Bit und bestimmt, ob der PCF8574 eine Schreiboperation (R=0), oder eine Lese-operation (R=1) durchführen soll.
Die tatschliche Adresse am Bus ist dann BASISADRESSE + ADRESSE * 2 + R In diesem Beispiel ergibt sich also die Adresse 76 (WRITE) und 77 (READ)

```
0  1  0  0  x  x  x  1
|--------------|------------|----|
FIX ADR       ADR    R/W
```

READ- Operation, LSB HI

```
0  1  0  0  x  x  x  0
|--------------|------------|----|
FIX ADR       ADR    R/W
```

WRITE- Operation, LSB LO

Tatsächlich aber braucht Sie das nicht zu kümmern, da die Adressierung vom Betriebssystem übernommen wird. Sie müssen nur wissen welche Ports zu einer bestimmten Adressierung gehören:

PCF 8574	**ADR 0**	**Ports 17 - 24**	**BYTEPORT 3**
PCF 8574	**ADR 1**	**Ports 25 - 32**	**BYTEPORT 4**
PCF 8574	**ADR 2**	**Ports 33 - 40**	**BYTEPORT 5**
PCF 8574	**ADR 3**	**Ports 41 - 48**	**BYTEPORT 6**
PCF 8574	**ADR 4**	**Ports 49 - 56**	**BYTEPORT 7**
PCF 8574	**ADR 5**	**Ports 57 - 64**	**BYTEPORT 8**
PCF 8574	**ADR 6**	**Ports 65 - 72**	**BYTEPORT 9**
PCF 8574	**ADR 7**	**Ports 73 - 80**	**BYTEPORT 10**
PCF 8574A	**ADR 0**	**Ports 81 - 88**	**BYTEPORT 11**
PCF 8574A	**ADR 1**	**Ports 89 - 96**	**BYTEPORT 12**
PCF 8574A	**ADR 2**	**Ports 97 - 104**	**BYTEPORT 13**
PCF 8574A	**ADR 3**	**Ports 105 - 112**	**BYTEPORT 14**
PCF 8574A	**ADR 4**	**Ports 113 - 120**	**BYTEPORT 15**
PCF 8574A	**ADR 5**	**Ports 121 - 128**	**BYTEPORT 16**
PCF 8574A	**ADR 6**	**Ports 129 - 136**	**BYTEPORT 17**
PCF 8574A	**ADR 7**	**Ports 137 - 144**	**BYTEPORT 18**

Allgemein und damit auch bei Verwendung der Externen Ports muss das LCD unbedingt initialisiert werden, da es sich leitungen mit dem I²C Bus teilt :
Die EXTERN PORTS 17-24 auf Adresse 0 (also Byteport 3) ist abgeschaltet wenn die AD-Ports als digitale Ports verwendet werden.

Die Hutschienen Versionen der M 2.0 / M ADV Units verwenden Byteport 4 zum Schalten der Relais und LEDs. Diese Ports stehen hier also nicht mehr als EXTERNE PORTS zur Verfügung

ANALOGPORTS

Die M 2.0 / M ADV Control Units haben acht A/D ports und zwei D/A-ports annehmen.Die maximale Eingangsspannung an den AD Eingängen ist durch die Betriebsspannung bestimmt welche auch gleichzeitig als Referenzspannung dient. Die Maximale Ausgangsspannung der DA Ausgänge entspricht der Betriebsspannung.
Sehen Sie im Kapitel INSTRUKTIONEN UND SCHLÜSSELWÖRTER wie ein Analogport gelesen oder geschrieben wird.
Die AD-Ports können alternativ auch als reine digitale Ports (auch einzeln) benutzt werden. Die beiden DA-Ports können auch als vollwertige Steuerausgänge für Servos verwendet werden. (siehe Kapitel CONFIG REGISTER)

Referenz-Spannung:
Der angelegte Spannungswert der Betriebsspannung wird dem internen Referenzspannungseingang zugeführt. Diese Spannung gilt als Obergrenze des Messbereiches der A/D-Wandlung und entspricht dem Wandlungswert 255 (FF hexadezimal). Meistens kann die Betriebsspannung aus einem standard Spannungsregler direkt benutzt werden. Bei größeren Ansprüchen an Genauigkeit und Stabilität muss die Betriebsspannung mit einer entsprechenden Präzision erzeugt werden, oder eine entsprechend genau Spannung dem Referenzspannungseingang zugeführt werden, der zugehörige Jumper auf der Unit muss dann offen sein. Als Referenz für das untere Ende des Messbereiches der A/D-Wandlung dient stets das Groundpotenzial (Masse, -, GND, Minus) der Betriebsspannung.

A/D CONVERTER PORTS

An den A/D-Ports können Sensoren aller Art angeschlossen werden, die eine Ausgangsspannung von 0 bis 5 Volt liefern. In den meisten Fällen werden hier aktive Sensoren zur Anwendung kommen, um das Signal des eigentlichen Sensorelementes zu verstärken und den Ansprüchen an Auflösung, Linearität und Driftverhalten zu genügen. Die AD- Eingänge haben eine Auflösung von 8 Bit, was einem Spannungswert von 19,6 mV pro Digit entspricht, wenn die Referenzspannung genau 5,0V ist. Schützen sie die Analogeingänge mit einem Serienwiderstand (10k) wenn Sie nicht sicherstellen können, dass die Spannung niemals größer als 5V ist. Der Widerstand beeinflusst die Genauigkeit der Messung absolut unwesentlich.

Zulässig ist die Definition für AD[1] bis AD[8]

```
define MyAnalogIn        as AD[1]
```

D/A CONVERTER PORTS

Die zwei 8-Bit-D/A-Wandler arbeiten nach dem Prinzip der Pulsweitenmodulation. In einem Zeitabschnitt (Modulationsintervall), der aus 256 Teilabschnitten besteht, wird ein D/A-Ausgang für die Dauer von so vielen Teilabschnitten high- gepulst, wie es dem 8-Bit-Wert entspricht, der zur Ausgabe bestimmt ist. Die Dauer eines Teilabschnittes beträgt 2µs, die des gesamten Modulationsintervalls etwa 52us
Die max. Ausgangsspannung der DA-Ports ist immer so groß (und ebenso genau) wie die 5V Betriebsspannung. Absoluter Fehler +/- 1 Digit (= 1/256 vom Messbereichsendwert) zzgl. Fehler der Betriebsspannung.
Zur Demodulation, also Wandlung in ein echtes Analogsignal genügt meist ein einfaches RC- Glied. Beachten Sie dabei jedoch die Restwelligkeit und den erzielbaren Maximalwert des Ausgangssignals. Beides ist abhängig von der Last, die nach dem RC- Glied folgt.
Beachten Sie bitte, dass der DA-Ausgang auch bei einem Wert von Null immer eine Restspannung am RC-Glied erzeugt. Das liegt zum einen daran, dass der Port im LO-Zustand etwa 50mV Spannung hat, und zum anderen daran, das auch bei einem DA-Wert von Null ein 2us langer Nadelimpuls erzeugt wird. Die Ausgangsspannung nach dem RC- Filter wird daher in diesem Fall etwa 70mV betragen.
Für den Betrieb von LED´s oder Lämpchen am DA-Port (zur Einstellung der Helligkeit) benötigen Sie kein RC- Filter, da das Auge zu träge ist dem schnellen hell/dunkel- Wechsel zu folgen.
Alternativ können die DA-Ausgänge für die Ansteuerung zweier Servos benutzt werden (siehe Kapitel CONFIG REGISTER). Der Ausgang DA1 kann zusätzlich zur Ausgabe von Sprache und Sound benutzt werden. (siehe Kapitel CONFIG REGISTER und SOUND MODUL)

Zulässig ist die Definition für DA[1] und DA[2]

```
define MyAnalogOut       as DA[1]
```

SYSTEM PROPERTIES

System Properties - System Eigenschaften oder auch System Variablen - sind spezielle Ressourcen die mit festen, reservierten Bezeichnern versehen sind, die nicht deklariert werden müssen.

BEEP

Die BEEP Anweisung erzeugt ein 5V Rechtecksignal am Port BEEP. Es ist üblicher Weise an einen Piezosummer angeschlossen.Eine Tonausgabe ist hilfreich zur Anzeige oder Bestätigung von Ereignissen. z.B. wenn eine Taste gedrückt wurde oder ein Alarmzustand besteht.
Der Frequenzbereich erstreckt sich von 100Hz bis ca. 10kHz. Der Wert für Tone nimmt dabei Werte von 100 bis 1 an. Werte kleiner als 2 sollten allerdings nicht benutzt werden, da durch die häufigen System-Interrupts die Systemstabilität nicht immer gewährleistet ist. Werte grösser als 100 verändern die Tonfrequenz unwesentlich.

Syntax: *Beep Ton, dauer, pause*

```
BEEP 10,100,20
```

Ton
Tonhöhe, Wertebereich 2 bis 100

Dauer
Dauer des Tons in Einheiten von 20ms

Pause
Pause nach dem Ton.

RTC - ECHTZEITUHR

Die Echtzeituhr, Real Time Clock (RTC) beinhaltet die Uhrzeit. Sie kann manuell durch Beschreiben der zugehörigen Variablen gestellt werden, oder auch automatich mit der gesetzlich gültigen Zeit synchronisiert werden, wenn ein entsprechended Empfangsmodul angeschlossen und die Systemerweiterung dafür geladen wurde. Alle Variablen sind Byte Werte.

Für die RTC sind die Variablen YEAR,MONTH,DAY,DOW,HOUR, MINUTE, SECOND reserviert. Sie sind lesbar und beschreibbar.
DOW ist der Wochentag, er nimmt Werte von 1 (Montag) bis 7 (Sonntag) an. Alle Variablen sind nach Programmstart mit NULL initialisiert.

Beispiel: Abfrage der Sekunden.

```
IF SECOND = 10 THEN GOTO X
```

Beispiel: Stellen der Sekunden.

```
SECOND = 0
```

TIMER

Der Timer ist eine Systemvariable und wird alle 20ms inkrementiert. Er lauft bis zu einem Zählerstand von 32768 und hält dann an. Der Timer kann gelesen und beschrieben (z.B. gelöscht) werden. Timer ist eine Word Variable.

Beispiel:Lesen des Timer

```
IF TIMER = 10000 THEN GOTO X
MyWord=TIMER/100
```

Beispiel: Schreiben des Timer

```
TIMER = 0
```

FREQ1 and FREQ2

Diese Word-Variablen beinhalten als Primärfunktion die beiden Frequenzzähler, welche digitale Impulse and den Ports FREQ 1 und FREQ2 zählen. Am Port FREQ1 wird auch der DCF77 Empfänger zur Synchronisation der RTC angeschlossen. FREQ1 und FREQ2 haben eine alternative Funktion als Ereigniszähler.
Ist der Zähler als Ereigniszähler konfiguriert, beeinhalten diese Variablen den aktuellen Zählerstand von Ereignissen. Im CONFIG-REGISTER bestimmen Bit 1 und Bit 2 die Funktion von FREQ1 u. FREQ2 Beide Eingänge können getrennt konfiguriert werden, d.h. während einer der Eingänge ein Ereigniszähler ist kann der andere auch ein Frequenzzähler sein. Die Zähler können gelöscht werden, wenn den Systemvariablen jeweils der Wert 0 zugewiesen wird. Ausserdem können Sie den Zählern beliebige Anfangswerte zuweisen, z.B. einem Zählerstand vom Vortag Der Ereigniszähler kann einen max. Wert von 32767 annehmen und zählt dann nicht mehr weiter.

FREQUENCY COUNTING

Im Betrieb Frequenzzähler ist FREQ1 und FREQ1 eine read only Variable, sie kann nicht beschrieben werden. Die Variablen enthalten die während einer Sekunde and den Ports FREQ1/FREQ2 gezählten Ereigniss, also die Frequenz an diesen Ports in der Einheit Hertz. Der Frequenzbereich geht bis etwa 32kHz, mit einer Auflösung von 1Hz. Für sehr kleine Frequenzen bietet es sich an die Ereignisse an diesen Ports über einen grösseren Zeitraum, in der Betriebsart Ereigniszählung, zu messn.

Beispiel: Abfrage des Zählerstands

```
IF FREQ1 = 10000 THEN GOTO X
MyWord=FREQ/100
```

EVENT COUNTING

In der Betriebsart Ereigniszählung ist FREQ1/FREQ2 schreib und lesbar. Sehen Sie im Kapitel CONFIG REGISTER wie diese Betriebsart eingestellt wird.

Beispiel: Abfrage des Zählerstands

```
IF FREQ1 = 10000 THEN GOTO X
MyWord=FREQ1/100
```

Beispiel: Löschen des Zählers

```
FREQ1 = 0
```

INSTRUKTIONEN UND SCHLÜSSELWÖRTER

INPUT/OUTPUT

Einige der verfügbaren Instruktione für Daten Input/Output werden von der seriellen Schnittstelle (default) zu anderen Objecten umgeleitet (LCD,CONFIG,IIC,RF und IR. Details finden Sie im Kapitel OBJECTS. Für Ein/Ausgabe Operationen von Float Variablen der M ADV sind im Kapitel FOATING POINT MODULE beschrieben.

BAUD

Die BAUD Anweisung ändert die Übertragungsgeschwindigkeit der seriellen Schnittstelle zur Laufzeit des Programms, wobei als Parameter entweder eine Zahl oder eine Konstante erwartet wird. Die Übergabe von Variablen ist nicht möglich. Der Übergabeparameter (sofern dieser nicht durch eine entsprechende Konstante symbolisiert wird) entspricht nicht der tatsächlichen Baud Rate, sondern symbolisierte diese nur. Je kleine der Parameter desto höher ist die Übertragungsgeschwindigkeit.

Standardmäßig läuft der Controller mit 9600 Baud.

Syntax: Baud Rate

```
BAUD R1200
```

Rate: Erforderlich, Konstante oder numerischer Wert. Als vordefinierte Übertragungskonstanten können R1200,R2400, R4800, R9600, R19200*, R38400* gewählt werden. (* = nicht die ADV Versionen)

RXD

Die RXD Instruktion liefer auf die Anfrage ob Zeichen im Puffer der seriellen Schnittstelle stehen den Wert TRUE oder FALSE zurück.Der Puffer ist 16 Bytes lang und muss ensprechend oft geleert werden, wenn keine Zeichen verloren gehen sollen.

```
IF RXD THEN GOTO GetBvte
```

INPUT

Die INPUT Anweisung dient zum Einlesen eines Befehls oder eines Wertes über die serielle Schnittstelle. Als Parameter wird eine Variable erwartet, in der der empfangene Wert gespeichert werden soll. Der im Textformat von der seriellen Schnittstelle erwartete Wert wird mit einem Carriage Return (0Dh) abgeschlossen. Das Programm wird erst nach der vollständigen Abarbeitung der Input Anweisung weitergeführt. Die Variable kann vom Typ Byte oder Word sein.
Manche Geräte senden ein CR *und* LF am Ende der Übertragung. Entfernen Sie in diesen Fällen das LF mit einer GET Anweisung aus dem Zeichenbuffer

Syntax: Input Variable
variable: Word/Byte Variable

```
INPUT Mvword
```

GET

Die GET Anweisung wartet solange bis ein Byte von der seriellen Schnittstelle oder durch eine Umleitung von einer der Objekte CONFIG,CONFIG2,IIC,IR oder RF empfangen wurde und speichert dieses in die angegebene Byte-Variable.

Syntax: Get Variable
variable: Byte Variable

```
GET MyByte
```

PRINT

Die PRINT Anweisung dient zur Ausgabe von Variablen, Zeichenketten, Portzustände und Zahlen an die serielle Schnittstelle. Ein am Ende der Print Anweisung stehendes Semikolon unterdrückt das automatische Senden des Zeilenvorschubs (ASCI 13, 10). Um mehrere Argumente mit einer Print Anweisung zu senden, können Sie den Verknüpfungsoperator & verwenden. PRINT steht auch für andere Objekte wie z.B. das LCD zur Verfügung. Sehen Sie Details dazu bitte im Kapitel OBJECTS

Syntax: Print Argument [& Argument [& ...]] [;]

```
PRINT "WERT :" &  Mvword & " V"
```

Die Verwendung des Zeichens # ist nicht zulässig, da es sich um ein Steuerzeichen handelt, das vom Betriebssystem verwendet wird.

PUT

Die PUT Anweisung sendet an die serielle Schnittstelle oder durch eine Umleitung an das Objekt CONFIG,IIC,IR or RF ein einzelnes Byte.

Syntax: Put Variable
variable: Byte Variable oder Konstante

```
PUT MyByte
```

SAY

Die SAY Anweisung gibt eine Text oder einen Variableninhalt als Sprache an DA1 aus. Voraussetztng dafür ist ein mit den Stimmsamples geladenes externes EEPROM. Auch eigene Klangdateien die in ein EEPROM geladen wurden sind so abrufbar. (siehe Kapitel SOUNDMODUL)

Syntax: Say Variable
variable: Byte, Word Floating Point Variable oder Konstante

```
SAY MyByte
SAY MyWord
SAY "Hallo Welt"
SAY
FPPrint(MyFloat,5)
```

INTERNE/EXTERNE DATEN SPEICHERUNG

Die Dateifunktionen erlauben das Aufzeichnen von Messwerten oder anderen Daten oder können zum Abspeichern von Information benutzt werden, die nach Ausfall der Betriebsspannung wieder in die Programmvariablen geladen werden sollen. Der Speicherbereich im FLASH nach dem Anwenderprogramm - meist der größte Teil - steht für diesen Zweck zur Verfügung. Der Speicherbereich wird als eine Datei verwaltet, auf die lesend oder schreibend zugegriffen werden kann, nachdem sie mit dem entsprechenden Attribut READ/WRITE geöffnet wurde.Die Speicherung von Float Daten wird im Abschnitt FOATING POINT MODULE beschrieben

ACHTUNG:
Das FLASH hat nur eine begrenzete Anzahl von zulässigen Schreibzyklen (100.000). Permanentes Schreiben auf das Flash wird also sehr schnell zur vollständigen Zerstörung des Programm-speichers führen.

OPEN# FOR READ - OPEN# FOR WRITE

Damit wird eine Datei zum sequentiellen Lesen oder Schreiben geöffnet und der Wert für FILEFREE initialisiert.
Dabei bedeutet WRITE das Öffnen zum Schreiben mit Überschreiben eventueller alter Aufzeichnungen. READ das Öffnen zum Auslesen der Aufzeichnungen. Gleichzeitig setzt OPEN den Datenzeiger (gemeinsam für READ und WRITE) auf den Anfangswert. Nach einem Reset ist der Dateizeiger gelöscht, es kann also nicht mit einer begonnenen Aufzeichnung fort gefahren werden, es sei denn die Datei wurde nach dem Schreiben mit CLOSE# geschlossen. Damit ist dann der Zeiger auf das Dateiende im FLASH gespeicher.

CLOSE#

CLOSE# schliesst eine Datei und sichert den Zeiger auf das Dateiende in das FLASH. Damit kann eauch nach einem Reset mit der Aufzeichnung an der letzten Position fortgefahren werden.

OPEN# FOR APPEND

Setzt den Datenzeiger auf die letzte Position im Datensatz, um beim nächsten PRINT# mit der Aufzeichnung fortzufahren.

INPUT#

Liest ein Byte von der durch den Datenzeiger adressierte Speicherstelle und inkrementiert den Datenzeiger.

Syntax: INPUT# Variable
variable: Word/Byte Variable

```
INPUT# MyWord
```

PRINT#

Schreibt ein Byte an die durch den WRITE Datenzeiger adressierte Speicherstelle und inkrementiert den Datenzeiger. Vor jedem Schreiben sollte geprüft werden, ob noch genügend Platz im FLASH zur Aufnahme der Daten vorhanden ist. Dafür kann die Funktion FILEFREE abgefragt werden, die als Ergebnis die Größe des noch freien Speichers liefert

Syntax: PRINT# Variable
*variable: W*ord/Byte Variable oder Konstante

```
PRINT# MyWord
```

FILEFREE

FILEFREE enthält den für die Datenaufzeichnung im FLASH verbleibenden Speicherplatz in Words. Die verbleibende Anzahl an Bytes ist also doppelt so gross wie der angezeigte Wert. Nach dem Laden eines BASIC-Programms ermittelt das Betriebssystem die für den Datenspeicher geltende Startadresse und berechnet die Grösse des nutzbaren Speichers. Da der Datenspeicher in Blöcken zu 64Byte organisiert ist, gibt es immer eine leichte Differenz zwischen dem tatsächlich zur Verfügung stehenden Datenspeicher und dem theoretischen Speicherplatz (Gesamtspeicher-BASIC-Speicher)Der ermittelte Wert wird mit dem ersten OPEN# an FILEFREE übermittelt. FILEFREE ist also vorher nicht initialisiert

Beispiel: Abfrage von FILEFREE

```
IF FILEFREE = 0 THEN GOTO X
```

EOF

EOF enthält den boolschen Wert der Abfrage ob die letzte gültige Position eines Datenfiles beim Lesen mit INPUT# erreicht wurde. Es wird also im Prinzip der aktuelle Datenzeiger mit der gespeicherten Position verglichen. Der Datenzeiger wurde mit CLOSE# gespeichert.

Beispiel:Abfrage EOF

```
IF  EOF THEN GOTO X
```

EEPROM.WRITE / APPEND / READ
Wenn nur wenige Bytes gesichert werden sollen gib es weiter Instruktionen welche das Öffnen und Schliessen der Datei im Huntergrund durchführen.

Syntax: EEPROM.Instruction [parameter,]
*Parameter: W*ord/Byte Variable oder Konstante

```
EEPROM.WRITE value1,value2
EEPROM.APPEND value3
EEPROM.READ value1. value2. value3
```

LOOKTAB / TABLE

Die LOOKTAB Anweisung gibt einen Word Wert aus einer definierten Tabelle(Table) wieder, wobei die Looktab Anweisung entweder als Funktion, oder traditionell als normale Anweisung verwendet werden kann.
Syntax: Looktab(Table, Index, Variable)
Variable = Looktab(Table, Index)

Index: Word/Byte Variable oder Konstante
Variable: Word/Byte Variable

```
LOOKTAB   MyTab,MyIndex,MyWord

TABLE myTab
0 1 2 3 4 5 6 7
END TABLE
```

Im Zusammenhang mit Tabellen gibt der Compiler eine Unterstützung, die sehr hilfreich ist, z.B. wenn die Tabelle sehr gross ist. Er ermittelt die Tabellenlänge und stellt sie in den Funktionen UBOUND/LBOUND zur Verfügung

Syntax: *LBound(MyTab) UBound(MyTab)*
MyTab: Erforderlich, Bezeichner für Tabelle.

```
FOR x = LBound(MyTab) TO
UBound(MyTab)
  PRINT LOOKTAB(MyTab,x)
NEXT

 TABLE MyTab
  1 2 3 4 5 6 7
```

CHIPRAM

Mittels CHIPRAM kann auf ein externes I²C-Bus EEPROM direkt als Datenspeicher zugegriffen werden. Das EEPROM ist als ausgelagerter Speicher zu verstehen und wird in BASIC++ mit CHIPRAM deklariert. Anders als bei beim internen Variablen-Speicher werden WORD und BIT-Variablen nicht unterstützt. Ansonsten erfolgt der Gebrauch dieser Variablen analog.

USER VARIABLEN EMULATION
CHIPRAM kann entweder als Emulation einer regulären User Variablen verwendet werden (dann muss die Variable wie üblich definiert werden) oder als indizierter Direktzugriff auf das EEPROM, ohne Definition eines Variablennamens, jedoch mit Angabe der Speicherzelle.

Define VAL as CHIPRAM	Benutzer Variable externer Speicher, vom Compiler verwaltet
Define VAL as CHIPRAM[1]	Benutzer Variable externer Speicher, vom Anwender verwaltet

Beispiel zur CHIPRAM Variablen Emulation:

```
define LIGHT as PORT[16]
define WERT1 as CHIPRAM[100]
define WERT2 as CHIPRAM

LIGHT=off
LCD.INIT
WERT1=10
WERT2=20
LCD.PRINT WERT1 & "  " & WERT2
```

Der Zugriff auf CHIPRAM ermöglicht einen sehr komfortablen Zugriff auf I²C-Bus EEPROMs, ändert aber leider nichts an den sehr geringen Schreibgeschwindigkeiten, die für EEPROMs typisch sind.
Dieses Feature wird also weniger als Erweiterung des Variablenbereiches zu nutzen sein, als vielmehr ein komfortables Mittel das EEPROM als Datenspeicher zu nutzen

Beachten Sie bitte dass EEPROMs nur eine begrenzte Anzahl von Schreibzyklen zulassen (meist 100.000 bis 1.000.000). Sehen Sie dazu bitte das Datenblatt des Herstellers

EEPROM DIREKT ZUGRIFF

Chipram bietet auch komfortablen Zugriff auf das EEPROM als Datenspeicher

Syntax: CHIPRAM(address)

Address: word/byte variable oder Konstante, welche diekt die Speicherzelle im EEPROM adressiert

Example for reading a value

```
MyByte=CHIPRAM(MyAddress)
```

Example for writing a value

```
CHIPRAM(MyAddress)=MyByte
```

```
define LIGHT as PORT[16]
define COUNTER as WORD

LIGHT=off
LCD.INIT
FOR COUNTER= 10 to 100
CHIPRAM(COUNTER)=150
NEXT
```

```
define LIGHT as PORT[16]
define COUNTER as WORD
define WERT as BYTE

LIGHT=off
LCD.INIT
FOR COUNTER= 10 to 100
WERT=CHIPRAM(COUNTER)
NEXT
```

Das Beispiel links beschreibt die Speicherstellen 10 bis 100 im EEPROM mit dem Wert 150.
Das Beispiel rechts liest diese Speicherstellen in die Variable WERT.
Der Vorteil von CHIPRAM ist, dass damit ein umfangreiches Unterprogramm (wie es bisher erforderlich war) ersetzt wird und vor allem, dass nicht über das I²C-Bus Objekt auf das EEPROM zugegriffen wird.

Zugriffszeiten:
Schreiben auf EEPROM 10ms
Lesen vom EEPROM 600us

CHIPRAM unterstützt MICROCHIP EEPROMS und auch Chipkarten auf der Adresse 160. Mit CHIPRAM kann also nur ein EEPROM (bis 64 KByte) betrieben werden. Für andere EEPROM Adressen gibt es spezielle Bibliotheken mit denen sich genauso komfortabel arbeiten lässt
Zu empfehlen ist ein 24LC256. Es hat 64kbyte Speicher.

VOICEBASE

Das Schlüsselwort VOICEBASE dient dazu, dem Soundmodul bei der Ausgabe von Sprache oder Klängen mitzuteilen welche EEPROM Adresse es dafür benutzen soll.

Das TTSS ist als default auf die Adresse 160 (dez) eingestellt, kann aber mit

```
VOICEBASE=174
```

auf jede beliebige Adresse eingestellt werden (hier z.B. 174) Es ist die EEPROM WRITE Adresse und ist immer geradzahlig. Stellen Sie VOICEBASE auf die Adresse des EEPROMS welches das Phonem oder Klangfile File enthält, das Sie abspielen wollen. (siehe Kapitel SOUNDMODUL)

VOICEBASE kann nicht gelesen werden. Ausdrücke wie

IF VOICEBASE = 174 THEN …..

sind nicht erlaubt.

Der @ OPERATOR

Wenn eine Variable als Pointer definiert wurde, dient der @ Operator dazu, die Speicherzelle zu lesen, welche durch den Pointer adressiert ist.

```
DEFINE Var1 AS WORD[3], Var2 AS
WORD[7]

' Ein Zeiger auf eine Word Variable
' deklarieren

DEFINE ^p AS WORD

' Variablen Var1 den Wert 200 zuweisen
' in Speicherstelle 3 steht nun der Wert
' 200:

Var1 = 200

' Der Zeiger p zeigt nun auf die Speicher-
' stelle von Var1 (also 3)
' Somit steht in p nun der Wert 3:

p = ^Var1

' Mit dem @ Operator kann man den Inhalt
' der Speicherstelle, auf die p zeigt
ausgeben

PRINT @p
```

STANDARD DIGITAL I/O PORTS

Nach einem RESET sind alle digitalen I/O Ports 1 bis 16 (bis auf Ausnahmen, siehe Kapitel CONFIG REGISTER) ein INPUT. Ports, welche als OUTPUT benutzt wurden können mit dem Schlüsselwort DEACT auf den Ursprungszustand INPUT zurückgeführt werden

PORT READ/WRITE

Bevor Ports im Programm angesprochen werden, müssen sie mit einem symbolischen Namen definiert werden (Kapitel DEKLARATIONEN and DEFINITIONEN).

Das Lesen von einem Port liefert dann den Portzustand als boolschen Wert, also TRUE/FALSE oder ON (TRUE, HI-Pegel) und OFF(FALSE, LO-Pegel).Der Port muss dazu ein Input sein.

Das Schreiben auf eine Port macht ihn zu einem OUTPUT mit dem entsprechenden Zustand OFF oder ON Schreiben und Lesen von einem Byteport liefert (oder setzt) den Zustand aller Ports als Byte Wert

Abfrage Bitport (Port ist Eingang)

```
IF  MyBitPort1=OFF THEN GOTO X
IF MyBitPort1=ON THEN GOTO X
```

Abfrage Byteport(Port ist Eingang)

```
IF  MyBytePort1=123 THEN GOTO X
```

Setzen von Bitport (Port ist Ausgang)

```
MyBitPort1=OFF
MyBitPort1=ON
```

Setzen von Byteport (Port ist Ausgang)

```
MyBytePort1=123
```

TOG

TOG (toggle) ist eine Instruktion die den gegenwärtigen Zustand eines Bitport-Ausgangs umkehrt. TOG hat keine Wirkung wenn der Port kein Digitalport ist, oder wenn er ein Digitalport-Eingang ist. TOG ist ausschliesslich für die digitalen Bitports 1 bis 16 anwendbar.

Syntax: TOG(bitport)

```
TOG MyBitPort
```

PULSE

Die PULSE Anweisung sendet kurze Impulse (5us) an einen Digitalport, wobei der Pegel des Ports kurz invertiert wird und dann wieder zurückgeschaltet wird. Achten Sie darauf, daß Pulse nur mit aktiven Ports verwendet werden kann. PULSE ist ausschliesslich für die digitalen Bitports 1 bis 16 anwendbar.

Syntax: PULSE(bitport)

```
TOG MyBitPort
```

DEACT

DEACT schaltet einen Digitalport-Ausgang als Eingang

Syntax: DEACT(bitport)

Beispiel Deaktivieren eine Digitalport Ausgangs

```
DEACT MyBitPort1
```

EXTENDED PORTS

Extended Ports werden behandelt wie die regulären digital en I/O Ports 1 bis 16, weisen aber hinsichtlich der Hardwareaspekte einige Unterschiede auf. Sehen Sie hierzu bitte im Anhang das Kapitel EXTENDED PORTS.

AD PORTS

Eine der Alternativen Port-Funktionen ist die eines AD-Wandlers.
Bevor ein AD-Port benutzt werden kann muss er mit einem symbolischen Namen als solcher definieret werden (siehe Kapitel DEKLARATIONEN and DEFINITIONEN). Danach wird er im Programm ausschliesslich mit diesem Namen angesprochen.

Das Lesen liefert einen Byte Wert zurück, welcher der Eingangsspannung des AD-Wandlers entspricht.

Beispiele zum Lesen des AD Wandlers:

```
IF MyAnalogIn = 100 THEN GOTO X
MyByte=MyAnalogIn / 10
```

AD PORTS DIGITAL MODE

Die AD-PORTS 1 bis 8 können wahlweise als Digitalports benutzt werden.Das kann sowohl bitweise als auch byteweise geschehen.D.h sie können z.B. nur den AD1 und AD2 als Analog-Digital Wandler und die anderen Ports als reine DIGITALPORTS nutzen. Sie müssen sich bewusst sein, dass der BYTEPORT[3] die gleichen Pins bezeichnet wie ad[1] bis ad[8] und port[17] bis port[24]. Der Unterschied liegt nur in der Funktion. Ist die alternative Funktion der AD-Ports einmal im CONFIG2 Register aktiviert (BIT 7), so ist der EXTPORT3 (ein PCF8574 IIC-BUS Port) abgeschaltet, da er ursprünglich als BYTEPORT[3] definiert war. Die anderen EXTPORTS 4 bis 18 sind natürlich nutzbar.

Grundsätzlich gilt für die AD-PORT DIGITAL OPERATION:

- Die alternative Funktion muss im CONFIG2 Register aktiviert werden.
- dann kann jeder Port als DIGITALPORT 17-24 oder als BYTEPORT3 gelesen oder beschrieben werden.
- Die DIGITALPORTS 17-24 bleiben so lange in der Betriebsart DIGITALPORT bis sie als ANALOGPORT (z.B.x=ADC1) angesprochen werden.
- Dann verbleibt er als ANALOGEINGANG und zwar so lange, bis er als DIGITALPORT angesprochen wird (z.B. DIGIPORT17=off) , danach ist er wieder ein DIGITALPORT

ACHTUNG:
- DIGITALPORTS 17-24 können nicht mit DEACT deaktiviert werden, sie müssenals AD PORT angesprochen werden und sind dann ein ANALOGPORT und damit ein INPUT was gleichbedeutend mit deaktiviert ist.
- Es ist grundsätzlich kein PULLUP wirksam und auch nicht schaltbar, offene Ports nehmen also undefinierten Pegel an.
- PULSE, DEACT und TOG sind nicht zulässig.

Beispiel zum DIGITAL MODE der AD Ports:

```
define ADC1          as  ad[1]
define TESTPORT      as port[17]
define BP3       as  byteport[3]
define WERT          as byte[1]

      TESTPORT=off      ' Schaltet den port[17]  ( entspricht ad[1] ) als DIGITALPORT LO
      WERT=ADC1         ' Reaktiviert den DIGITALPORT port[17] als ADPORT ad[1]
                        ' und schreibt den Analogwert in die Variable
      BP3=on            ' schaltet alle PORTS [17] bis [24] als DIGITAL
                        ' PORTS HI, alle AD-Ports sind abgeschaltet.
```

DA PORTS

Bevor ein DA-Port benutzt werden kann muss er mit einem symbolischen Namen als solcher definieret werden (siehe Kapitel DEKLARATIONEN and DEFINITIONEN). Danach wird er im Programm ausschliesslich mit diesem Namen angesprochen.
Das Schreiben eine Byte-Wertes setzt den DA Ausgang auf den entsprechenden durchschnittlichen Spannungswert.

Beispiel zum Schreiben des DA Port:

```
MyAnalogOut =10
```

MATH FUNCTIONS

Alle Mathematischen Funktionen und Operationen sind auf Byte oder Word Grösse beschränkt. Das bedeutet, dass selbst wenn Einggangs und Ausgangsgrössen im Wertebereich eine Byte/Word liegen, dafür Sorge getragen werden muss, dass Zwischenergebnisse bei Berechnungen keinen Überlauf erzeugen.

Beispiel für Byte Variablen: x=100*3/2

Das Ergebnis ist 150 und ist im gültigen Wertebereich. Die Berechnung 100*3 jedoch führt zu einem Überlauf und damit zu einem falschen Ergebnis. Hier sieht man deutlich dass die Reihenfolge der Berechnung eines Term durchaus von Bedeutung ist.

x=100/2*3 führt zu einem richtigen Ergebnis
Die MATH FUNCTIONS für Float Variablen sind im Kapitel FOATING POINT MODULE beschrieben.

GRUNDRECHENARTEN

Syntax: *Variable = value1 [operator] value2*

Variable: Variable Byte oder Word typ
value1: Variable, Wert oder Konstante, Byte oder Word Typ
value2: Variable, Wert oder Konstante, Byte oder Word Typ
operator: division /
multiplikation *
addition +
subtraktion

```
MyWord=Value1 / Value2
MyWord=Value1 * Value2
MyWord=Value1 + Value2
MyWord=Value1 - Value2
```

INC DEC (INKREMENT , DECREMENT)

Für eine Inkrementierung um 1, wie sie häufig vorkommt gibt es die eine bedeutend schnellere und Speicher sparende Variante. INC erhöht dabei den Wert der Variablen, DEC verringert den Wert der Variablen um jeweils eins.

```
MyWord=MyWord + 1
```
Traditionelle Variante

```
INC MyWord
```
Variante mit INC

Syntax: *INC(Variable)*
Variable: Variable Byte oder Word Typ
INC und DEC haben eine deutliche höhere Ausführungsgeschwindigkeit (66us gegenüber 196us) und benötigen auch weniger Token (2 Token gegenüber 8 Token)
Bei vielen Anwendungen welche eine FOR TO NEXT Schleife benutzen ist die Schrittweite (STEP) gleich eins. Auch für diese Fälle sind die wesentlich schnelleren Befehle INC und DEC zu bevorzugen.

Die FOR TO NEXT Variante: 650us

```
FOR MyByte= 0 to 10
NEXT
```

Die Variante mit INC: 230us

```
MyByte=0
#LOOP
INC MyByte
IF MyByte<10THEN GOTO LOOP
```

MAX / MIN

Die MAX Funktion liefert aus zwei Byte oder Word Zahlen die größte Zahl zurück. Die MIN Funktion ist das Gegenstück zu dieser Funktion.

Syntax: *Variable = Max(value1, value2)*

```
MyWord=MAX(MyByte1,MyByte2)
```

Variable: Variable Byte oder Word typ
value1 Variable, Wert oder Konstante, Byte oder Word Typ
value2 Variable, Wert oder Konstante, Byte oder Word Typ

ABS

Die ABS Funktion gibt den Absolutwert (Betrag) des übergebenen Parameters wieder.

Syntax: *Variable = ABS(value1)*

```
MyWord=ABS(MyWord)
```

Variable: Variable Byte oder Word Typ
value1 Variable, Wert oder Konstante, Byte oder Word Typ

SGN

SGN liefert -1 zurück wenn der Wert < 0 ist, sonst immer Null

Syntax: *Variable = SGN(value1)*

```
MyWord=SGN(MyWord)
```

Variable: Variable Byte oder Word Typ
value1 Variable, Wert oder Konstante, Byte oder Word Typ

RAND RANDOMIZE

Die Funktion RAND liefert eine Zufallszahl vom Datentyp Word (zwischen -32768 und 32768) zurück. Es sollte einmalig vor dem Aufruf von Rand der Befehl RANDOMIZE zur Initialisierung des Zufallsgenerators aufgerufen werden.

Syntax: RANDOMIZE *init*
Variable = RAND

```
RANDOMIZE 2000
MyWord=RAND
```

Variable: Variable Byte oder Word Type
init: Variable oder Konstante, Byte oder Word Type

SQR

Die SQRT Funktion liefert einen ganzzahligen Näherungswert einer Quadratwurzel zurück.

Syntax: *Variable* = Sqrt(*value*)

```
MyWord=SQRT(MyWord)
```

Variable: Variable Byte oder Word Typ
value Variable, Wert oder Konstante, Byte oder Word Typ

MOD

Die MOD (Modulo) Funktion liefert den Rest einer Division.
Beispiel: 10 MOD 3 liefert z.B. 1 als Ergebnis.

Syntax: *Variable = value1 MOD value2*

```
MyByte=MyByte1 MOD MyByte2)
```

Variable: Variable - Byte oder Word Typ
value1 Variable, Wert oder Konstante - Byte oder Word Typ
value2 Variable, Wert oder Konstante - Byte oder Word Typ

MATH AND BOOLEAN OPERATORS

Die MATH AND BOOLEAN FUNCTIONS für Float Variablen sind im Kapitel FOATING POINT MODULE beschrieben. Bei der Berechnung von Termen ist die Hierarchie der Operatoren sehr wichtig.

RANK	**OPERATOR**
9	( )
8	MATH FUNCTIONS
7	NEGATIVE SIGN
6	MULTIPLY DIVISION MOD SHL SHR
5	PLUS MINUS
4	COMPARES
3	NOT
2	AND
1	OR

COMPARES - VERGLEICHE
> (grösser) <(kleiner) >=(grösser oder gleich) <=(kleiner oder gleich) =(gleich) <>(ungleich)

Vergleiche sind zulässig für Byte-Werte, Terme und Funktionen.

```
IF MyByte1 <= MyByte2 THEN GOTO X
IF  MyByte1*10>100/(MyByte2) THEN GOTO X
```

SHL - SHR
Das SHL Schlüsselwort dient zum Verschieben der Bits einer Byte oder Word Variable nach links. SHL bedeutet Shift Left (nach links verschieben) SHR schiebt nach rechts. Es handelt sich in beiden Fällen um eine logische Verschiebung, d.h. es wird eine NULL eingeschoben, das herausgeschobene Bit geht verloren. SHL um ein Bit ist gleichbedeutend mit einer Multiplikation mit 2, SHR bedeutet eine Division durch 2.

Syntax: *Variable = value1 SHL value2*

```
MyByte1 = MyByte2 SHL 8
```

Variable: Variable of Byte oder Word type
value1 Variable, Wert oder Konstante - Byte oder Word Typ
value2 Variable, Wert oder Konstante - Byte oder Word Typ

AND, OR XOR NOT

sind boolsche Operatoren und dienen zur bitweisen Manipulation von Werten

Syntax: *Variable = value1 AND value2*

```
MyWord=MyByte1 AND MyByte2
```

Variable: Variable of Byte oder Word type
value1 Variable, Wert oder Konstante - Byte oder Word Typ
value2 Variable, Wert oder Konstante - Byte oder Word Typ

Bei der Verwendung innerhalb von IF THEN entspricht das der logisschen Verknüpfung von TRUE/FALSE welche wiederum TRUE/FALSE als Ergenis liefert.

```
IF (MyByte=1) OR (MyByte=2) THEN GOTO X
```

PROGRAMM FLUSS KONTROLLE

PAUSE

Stoppt die Programmausführung für eine bestimmte Zeit. Systeminterrupts z.B. die der RTC werden weiterhin ausgeführt. Die Zeit für die PAUSE wird in Einheiten von 20ms angegeben. PAUSE 50 hält das Programm demnach für 1 Sekunde an.

Syntax: PAUSE = *value1*

```
PAUSE 25
```

V*alue1:* Variable, Wert oder Konstante, Byte oder Word Typ

FOR, TO, NEXT, STEP, EXIT FOR

Wie fast jeder Programmiersprache verfügt auch Basic++ über eine Zählerschleife. Eine Zählerschleife weist einer Variable am Anfang einen Wert zu und addiert zu der Variable bei jedem Schleifendurchgang einen festgelegten Wert bis die Abbruchbedingung erfüllt ist. Solange innerhalb der Zählerschleife (FOR...NEXT) die Zählervariable nicht manipuliert wird kann es auch nicht zu einer Endlosschleife kommen. Der Nachteil der FOR Schleife ist jedoch, dass Sie ohne Manipulation der Zählervariable nicht so flexibel ist wie die DO...LOOP Schleife.

Mit STEP nach der Abbruchbedingung können Sie festlegen inwiefern die Zählervariable bei jedem Durchlauf inkrementiert werden soll. Wenn Sie STEP
weglassen wird die Variable standardmäßig um 1 inkrementiert. Eine Zähler-schleife sollte immer dann benutzt werden, wenn die Anzahl der Schleifen-durchläufe entweder als Konstante oder Variable bekannt ist.

EXIT FOR führt zu einem vorzeitigen Verlassen der Schleife.
FOR TO NEXT Schleifen können verschachtelt werden.

Syntax: For [Counter] = [Start] To [End] [Step [Incr]]
[Exit For]
Next

```
FOR MyWord=MyStart TO MyEND STEP  MyStep
PRINT MyWord
IF MyPort=OFF THEN EXIT FOR
NEXT MyWord
```

Counter: Variable used as counter.
Start: Variable, value or constant which is moved to the counter variable as start condition.
End: Variable, value or constant, the counter variable is compared with.
Incr: Required if STEP is used. Term, variable, value or constant defining the increment.

DO, LOOP UNTIL, EXIT DO

Die DO...LOOP Schleife ist eine sog. Wiederholungsschleife, wobei die Schleifenabbruchbedingung nach der Ausführung des Codes innerhalb des Schleifenblocks geprüft wird. Die Schleife wird also mindestens einmal ausgeführt. Die DO...LOOP Schleife ist blockorientiert aufgebaut und kann somit mehrzeilige Anweisungen enthalten und Verschachtelt verwendet werden. Die Schleife kann mit EXIT DO vorzeitig verlassen werden

Syntax: Do
[instruction]
[Exit Do]
Loop (Until [expression])

```
DO
Mybyte=Mybyte+1
IF MyPort=OFF THEN EXIT DO
LOOP  UNTIL MyByte=5
```

WHILE, EXIT WHILE

Im Gegensatz zur DO...LOOP Schleife prüft WHILE vor dem Schleifendurchlauf ob die Abbruchbedingung erfüllt ist. Die Schleife kann mit EXIT WHILE vorzeitig verlassen werden.

Syntax: While
[expression]
[Exit While]
Wend

```
WHILE MyByte < 10
Mybyte=Mybyte+1
WEND
```

IF, THEN, ELSE, END IF

Die IF Bedingung ist die am meisten verwendete Kontrollstruktur in Basic.
Mittels IF können Sie abfragen, ob eine Bedingung erfüllt ist oder nicht und abhängig davon weitere Befehle ausführen. Beim IF Befehl handelt es sich um einen mehrzeiligen Befehl einen sogenannten Block. Das Blockende wird durch das Schlüsselwort END IF markiert.
Zwischen IF und END IF können Sie beliebig viele Anweisungen einfügen, die ausgeführt werden, wenn die Bedingungen erfüllt sind. Optional können sich zwischen IF und END IF noch ELSE einfügen. Alle Anweisungen zwischen ELSE und END IF werden ausgeführt, wenn die IF Bedingungen nicht erfüllt sind.
Natürlich können Sie auch IF Anweisungen ineinander Verschachteln. Achten Sie hier jedoch besonders darauf, dass jeder IF Block mit END IF geschlossen wird.
Sollte ein IF Block offen bleiben oder eine END IF Markierung ohne passende IF Anweisung auftreten meldet sich der Compiler mit einem Fehler.

Syntax1: *If [expression] Then [instruction] [Else] [instruction]*

```
IF MyByte=10 THEN GOTO X ELSE GOTO Y
```

Syntax2: *If [expression] Then*
[instructions]
[Else]
[elseinstructions]
End If

```
IF MyWord=10 THEN
    PRINT "This is "
    PRINT MyWord
ELSE
    PRINT "No Match"
    PRINT "found"
END IF
```

SELECT CASE, CASE, CASE ELSE

Neben dem IF Befehl können Sie noch den SELECT Befehl verwenden um eine Selektion in Ihren Programmablauf einzufügen. Der SELECT Befehl eignet sich besonders immer dann, wenn mehrere unterschiedliche Bedingungen abgefragt werden sollen. Wie der IF Befehl ist auch der SELECT Befehl mehrzeilig angelegt. Der SELECT Block schließt man mit END SELECT.Die einzelnen Bedingungen werden mit dem Schüsselwort CASE (jeweils zeilenweise) eingeschoben. Analog zum IF Befehl kann man optional auch mit CASE ELSE einen Zweig einfügen, der ausgeführt wird, wenn keine andere der bisher gelisteten CASE Bedingungen erfüllt war.Ebenso wie IF können auch SELECT Anweisungen verschachtelt werden. Auch hier müssen Sie wieder darauf achten, dass jeder SELECT Block durch END SELECT geschlossen wird.

```
SELECT CASE i
   CASE 1
      PRINT "1"
   CASE 2
      PRINT "2"
   CASE ELSE
      PRINT "No Match"
END SELECT
```

Select Case matchexpression
[Case expression]
[instructions]
[Case Else expression]
[elseinstructions]
End Select

WAIT

Eine wesentliche Voraussetzung für ein Programm ist eine Steuerung des Programmflusses, z.B. Verzweigungen. Die WAIT Anweisug ist die kurze Ausführung einer Bedingungsabfrage mit anschliessendem GOTO. Sie wartet bis die Bedingung erfüllt, also TRUE ist

Syntax: WAIT [expression]

```
WAIT (Mybitport = ON)
```

Weil ein Bitport immer Typ Bool ist kann man auch schreiben:

```
WAIT (MyBitPort)
```

GOTO

Sprungmarken sog. „Labels“ werden Ihnen beim Programmieren von Basic++ nicht so häufig begegnen, wie beim klassischen BASIC. BASIC++ ist viel modularer strukturiert und damit übersichtlicher. Hin und wieder sind Sprungmarken dennoch ganz nützlich. Sprungmarken werden mit dem Hash-Zeichen („#“) deklariert, wobei beim Aufruf der Sprungmarke das Hash-Zeichen nicht Bestandteil des Namens ist.
Achten Sie darauf, dass ein Label niemals lokal, sondern nur global initialisiert wird. Sie können auf eine Sprungmarke verweisen, die noch nicht deklariert wurde. Wenn nach der Kompilierung die Sprungmarke jedoch nicht gefunden wurde erscheint ein Fehler.
Mit Hilfe des GOTO Befehls gelangen Sie zu einer Sprungmarke, es wird keine Rücksprungadresse gesichert.

Syntax: GOTO Label

```
MyLabel:    IF (MyBitPort=OFF  THEN  GOTO MyLabel
```

Das kleine Beispiel entspricht: WAIT MyBitPort.

FUNCTION

ein Unterprogramm leiten Sie mit dem Befehl FUNCTION ein. Jedes Unterprogramm benötigt einen eindeutigen Namen, wobei der Name den allgemeinen Namenskonventionen entspricht, die Sie auch schon für Variablen- und Sprung-markennamen kennen gelernt haben.Wie Sie an der Syntaxdefinition schon erkennen können müssen auch Funktionen mit END FUNCTION geschlossen werden.
Mit RETURN können Sie einer Funktion einen Rückgabewert zuweisen. Verwechseln Sie das RETURN nicht mit dem RETURN des standard BASIC
Eine FUNCTION kann vorzeitig mit EXIT FUNCTION verlassen werden.
Zwischen der geöffneten und geschlossenen Klammer hinter dem Funktionsname können Sie optional Parameter listen, die beim Aufruf der Funktion gesetzt werden müssen. Parameter sind lokale Variablen, die nur innerhalb der Funktion verwendet werden können. Der Umgang und die Gebrauchskonventionen von Parameter entspricht dem von normalen Variablen, die mit DEFINE erstellt wurden.

Syntax: Function [Name] ([[Parameter] As [Data type] | [Parameter] Ref [Variable], ...])
[instructions]
Return [expression]
End Function

Name: Erforderlich, Gültiger Bezeichner
Parameter: Optional, Gültiger Bezeichner für eine Parametervariable
Data type: Erforderlich, wenn kein Ref bei Parametervariable. Gültiger Datentyp
Variable: Erforderlich, wenn Ref bei Parametervariable. Bezeichner einer schon deklarierten Variable
Expression Optional, Term, Variable, Zahl oder Konstante als Rückgabewert
Bei der Parameterübergabe lassen sich Variablen auch Referenzieren

```
FUNCTION MyFunction(X as byte, Y as byte)
    X=2*Y
    RETURN 2*X
END FUNCTION
----------------------------------------------------------
PRINT MyFunction(1,2)
```

```
FUNCTION MyFunction(X Ref MyByte, Y Ref MyWord)
    X=2*Y
    RETURN 2*X
END FUNCTION
----------------------------------------------------------
PRINT MyFunction(1,2)
```

OBJECTS

Einige spezielle Interfaces wie das LCD,IR,RF,I²C und CONFIG arbeiten intern mit der Umleitung der Instruktionen PUT,GET,PRINT von der seriellen Schnittstelle auf das jeweilige Objekt. nach dem RESET ist immer die Schnittstelle aktiv, und keine Umleitung ist wirksam. Vor dem Zugriff auf einer der genannten Objekte muss eine Umleitung darauf aktiviert werden - das OBJECT muss initialisiert werden

OBJECT.INIT

Danach erfolgt in der Regel ein Schreib/Lesezugriff auf das OBJECT. Es werden immer Byte Werte übertragen, selbst wenn der Wert ein Word ist (dann wird nur das LO Byte übertragen)

Solange keine anderen OBJECTS oder die serielle Schnittstelle benötigt wird kann das OBJECT geöffnet bleiben. Andernfalls muss vor dem Öffnen eines neuen OBJECTS, das gerade aktive OBJECT geschlossen werden.

OBJECT.OFF

CONFIG REGISTER

Das Config Objekt dient zur Konfiguration alternativer Portfunktionen und anderer Optionen. Um die jeweilige Funktion aufzurufen müssen Sie die jeweiligen BITs des Config-Registers setzten.

Mit Config.Put können Sie das Config-Register beschreiben und mit Config.Get können Sie den Wert des Config-Register lesen. Diese beiden Anweisung verhalten sich dabei analog zu den PUT und GET Befehlen der seriellen Schnittstelle.

CONFIG (CONFIG REGISTER 1)

Bit 0 - Switch both PWM-DACs to SERVO-Mode
Bit 1 - Switch frequency counter 1 to event counter mode
Bit 2 - Switch frequency counter 2 to event counter mode
Bit 3 - switch on PULLUP- resistors for PORT 1 to 8
Bit 4 - switch on PULLUP- resistors for PORT 9 to 15
Bit 5 - radio controlled synchronising of RTC. Clock is Sync
Bit 6 - IIC-BUS Communication Error
Bit 7 - Mirrors logic level at Start button input port

Die Funktion is aktiv, wenn das ensprechende Bit HI ist

CONFIG2 (CONFIG REGISTER 2)

Bit 0 - IRQ disabled and replaced by 20 ms timer interrrupt
Bit 1 - IRQ line logic level
Bit 2 - IRQ disabled, replaced by RF-Module interrupt.
Bit 3 - IRQ disabled, replaced by IR-Module interrupt.
Bit 4 – DA1 und DA2 Voice Mode
Bit 5 – Disable Sound/Speech Compression
Bit 7 - AD-Ports now acting as digital BYTEPORT 3, external

Die Funktion is aktiv, wenn das ensprechende Bit HI ist

Syntax: OBJECT.instruction

CONFIG.INIT
OBJECT wird initialisiert (geöffnet). Vorher müssen andere OBJECTS geschlossen werden, sofern sie geöffnet wurden

Syntax: OBJECT.instruction Variable
variable: Variable oder Konstante von Typ Byte/Word
CONFIG.PUT
CONFIG.GET
Schreiben (PUT) und Lesen (GET) eines Byte Wertes auf oder von einem OBJECT

Syntax: OBJECT.instruction
CONFIG.OFF
Schliessen eines OBJECTS

```
CONFIG.INIT
CONFIG.GET MyByte
MyByte = MyByte OR 00000001b
CONFIG.PUT MyByte
CONFIG.OFF
```

Beispiel: Bitmanipulation im CONFIG REGISTER PWM DA Ports werden in den Servo Mode geschaltet.

IIC (I²C-BUS OBJECT)
Das IIC Objekt erleichtert die Handhabe mit Komponenten, die an den I²C Bus angeschlossen sind. Dabei muss der Anwender auf Grund der gerätespezifischen Protokolle der einzelnen I²C Komponenten jedoch Kenntnisse über die am I²C angeschlossenen Geräte haben.

Mit den I²C BUS Komponenten wird durch PORT 9 (SDA) und PORT 10 (SCL) kommuniziert.

Neben IIC.Init und IIC.Off unterstützt das IIC Objekt eine Reihe weiterer Funktionen. Die IIC.Start Funktion sendet eine START Sequenz, die IIC.Stop Funktion sendet eine STOP Sequenz. IIC.Send sendet den Wert eines Terms an den Bus. IIC.Get fragt den Wert am I²C BUS ab und IIC.Print sendet eine Zeichenkette an den I²C BUS.

Syntax: OBJECT.instruction
IIC.INIT
OBJECT wird initialisiert (geöffnet). Vorher müssen andere OBJECTS geschlossen werden, sofern sie geöffnet wurden.

Syntax: OBJECT.instruction
IIC.START
IIC.STOP
Sendet die I²C-Bus START und STOP Signale, die Teil des Protokolls sind.

Syntax: OBJECT.instruction Valiable
variable: Variable oder Konstante von Typ Byte/Word
IIC.GET
IIC.SEND
Schreiben (SEND) und Lesen (GET) eines Byte Wertes auf oder von einem I²C-Bus Gerät

Syntax: OBJECT.instruction
IIC.OFF
Schliessen eines OBJECTS

```
IIC.INIT
IIC.START
IIC.SEND MyAddress
IIC.SEND MyByte
IIC.STOP
IIC.OFF
```

Das Beispiel überträgt MyByte an die I²C-Bus Adresse MyAddress

IR OBJECT

Das Format wird von den meisten Geräten der Unterhaltungsindustrie verwendetund ermöglicht es Ihnen diese Geräte mit dem BASIC-Computer zu steuern. Das RC5 Format besteht aus einer Geräteadresse und einem Kommando. Das IR-MODULE erwartet zwei Parameter zu IR.SEND (jeweils die Adresse, danach das Kommando)und sendet dann den entsprechenden Datenrahmen. Empfangene Daten werden in jeweils zwei Variablen abgerufen. Eine Enthält die empfangenen Daten, die andere die Geräteadresse des sendenden Geräts.
Das IR-OBJECT erwartet an Port 2 den Ausgang eines IR-Empfängers (z.B. TSOP 4836) und an Port 3 eine IR-LED als Datensender. Wollen Sie nur Daten empfangen, so ist Port 3 zur freien Verfügung.

```
 RC5 FORMAT:
    13-12-11-10-09-08-07-06-05-04-03-02-01-00   DATA BIT
     S  S  T a4 a3 a2 a1 a0 c5 c4 c3 c2 c1 c0   RC5 Format

S = Start Bit (auto set/remove by the IR OBject)
T = Toggle
a = Address
c = Command

The IR MODULE expects a receiver connected at port 2 and the
transmitter connected to port 3
```

Syntax: OBJECT.instruction
IR.INIT
OBJECT wird initialisiert (geöffnet). Vorher müssen andere OBJECTS geschlossen werden, sofern sie geöffnet wurden.

Syntax: OBJECT.instruction Address,Data
IR.GET
IR.SEND
Schreiben (SEND) und Lesen (GET) eines Byte Wertes auf oder von einem IR-Gerät

Address: Variable oder Konstante die als Adresse gesendet oder empfangen wird.
Byte und Word ist gültig

Data: Variable oder Konstante die als Daten gesendet werden
Variable in welche das empfangene Datenbyte geschrieben wird. Byte und Word ist gültig

```
IR.INIT
IR.SEND MyAddress , MyByte
IR.OFF
```

```
IR.INIT
IR.GET MyAddress , MyByte
IR.OFF
```

Beide Werte, Adress und Data werden als 255 gelesen wenn kein gültiger Datenrahmen empfangen wurde. werden Adress und Data gelesen, ist der Empfangsbuffer geleert und wird auf den Wert 255 gesetzt.

Syntax: OBJECT.instruction
IR.OFF
Schliessen eines OBJECTS

RF OBJECT

Das verwendete Datenformat ist du den gebräuchlichsten Geräten der Haushaltstechnik kompatibel, obwohl es hier keinen Standard gibt. Es wird eine 8Bit lange Geräteadresse verwendet (die beim Empfänger entsprechend eingestellt sein muss) und ein 4Bit langes Kommando um z.B. eine Relais am Empfänger zu schalten. Das RF-OBJECT erwartet nach der RF.SEND Anweisung jeweils die Adresse, und danach das Kommando und sendet den entsprechenden Datenrahmen.
Empfangene Daten werden mit RF.GET in zwei entsprechende Variablen geschrieben.
Bei den meisten kompatiblen Geräten (z.B. Funk-Steckdosenschalter) sind zur Sicherheit vor Störungen 4 solcher Datenrahmen zu senden.' Das RF-Module erwartet an Port 2 den Ausgang eines 433MHz-Empfängers und an Port 3 einen 433MHz Sender. Wollen Sie nur Daten empfangen, so ist Port 3 zur freien Verfügung.

```
    FORMAT:
          11-10-09-08-07-06-05-04-03-02-01-00   DATA BIT
          c3 c2 c1 c0 a7 a6 a5 a4 a3 a2 a1 a0   HT12 Format

a = Address
c = Command

The RF MODULE expects a receiver connected at port 2 and the
transmitter connected to port 3
```

Syntax: OBJECT.instruction
RF.INIT
OBJECT wird initialisiert (geöffnet). Vorher müssen andere OBJECTS geschlossen werden, sofern sie geöffnet wurden.

Syntax: OBJECT.instruction Address,Data
RF.GET
RF.SEND
Schreiben (SEND) und Lesen (GET) eines Byte Wertes auf oder von einem RF Gerät

Address: Variable oder Konstante die als Adresse gesendet oder empfangen wird.
Byte und Word ist gültig

Data: Variable oder Konstante die als Daten gesendet werden
Variable in welche das empfangene Datenbyte geschrieben wird. Byte und Word ist gültig

```
RF.INIT
Rf.SEND MyAddress , MyByte
RF.OFF
```

```
RF.INIT
RF.GET MyAddress , MyByte
Rf.OFF
```

Beide Werte, Adress und Data werden als 255 gelesen wenn kein gültiger Datenrahmen empfangen wurde. werden Adress und Data gelesen, ist der Empfangsbuffer geleert und wird auf den Wert 255 gesetzt.

Syntax: OBJECT.instruction
RF.OFF
Schliessen eines OBJECTS

LCD OBJECT

Das LCD Objekt ermöglicht die komfortable Ausgabe von Zahlen und Strings auf einem LC Display. Neben einer LCD.PRINT Anweisungen stehen Ihnen auch Funktionen zur Formatierung der Ausgabe zur Verfügung. Wie beim allen Modulen müssen Sie auch das LCD Objekt mit INIT initialisieren und mit OFF ausschalten. Mit dem Befehl CLEAR löschen Sie den Text auf dem LC Display. POS gibt die Positionierung des Textes auf dem Display an. Die Befehle SR und SL zum „Right Scroll" bzw. „Left Scroll".
PRINT kann sowohl Strings als auch Zahlen auf dem Display ausgeben.

Syntax: OBJECT.instruction
LCD.INIT
OBJECT wird initialisiert (geöffnet). Vorher müssen andere OBJECTS geschlossen werden, sofern sie geöffnet wurden. Gleichzeitig wird der Zeichenspeicher des LCD gelöscht und der Cursor auf den Anfang der Zeile 1 gestellt. Die Initialisierung benötigt etwa 20ms, und ist zu lange wenn man zwischen dem LCD OBJECT und anderen OBJECTS hin und herschaltet. LCD.INIT wird nur eimalig am Programmstart verwendet. Danach ist es vorteilhafter das LCD.OBJECT nur noch wieder zu öffnen und auf die komplette Initialisierung zu verzichten LCD.INIT switchonly

Syntax: OBJECT.instruction
LCD.INIT switchonly
Öffnen des LCD OBJECTs ohne Initialisierung.

Syntax: OBJECT.instruction
LCD.CLEAR
LCD löschen und Cursor auf Position 1, Zeile 1 stellen. Benötigt 2ms zur Ausführung.

Syntax: OBJECT.instruction,Zeile,Position
LCD.POS
Setzt den Cursor auf Zeile und Position
Gültig ist Zeile 1 und 2, sowie Position 1 bis 16. Hier sind nur Konstanten zulässig.
Der Bereich jenseits von Position 16, ist jedoch nur sichtbar wenn der LCD Inhalt nach links geschoben wird.

Syntax: OBJECT.instruction
LCD.SR
LCD.SL
LCD Zeichenbuffer rechts(SR) oder links (SL) schieben. Bei links schieben werden die Zeichen sichtbar die jenseits der Position 16 im Zeichenbuffer stehen.

LCD.PRINT
Die PRINT Anweisung dient zur Ausgabe von Variablen, Zeichenketten, Portzustände und Zahlen an das LCD Um mehrere Argumente mit einer Print Anweisung zu senden, können Sie den Verknüpfungsoperator & verwenden. LCD.PRINT unterscheidet sich kaum von einem PRINT das an die serielle Schnittstelle gerichtet ist. Details zur Ausgabe von Floating Point Werten finden sie im Kapitel FLOATING POINT MODULE

Syntax: Print Argument [& Argument [& ...]] [;]

```
LCD.PRINT "WERT :" &  Myword & " V"
```

LCD.OFF
Close the Object prior to open any other Object.

Syntax: OBJECT.instruction

START OPTION REGISTER

Die Control Units haben die Möglichkeit beim Programmstart Optionen zu definieren. Diese werden einmalig durch Laden eines entsprechenden Programms aktiviert und sind dann dauerhaft aktiv, können aber mit einem gesonderten Programm wieder rückgängig gemacht werden.
Das START-OPTION REGISTER darf nicht mit den CONFIG REGISTERN verwechselt werden. Die CONFIG REGISTER können zur Programmlaufzeit geändert werden und die Einstellungen sind nur bis zum nächsten Reset aktiv.

Da der Compiler diese Optionen nicht unterstützt kann man die entsprechenden Token mit ADDTOKEN erzeugen.

ADDTOKEN 39 Diese Instruktion schreibt den darauf folgenden Wert in das Option Register
ADDTOKEN 3 Dieser Wert 0000 0011 setzt im START OPTION REGISTER das Bit 0 und 1

Ein Programm das nur diese zwei Zeilen enthält muss nur ein einziges mal laufen um die Option dauerhaft einzustellen. Sie kann natürlich durch ein anderes Programm wieder rückgängig gemacht werden. Auch hierzu finden Sie fertige Programme bei den Beispielen auf der Installations Disk

Die Option Autostart ist nur für die M ADV Control Unit verfügbar.

M ADV

Bit 0	**EEPROM BOOT OPTION**
Bit 1	**AUTOSTART AFTER PROGRAMM DOWNLOAD**

Die Funktion is aktiv, wenn das entsprechende Bit gesetzt ist. Es können beide Optionen aktiv sein.

M 2.0

Das START OPTION REGISTER muss den Wert 255 enthalten um die EEPROM BOOT OPTION zu deaktivieren
Das START OPTION REGISTER muss den Wert 1 enthalten um die EEPROM BOOT OPTION zu aktivieren

OPTION AUTOSTART

Wenn das entsprechende Bit gesetzt ist, beginnt die Control Unit sofort nach dem Download mit der Abarbeitung des Anwenderprogramms.

EEPROM BOOT OPTION

Die Control Units haben die Möglichkeit das Anwenderprogramm von einem I²C-Datenspeicher, z.B. in Form einer Chipkarte, wie bei einem Programm- Download (Bootvorgang) in den internen Flash-Speicher zu laden.
Dies ist besonders bei Anwendungen interessant bei denen im Feld kein PC für eine Programm Update zur Verfügung steht oder das erforderliche qualifizierte Personal hierfür fehlt.
Passende Speicherkarten und Kartenhalter sind als Zubehör erhältlich.

Ein sicherer Betrieb des Boot Modus ist nur gewährleistet, wenn der ENABLE Pin des LCD mit einem 10K Widerstand nach GND gelegt wird (sofern ein LCD verwendet wird)

BOOT-MODUS EIN/AUSSCHALTEN
Die Option ob dieser Bootvorgang erlaubt wird oder nicht, wird mit einem kleinen Hilfsprogramm dauerhaft in der Control Unit gespeichert.

1. Das Programm SET_BOOTMODE_ON.bas aktiviert die Option
2. Das Programm SET_BOOTMODE_OFF.bas deaktiviert die Option

Laden Sie das entsprechende Programm und starten Sie es. Wenn die ensprechende Meldung am LCD erscheint ist das Programm beendet, und die angezeigte Option gespeichert.

Bei der Auslieferung der Unit ist diese Option aktiviert was zu kurzen Zugriffen auf Port9 und Port10 nach dem Einschalten und nach einem Reset führt. Beachten Sie bitte dass in diesem Fall der Programmablauf eventuell Ihre angeschlossene Hardware beschädigen kann. Schließen Sie keine externe Hardware (außer zur Programmierung) an, bevor Sie ein zu Ihrer Hardware passendes Programm geladen haben. Die durch ein Update installierte OS Version hat den Boot-Modus als default ausgeschaltet.
Wenn Sie diese zusätzliche Boot-Option nicht benötigen oder sicherstellen wollen dass es zu keinen Aktionen an den Ports 9 u. 10 kommt, deaktivieren Sie die Option wie unter 2.beschrieben.
Der Speicher Bereich hinter dem BASIC Programm kann natürlich (auch zur Laufzeit desProgramms) als Datenspeicher genutzt werden.

UNIT M 2.0
Bei einem RESET der Unit versucht das Betriebssystem eine gültige Programmdatei auf einer Chipkarte zu identifizieren und zu laden

UNIT STATION
Um bei einer STATION ein Programm zu laden bedarf es der üblichen Prozedur für einen Download:

- RESET drücken
- PROG drücken
- RESET lösen
- PROG lösen

LADEN VON DER CHIPKARTE:
Diese Option hat Priorität. Wird beim Reset der Control Unit eine CHIPKARTE erkannt (und ist das erste Datenbyte darauf $55, wird das auf der CHIPKARTE befindliche Programm in die Control Unit geladen und danach sofort mit der Ausführung des Programms begonnen.
Eine angeschlossene Chipkarte, die kein gültiges Programm enthält, kann zu völlig unkontrollierten Reaktionen der Unit führen. Beachten Sie dass der maximale Speicherplatz der Unit nicht durch das Anwenderprogramm überschritten wird und dass die Speichergröße der Chipkarte groß genug ist um Ihr Anwenderprogramm aufzunehmen. Empfohlen wird der Betrieb mit den im Zubehör angebotenen Chipkarten.

LADEN VON DER SERIELLEN SCHNITTSTELLE
Nach dem Reset (oder im Programmier-Modus bei der STATION) ist die Control Unit zum Download über die serielle Schnittstelle bereit, wenn:

1) keine CHIPKARTE angeschlossen ist, oder
2) diese zwar angeschlossen ist aber eine Adresse ungleich 160 hat, oder
3) diese angeschlossen iat aber an der ersten Speicherstelle ein Datenbyte ungeleich $55 hat (in diesem Fall geht die Control Unit davon aus, dass die Karte kein gültiges Programm gespeichert hat, also ein z.B. ein reiner Datenspeicher ist.

ERZEUGEN DES BOOT-FILES AUF DEM SPEICHER
Ein Boot-File (also eine Bootfähige Chipkarte) erzeugen Sie indem ein kleines Hilfsprogramm in eine Control Unit M-2.0 geladen wird. Dieses Hilfsprogramm leitet dann den Download (also das Anwender-Programm) den Sie anschließend wie gewohnt durchführen nicht auf den internen Flash-Speicher der Unit M-2.0 sondern auf die Chipkarte um. Schließen Sie das Chipkarten- Modul entweder über das Schnittstellenkabel (Best.-Nr. 198876) oder über den Unit- Bus Steckverbinder an Ihrem System mit einer Unit M-2.0 (Application-Board, Einbaumodul, Station 2.0 mit entsprechender Verkabelung) an.
Achten Sie darauf dass noch keine Chipkarte eingesteckt ist. Starten Sie Ihre Entwicklungsumgebung (IDE). Laden Sie das BASIC Programm MAKE_BOOTFILE.bas

Kompilieren und übertragen Sie das Programm in die Unit bzw. Station. Starten Sie das Programm. Es erscheint folgende Meldung auf dem Display.

MAKE BOOTFILE

INSERT CARD

Stecken Sie nun die Chipkarte in das Chipkarten-Modul. Das Programm wartet etwa 10 s bis es das Programm fortführt und die Speichergösse ermittelt. Es erscheint folgende Meldung auf dem Display.

SIZE: 12345 Bytes

Kurz danach werden sie aufgefordert den Download auszuführen:

START DOWNLOAD

Laden Sie das Programm das Sie auf die Chipkarte übertragen wollen jetzt in Ihre IDE. Compilieren und übertragen Sie das Programm in die Unit bzw. Station. Das geladene Programm in der Unit (MAKE_BOOTFILE) emuliert dabei einen normalen Download, schreibt das Programm aber nicht auf das Flash der Control Unit sondern auf die angeschlossene CHIPKARTE.Sie können auch Files auf die CHIPKARTE schreiben, welche einen Treiber (also ein xyz.S19 File beeinhalten)

Während dem „Download“ wird die Grösse des BASIC-Programms und ein Zähler am LCD angezeigt, der den Download-Fortschritt anzeigt. Es erscheint folgende Meldung auf dem Display.

BAS: 12345 BYTES
WRITING: 123 BYTES

Nach erfolgtem Download erscheint folgende Meldung auf dem Display.

-- READY ----

Ihr BASIC Anwenderprogramm ist nun auf der Chipkarte gespeichert und Sie können es jederzeit durch einstecken in das Chipkarten-Modul ausführen.

Anmerkung:
Wenn Sie dieses Programm (MAKE_BOOTFILE.BAS) selbst von einen Chip-Karte booten wollen, um andere BOOT-Files zu erzeugen, müssen sie die Karte sofort nach der Anzeige "INSERT CARD" entfernen und durch eine Karte ersetzen welche das neue BOOT-File aufnehmen soll.

DAS FLOATING POINT MODUL (FPM)

EINFÜHRUNG

Das Betriebssystem der M ADV Units basiert auf dem Betriebssystem 2.05 einer herkömmlichen M 2.0, ist aber mit einem 32 Bit Floating Point Module im Betriebssystem und einem erweiterten BASIC Programmspeicher (22kB) sowie einem größeren Variablenspeicher ausgestattet. Allerdings gibt es hier eine kleine Einschränkung:
Der Variablenspeicher ist, wie bisher, ohne Einschränkungen bis 140 Bytes zu nutzen. Bis zu 240 Bytes lassen sich nutzen, wenn die Datei Funktion PRINT# x nicht benutzt wird, oder so benutzt wird (z.B. für temporäre Variablen) dass es nicht stört wenn der Variablenbereich 140-240 bei Verwendung von PRINT# gelöscht wird.
Das gilt auch für die Funktion PUT_FLOAT(myFloat) in der FLOATMATH.BLIB. Sie benutzt eine modifizierte Version von PRINT#

Ergänzungen in der Library
Nicht alle Funktionen der Gleitkommaarithmetik sind auf Betriebssystem-Ebene installiert. Ein Teil davon, wie z.B. der LN (der natürliche Logarithmus) sind als Library auf BASIC Ebene installiert, da hier oft die Rechengenauigkeit gegen die Rechenzeit abgewogen wird. Gleiches gilt für die Eingaberoutine von Gleitkommazahlen über Tastatur. Auch hier hat der Anwender die Möglichkeit diese Funktionen seinen individuellen Bedürfnissen anzupassen.

Einschränkungen:
Die Implementierung eine Gleitkommaarithmetik auf einer so kleinen Maschine, wie es die Unit ADV ist, erfordert zwar bei der Funktionalität keine Einschränkungen wohl aber bei der Programmierung selbst.
Gemeint sind hier z.B. grundsätzliche Überlegungen bei der Programmierung von Termen hinsichtlich des Stack-Bedarfes dieser Ausdrücke. Das gilt auch allgemein bei der Mischung verschiedener Datentypen, die oftmals schon vom Compiler oder im Betriebssystem umgewandelt werden (ohne das es der Programmierer merkt), was in anderen Umgebungen komplikationslos ist. Das ist hier nicht möglich und der Programmierer ist selbst dafür verantwortlich die gegebenen Rahmenbedingungen sorgfältig einzuhalten.

Der Compiler gibt hier keine Hinweise bzw. Fehlermeldungen. Der Anwender muss selbst darauf achten. Die Folgen solcher Fehler sind in der Regel unerklärlich falsche Rechenergebnisse und auch Programmabstürze als Folge von Stack-Fehlern.

Interne Darstellung
Während die BASIC Steuercomputer bislang mit den, auch von Laien zu überschauenden Datentypen BYTE und WORD ausgekommen ist, erfordert die Darstellung einer Fliesskommazahl einen weiteren Datentyp FLOAT.
Dieser Datentyp besteht aus einem Byte Exponent und 3 Byte Mantisse mit Vorzeichen, also 32Bit.
Für die interne Darstellung einer Gleitkommazahl gilt folgende Spezifikation:

1) Bias $80
2) Normiert -> Mantissa MSB immer 1
3) Darstellung der Mantissen mit "Mantissa hidden sign bit"

Die Darstellung mit Mantissa hidden sign bit ist leicht verständlich wenn man sich die Darstellung der Zahl +PI und -Pi ansieht:

```
 3.1415927   82 49 0F DB
-3.1415927   82 C9 0F DB
```

Das MSBit im 2.Byte (49 u. C9) ist Vorzeichen der Mantisse Weil wegen der Normierung das MSB immer als 1 angenommen wird, kann man es als Vorzeichen verwenden.

49 0F DB
ist Positiv (MSBit =0) die FP-Routine macht dann daraus C9 0F DB und setzt intern ein positiv FLAG

C9 0F DB
ist Negativ (MSBit =1) die FP-Routine belässt es bei C9 0F DB und setzt intern ein negativ FLAG.

Wertebereich
Der Wertebereich der mit dieser FP-Arithmetic verarbeitet werden kann liegt im Bereich 1 × 10±38.

Genauigkeit
Sie werden feststellen dass sich selbst mit einer 32 Bit FP-Arithmetik simple Zahlen nicht exakt darstellen lassen. So lässt sich z.B. die Zahl 1234 im Rahmen dieser 32Bit nur als 1233.999 darstellen. Direkt damit verbunden sind auch Rechenungenauigkeiten bei grossen Zahlen. 123456.0 + 0.8 führt zum Ergebnis 123456.7, was aber allein auf der endlichen Genauigkeit der 32Bit FP Darstellung beruht.

Das Stack der Control Unit
Das Stack der Control Unit ist ein Speicher der zur kurzzeitigen Ablage von Werten dient. Das können Werte sein, mit denen etwas gerechnet werden soll, oder auch Rücksprungadressen die für ein RETURN (nach einem GOSUB) gebraucht werden. Dabei können Sie sich eine Ablage von einem Wert vorstellen wie die Ablage von einem Holzscheit auf einem Stapel. Das bedeutet, das Holzstück, das zuletzt am Stapel abgelegt wurde muss auch als erstes wieder vom Stapel genommen werden. Und es muss auf jeden Fall vom Stapel genommen werden, weil sonst irgendwann der Stapel an der Decke anstößt (das ist dann der besagte STACKÜBERLAUF)

Die Verarbeitung der Flieskommazahlen erfolgt bei der Control Unit in folgender Weise:

MyFloat=FloatVar1*Floatvar2

Der Compiler erzeugt daraus eine Tokenfolge die folgendes veranlasst:

Lege den Inhalt von FloatVar1 auf das Stack
Lege den Inhalt von Floatvar2 auf das Stack
Lade die beiden letzen Werte von Stack in die beiden Fliesskomma-Akkumulatoren (FPACC)
Multipliziere beide FPACCs
Lade das Ergebnis auf das Stack
Lade den letzten Stackeintrag in die Variable MyFloat

Wenn jetzt der Term zur Berechnung komplexer wird werden sehr viele Einträge im Stack gemacht bevor sie nach und nach zur Berechnung vom Stack geholt werden.

Beispiel: MyFloat=Floatvar3-FloatVar1*(Floatvar2+Floatvar4)

Der Compiler erzeugt daraus eine Tokenfolge die Folgendes veranlasst:

Lege den Inhalt von FloatVar3 auf das Stack
Lege den Inhalt von Floatvar1 auf das Stack
Lege den Inhalt von FloatVar2 auf das Stack
Lege den Inhalt von Floatvar4 auf das Stack
Lade die beiden letzte Werte von Stack in die beiden FPACCs
Addiere beide FPACCs
Lade das Ergebnis auf das Stack
Lade die beiden letzte Werte von Stack in die beiden FPACCs
Multipliziere beide FPACCs
Lade das Ergebnis auf das Stack
Lade die beiden letzte Werte von Stack in die beiden FPACCs
Subtrahiere beide FPACCs
Lade das Ergebnis auf das Stack
Lade den letzten Stackeintrag in die Variable MyFloat

Sie sehen, dass selbst bei einem relativ einfachen Term wie hier, bereits 4 Floats auf das Stack gelegt werden, bevor auch nur eines zur Berechnung vom Stack genommen wird. Das sind bereits 16 Bytes.
Die maximale Stacktiefe (oder im Fall des Vergleichs mit dem Holzstapel: die max. Stapelhöhe) beträgt 46 Bytes, und man darf nicht vergessen dass jede Unterprogrammebene jeweils 2 Bytes für die Rücksprungadresse im Stack belegt.
Bei sehr großen Termen kann es also sinnvoll (und auch meistens übersichtlicher) sein, den Term in 2 oder mehrere Terme aufzuspalten. Damit ist man auf jeden Fall auf der sicheren Seite.

Stacküberläufe äußern sich durch unkontrollierbares Verhalten der Control Unit I und sind nicht immer sofort ersichtlich.

DEKLARATIONEN UND DEFINITIONEN

COMPILER OPTION

Wichtig ist zunächst einmal dem Compiler anzuweisen, dass Fliesskomma Operationen compiliert werden müssen. Das macht "OPTION FLOAT" die am Anfang des Programms stehen muss.
Dann müssen die entsprechenden Variablen als FLAOT definiert werden. Zu beachten ist, dass eine Float-Variable 4 Bytes belegt, sodass man also nicht zu sorglos mit Variablen umgeht, auch wenn man 120 Bytes. Zur Verfügung hat. Die Verteilung der Variablen übernimmt der Compiler selbst. Er sorgt auch dafür dass die ersten 16 Bytes im Variablenspeicher für das FPM reserviert werden.

```
option float
define FV1 as float
define FV2 as float
```

ZUWEISUNGEN

Damit ist bereits alles getan um mit einem kleinen Programm zu beginnen. Dieses beginnt im allgemeinen mit der Zuweisung von Werten an Variablen. Das kann entweder in rein dezimaler Form geschehen oder bei sehr großen und sehr kleinen Zahlen in wissenschaftlicher Notation. Das FPM verarbeitet max. 7 Stellen (weitere Stellen werden ignoriert) und Exponenten bis + - 38

```
FV1=3
FV1=2.3
FV1=0.023
FV1=2.12345678
FV1=exp(1.602,-19)    -> entspricht 1.602E-19
```

Beachten Sie bitte das Format bei Angabe in wissenschaftlicher Notation.

FLOATING POINT EIN/AUSGABE

EINGABEN

Das FPM hat auch eine INPUT Funktion wie sie es bisher auch von den Word Variablen gewohnt sind.

EINGABEFORMAT

Eine Gleitkommazahl auf viele Varianten geschrieben werden so dass man hier drei Vereinbarungen treffen muss:

1) die führende Null vor dem Dezimalpunkt muss angegeben werden: 0.001 (nicht: .001)
2) Der Exponent muss immer zweistellig geschrieben werden: 0.234E03 (nicht: E3)
3) Nach einem Dezimalpunkt muss mindestens eine Stelle folgen 1234.0 (nicht:1234.)

Darüber hinaus ist das Eingabeformat recht flexibel.
Beispiele:

```
0.00000000000000000000000001
1.123456789123456789
12345678912345678912345678
123456789123456789.123456789123456789
123456789123456.7891234E09
-12345678912345.123456789123456789E-12
12.3E12
0.0000000012345E13
```

INPUT MyFloat ;

Eingabe einer Gleitkommazahl über das Terminal, Zuweisung an MyFloat
Eine Eingabe ist abgeschlossen wenn vom Terminal CR oder ein anderes nicht numerisches Zeichen (außer E,e, .) empfangen wird. Wenn Ihr Eingabegerät CR LF am Ende des Strings sendet müssen sie das LF nach erfolgter Eingabe aus dem Zeichenbuffer entfernen (get MyByte).
Ansonsten gelten die gleichen Voraussetzungen wie beim Betrieb der seriellen Schnittstelle üblich:
Es darf kein anderes Objekt (z.B. das LCD) aktiv sein.
Beachten Sie bitte, dass sämtliche Eingaben entsprechend der Formatierung im Betriebssystem (siehe Kapitel "Ausgabeformat") bei der Ausgabe anders erscheinen können.

AUSGABEN

Bei der Ausgabe auf das LCD oder ein Terminal entscheidet wie bisher auch, welches OBJECT gerade aktiv ist. In allen Beispielen wird auf das LCD ausgegeben.

AUSGABEFORMAT

Zahlen größer als 1:
Für eine Ausgabe in dezimaler Form stehen insgesamt 7 Stellen zur Verfügung die auf Vorkomma und Nachkommastellen verteilt werden. Es muss aber mindestens eine Nachkommastelle verbleiben. Wenn dies nicht mehr möglich ist (also bei 7 Stellen vor dem Komma) wird die wissenschaftliche Notation als Darstellung gewählt. Wenn die Zahl positiv ist, wird als erstes Zeichen eine Leerstelle ausgegeben,
sonst ein Minus-Zeichen

Zahlen 0 bis 1:
Sehr kleine Zahlen werden dezimal dargestellt solange sich nur eine Null nach dem Komma befindet. Folgen mehr Nullen, so wird die Zahl in der wissenschaftlichen Notation dargestellt.

12345.123456	**wird dargestellt**	**12345.12**
123456.1234567	**wird dargestellt**	**123456.1**
1234567.1234567	**wird dargestellt**	**1.23456E06**
-1234.1234567	**wird dargestellt**	**-1234.123**
1.234566E03	**wird dargestellt**	**1234.123**
0.0123456	**wird dargestellt**	**0.012345**
0.001234567	**wird dargestellt**	**1.234567E-03**

FPPRINT

Formatierte Ausgabe von Termen oder Variablen

Da eine Gleitkommazahl recht lang sein kann, hat man die Möglichkeit die Anzahl der Stellen nach dem Komma zu begrenzen. Der Dezimalpunkt gilt dabei als eine Stelle. Mit diesem Befehl können sie eine Zahl auch in ihrer vollen Länge ausgeben (7 Stellen hinter dem Komma) oder auch das Komma samt Stellen danach unterdrücken. FPPRINT gibt grundsätzlich auf das aktive Objekt aus. Auch wenn Sie auf das LCD ausgeben bleibt es bei FPPRINT und **nicht** LCD.FPPRINT

FPPRINT (Term, Stellen)

Beispiel für Terme in FPPRINT:

FPPRINT(MyFloat*FLOAT(MyWord),5)
FPPRINT(MyFloat*MyFloat,5)

Anmerkung:
FPPRINT Ignoriert bei der Ausgabe den Verknüpfungsoperator &
FPPRINT (MyVar,3) & "VOLT" ist also nicht möglich kann aber mit einem nachfolgenden LCD.PRINT "VOLT" nachgebildet werden.

PRINT

Standard Ausgabe von Variablen

Ein einfacher PRINT Befehl gibt die Variable mit vier Stellen hinter dem Komma aus und unterscheidet sich in der Anwendung nicht von einem Print für Byte/Wort-Variablen. Bei einer Ausgabe auf das LCD muss es also hier heißen:

```
LCD.PRINT MyVar & "VOLT"
```

Anmerkung:
*Bei der Ausgabe von Gleitkommazahlen mit PRINT ist es nicht möglich einen Term wie z.B. PRINT (MyVar1*MyVar2) auszugeben. Es muss immer eine Variable ausgegeben werden.*
Eingaben über ein Terminal, in der Form wie es mit INPUT für Wortvariablen möglich ist, werden für Floatvariablen nicht unterstützt.

Ein kleines Beispiel zur Zuweisung und Ausgabe:

```
option float
define FV1 as float
define FV2 as float

LCD.INIT

'------ EINGABE IN WISSENSCHAFTLICHER NOTATION -------------------------

FV1=exp(1.23456789,4)  '1.23456789E04

'------ EINGABE IN ÜBLICHER DEZIMALDARSTELLUNG -------------------------

FV2=12345.6789

'------------ AUSGABE MIT VOLLER STELLENZAHL -------------------------------

LCD.POS 1,1
LCD.PRINT FV1 & " VOLT      "
LCD.POS 2,1
LCD.PRINT FV2 & " VOLT      "
```

Starten Sie das Programm FP_INPUT_OUTPUT. Ändern Sie Werte und Formatierung um einen ersten Eindruck zu bekommen.

DATENWANDLUNG FLOAT<-> INTEGER WORD/BYTE

Je nach Anwendung kann es erforderlich sein die beiden Datentypen umzuwandeln. Zu beachten ist dabei, das z.B. die Zahl 691234 natürlich nicht als Wort dargestellt werden kann und das Ergebnis einer solchen Zuweisung oder Umwandlung natürlich fehlerhaft ist. Sehen sie dazu auch das Thema "Floating Point Errors". Das INTEGER muss nicht zwingend eine Variable vom Typ WORD sein. Eine Bytevariable darf es hier auch sein, aber diese hat natürlich einen anderen Wertebereich und trägt kein Vorzeichen.

Grundsätzlich gilt:
Unbedingt erforderlich ist die Umwandlung immer innerhalb eines Terms da hier immer gleiche Datentypen stehen müssen. Der Compiler gibt hier keine Hinweise bzw. Fehlermeldungen. Der Anwender muss selbst darauf achten. Die Folgen solcher Fehler sind in der Regel unerklärlich falsche Rechenergebnisse und auch Programmabstürze als Folge von Stackfehlern.

FLOAT

Wandlung eines Term oder einer Variablen von INTERGER zu FLOAT
Hier sind nahezu alle möglichen Terme zulässig solange man keine Datentypen mischt.

```
MyFloat = FLOAT(MyWord)
MyFloat = MyFloat*FLOAT(MyWord)
MyFloat = MyFloat*FLOAT(MyWord*MyWord)
MyFloat = MyFloat*FLOAT(MyWord*MyByte)
```

NICHT ZULÄSSIG (Mischung von Datentypen) Z.B.

*MyFloat = MyFloat*FLOAT(MyWord)*MyWord*
Führt hier zu falschen Ergebnissen

MyFloat = MyWord
Führt hier zu Stackfehlern und damit zu unvorhersehbarem Verhalten des Programms

INTEGER

Wandlung einer Variablen von FlOAT nach INTEGER
Anders ist das bei der Konvertierung von FLOAT nach INTEGER. Hier kann nur eine Variable gewandelt werden und kein Term.

```
MyWord = INT(MyFloat)
MyWord = 3*INT(MyFloat)/MyWord
MyByte = 3*INT(MyFloat)/MyWord
```

NICHT ZULÄSSIG (Konvertierung eines Terms) Z.B.
MyWord=INT(MyFloat/MyFloat)

FALLEN BEI DER UMWANDLUNG:
Grundsätzlich muss unbedingt darauf geachtet werden, dass der Zahlenbereich der mit einem Wort (oder Byte) dargestellt werden kann nie überschritten wird. Sehen Sie dazu die Beispiele:

Variablen:
MyFloat1=50000
MyFloat2=30000
MyWord=1000

FALL1:
INT_RESULT=INT(MyFloat)
Hier wird einfach nur der Wertebereich eines WORD überschritten

FALL2:
INT_RESULT=INT(MyFloat)/MyWord
Obwohl das Ergebnis (5000) eigentlich nicht zu groß für ein WORD ist, wird der Wertebereich bereits bei der Umwandlung (noch vor der Division)überschritten

FALL 3:
*INT_RESULT=10*INT(MyFloat2)/MyWord*
*Auch hier würde das Ergebnis in ein Word passen, auch 30000 lässt sich ohne Überlauf in ein WORD verwandeln. Hier passiert der Überlauf bei der Multiplikation 10*INT(30000) deren Ergebnis zu groß für ein Word ist.*
*INT_RESULT=INT(MyFloat2)/MyWord*10 liefert dagegen ein richtiges Ergebnis*

Beispiele:

```
option float
define LIGHT as port[16]
define WV1 as word
define WV2 as word
define FV1 as float
define FRESULT as float
define IRESULT as word
LIGHT=off
LCD.INIT


FV1=exp(1.23456789,4)  '1.23456789E03
WV1=1000
'*****************************************************************************************************************
'***      FLOAT CONVERTIERUNG INNERHALB EINES TERMS                                                              ***
'*****************************************************************************************************************
*
'------- TERM MIT FP-VARIABLE UND KONSTANTE -------------------------
FRESULT=FV1*1000
LCD.POS 1,1
LCD.PRINT FRESULT & " VOLT      "
'------- TERM MIT FP-VARIABLE UND WORD-Variable ----------------------
WV1=1000
FRESULT=FV1*FLOAT(WV1*10)                          'Konvertierung WORD-> FLOAT !!
LCD.POS 2,1
LCD.PRINT FRESULT & " VOLT      "
PAUSE 100
'*****************************************************************************************************************
*
'***                  INTEGER CONVERTIERUNG  EINES TERMS
***
'*****************************************************************************************************************
*
' Die Integer Konvertierung eines Terms ist nicht zulässig.
' Es muss immer der Umweg über die Konvertierung einer Variablen
' genommen werden.
'------- TERM MIT FP-VARIABLE UND KONSTANTE -------------------------
FV1=5678.234
IRESULT=INT(FV1)/1000
LCD.POS 1,1
LCD.PRINT IRESULT & " VOLT         "
'------- TERM MIT FP-VARIABLE UND WORD-Variable ----------------------
WV1=1000
IRESULT=3*INT(FV1)/WV1                             'Konvertierung FLOAT-> WORD
LCD.POS 2,1
```

Starten Sie das Programm 2_FP_CONVERTIERUNG. Fügen Sie eigene Terme mit Umwandlungen ein, um zu sehen ob sie das Prinzip verinnerlicht haben

SCHLEIFEN MIT FLOAT VARIABLEN

DO, LOOP UNTIL EXIT DO

Schleifen mit reinen Gleitkommazahlen.

Gleitkommazahlen dürfen in FOR TO NEXT Schleifen nicht verwendet werden. Das macht aber nichts, da man ohnehin besser DO LOOP UNTIL verwendet und hier ist es natürlich möglich Schleifen im vollen Zahlenbereich mit beliebigen Schrittweiten zu konstruieren.

```
DO
. . . . .
. . . . .

LOOP UNTIL (COUNTER>ENDVALUE)
```

Beachten Sie bitte, dass die Endbedingung für die Schleife aus dem Vergleich von Variablen bestehen muss und hier keine Terme oder konstante Werte stehen dürfen.

LOOP UNTIL (COUNTER>123456) ist also nicht zulässig

Eine Schleife kann vorzeitig mit EXIT DO verlassen werden

Beispiel für eine solche Schleife:

```
'************************************************************************
'***          EINE SCHLEIFE MIT FLOAT VARIABLEN       ***
'************************************************************************
COUNTER=123.4567
ENDVALUE=999.567

'---- FOR COUNTER=123.4567 TO 999.567 STEP 0.123 ----

DO
COUNTER=COUNTER+0.123
LCD.POS 1,1
LCD.PRINT COUNTER & "        "
LOOP UNTIL COUNTER>ENDVALUE
```

Schleifen mit gemischten Datentypen
Wenn auch die Schleifenzähler keine FLOAT-Variablen sein dürfen, so kann man doch WORD Variablen oder auch Bytevariablen verwenden und sie nach FLAOT wandeln wenn keine Dezimalzahlen als Schleifenzähler oder Schrittweite erforderlich sind. Wenn man die reine Rechenzeit (ohne LCD Ausgabe) betrachtet stellt man fest, dass diese Schleifen etwa 20% schneller sind als Schleifen mit reinen Floatvariablen.

```
'************************************************************************
'***          EINE SCHLEIFE MIT WORD VARIABLEN       ***
'************************************************************************
FOR LOOPCOUNTER=0 TO 3000
RESULT=FLOAT(LOOPCOUNTER)/10.9
LCD.POS 2,1
LCD.PRINT RESULT & "      "
NEXT
```

Starten Sie das Beispiel 3_FOR_TO_NEXT. Modifizieren sie die Schleifen für andere Wertebereiche.

FUNCTION

FUNCTIONS MIT PARAMETERÜBERGABE

Wie auch bei Bytes und Words als Variablen können Sie an Funktionen auch Floatwerte als Parameter übergeben (auch gemischt)

```
FUNCTION MyFunction(FVAL1 as FLOAT,FVAL2 as FLOAT)
. . . . . .
. . . . . .

END FUNCTION
```

Da gerade bei Floatvariablen der Speicherbedarf recht groß ist (4 Bytes für eine Variable) können lokal angelegte Variablen sehr schnell viel Speicher verbrauchen. Es empfiehlt sich daher die Variablen zu referenzieren. Das bedeutet, verschieden Variablenbezeichner (wie hier INPUT und WERT) nehmen Bezug auf die gleiche (global definierte) Variable. Gerade bei komplexen Programmen lassen sich somit oft viele Variablen sparen.

```
define FVALUE as FLOAT

FUNCTION ABC(INPUT ref FVALUE)
RESULT=RESULT*INPUT
END FUNCTION

FUNCTION XYZ(WERT ref FVALUE)
RESULT=RESULT*WERT
END FUNCTION
```

FUNCTIONS MIT PARAMETER RÜCKGABE

Eine Funktion kann auch einen Float Wert zurückgeben. Dazu wird in der Funktion einer Hilfsvariablen ein Wert (z.B. das Ergebnis einer Rechnung) zugewiesen. Diese Hilfsvariable muss nicht deklariert werden, muss aber den Namen der Funktion haben. Die Rückgabe muss immer in eine deklarierte Float-Variable erfolgen.

```
FUNCTION MyFunction(FVAL1 as FLOAT,FVAL2 as FLOAT)
. . . . . .
MyFunction=FVAL1*FVAL2
. . . . . .
END FUNCTION


'------------ FUNCTION AUFRUF MIT RÜCKGABE -----------------------
FLOATVAR=MyFunction(10.9, 33.0)
```

Starten Sie das Beispiel 4_FUNCTION_1 und 2 und erweitern Sie es.

MATHEMATISCHE OPERATIONEN

Das Control Unit FLOATING POINT MODULE (FPM) das im Betriebssystem integriert ist kann neben den Umrechnungen der Datentypen FLOAT,BYTE und INTEGER auch die Grundrechenarten MULTIPLY, DIVIDE, SUBTRACT und ADD. Daneben ist auch die Wurzelberechnung SQRT, Sinusberechnung SIN und COS sowie ABS und Vergleichsoperationen möglich.

ÜBERSICHT

```
MULTIPLY
DIVIDE
ADD
SUBTRACT
SQRT
SIN
COS
ABS
<, >, =, <=, >=
```

Darüber hinaus gibt es natürlich noch andere wichtige Funktionen die in einer separaten Library FLOATMATH.BLIB zur Verfügung stehen. Die Beispiele in diesem Kapitel zeigen einige einfache Anwendungen der Grundrechenarten. Zur Anwendung der FLOATMATH.BLIB sehen Sie sich bitte das Programm EXTENDED_FLOAT_MATH.bas und das zugehörige Kapitel in diesem Dokument an.

Laden Sie das Programm STANDARD_FLOAT_MATH.BAS und probieren sie die verschiedenen Funktionen aus.

GRUNDRECHENARTEN

Bildung von Termen mit den Grundrechenarten

Zur Demo der Multiplikation bietet es sich an den Analogwert eines ADC in den richtigen Spannungswert zu wandeln. Das Ergebnis wird mit 4 Stellen hinter dem Komma ausgegeben. Beachten Sie dass hier eine Konvertierung von Byte nach Float erforderlich ist, da der AD-Wert ja im Byte-Format vorliegt.
Daneben sehen sie wie aufwändig die gleiche Funktion ohne Flieskomma ausgeführt werden muss.
Die anderen Rechenoperationen sind äquivalent zu formulieren und benötigen wohl kein eigenes Beispiel.

```
DO
VOLTS=FLOAT(ADC8)*0.0196
LCD.POS 1,1
LCD.PRINT VOLTS & "V      "
LOOP
```

```
DO
MILLIVOLT=98*ADC8/5
VOLT=MILLIVOLT/1000
NACHKOMMA=MILLIVOLT MOD 1000
LCD.POS 1,1
LCD.PRINT "ADC8: "& VOLT & "."
if NACHKOMMA<100 then LCD.PRINT "0"
LCD.PRINT NACHKOMMA & " V    "
LOOP
```

SIN, COS

Berechnet den Sinus/Cosinus einer Variablen oder konstanten Wertes (Grad oder Bogenmaß)

Genauigkeit
Die Berechnung des SINUS erfolgt aus einer Taylor-Reihe und ist im Sinne einer Optimierung der Rechenzeit eine (wenn auch recht gute) Näherung beschränk. Während die Werte für Winkel im Bereich 0 bis 90° auf mehr als 5 Stellen genau sind, nimmt die Genauigkeit darüber zunehmend ab.
Bei 130° ist die Genauigkeit auf 3 Stellen und bei 179° auf ca. 2 Stellen nach dem Komma gesunken.
Sie können aber die Genauigkeit verbessern indem sie das Glied **x^11/11!** und die folgenden Glieder der Taylorreihe bis zur gewünschten Genauigkeit zum Ergebnis addieren

Definitionsbereich
Außerdem ist die SINUS Berechnung auf einen Winkelbereich von -180°...+180° beschränkt Größere oder kleinere Winkel müssen auf diesen Bereich abgebildet werden, wenn sie verarbeitet werden sollen. Das gilt auch für COSINUS der mit COS(x)=1-SIN(x) berechnet wird.

SIN kann keinen Term berechnen. Es ist entweder ein konstanter Winkel oder eine Gleitkommavariable als Berechnungsgrundlage erforderlich. **BPPDEGREES** gibt an, dass der Winkel in Grad angegeben ist.
Alternativ kann der Winkel auch als **BPPRADIANS** also im Bogenmaß (-3.14.....+3.14) angegeben werden.

```
MyFloat=SIN(ANGLE,bppdegrees)
MyFloat=SIN(30.33,bppdegrees)
```

```
MyFloat=COS(ANGLE,bppdegrees)
MyFloat=COS(30.33,bppdegrees)
```

SQRT

Berechnet die Quadrat-Wurzel einer Variablen oder eines konstanten Wertes

Genauigkeit
Die Berechnung der Wurzel erfolgt mit einer Näherung und ist im Sinne einer Optimierung der Rechenzeit eine sehr gute Näherung mit einer Genauigkeit von mindestens 5 Stellen hinter dem Komma.

SQRT kann keinen Term berechnen. Es ist entweder ein konstanter Wert oder eine Gleitkommavariable als Berechnungsgrundlage erforderlich.

```
MyFloat=SQRT(VALUE)
MyFloat=SQRT(12.345)
```

Es gibt in der FLOATMATH.blib auch eine Funktion um die n-te Wurzel aus einer Zahl zu ziehen. Für eine Quadratwurzel sollten Sie aber auf jeden Fall SQRT benutzen.

ABS

Berechnet den Absolutwert eines Terms oder einer Variablen.

```
MyFloat=ABS(FLOAT(ADC8*MyByte)*0.0196+MyFloat*MyFloat)
MyFloat= ABS(FLOAT(MyWord-MyByte*MyWord))/ABS(MyFloat+MyFloat/5)
```

COMPARES

Vergleichen von zwei Variablen

Als Compares sind die Vergleichsoperationen **<, >, =, <=, >=** zusammengefasst.
Ein Compare muss sich immer auf zwei Variablen beziehen.

```
IF MyFloat1>MyFloat2 THEN.....
IF FLOAT(MyWord) > MyFloat1 THEN.....
LOOP UNTIL  MyFloat1>MyFloat2
LOOP UNTIL FLOAT(MyWord) > MyFloat1
IF NOT(MyFloat1=MyFloat2) THEN.....
```

Wie sie vielleicht gemerkt haben fehlt der <> Operator. Er wird ersetzt durch:
IF NOT(MyFloat1=MyFloat2) THEN.....

FLOATING POINT ERRORS

Verbotene Rechenoperationen

Obwohl die Berechnungen so ausgelegt sein müssen, dass keine verbotenen Operationen durchgeführt werden (z.B. Division durch Null) kann es im Betrieb z.B. beim Ausfall eines Sensors einer Regelung zu diesen Situationen kommen. Da Rechnungsergebnisse in solchen Fällen durchaus plausibel erscheinen können obwohl sie total falsch sind, haben Sie die Möglichkeit eine Fehlerkontrolle zu implementieren und im Fall eines Fehlers gezielt darauf zu reagieren.

ON ERROR GOTO

Fügt in den compilierten Programmcode selbstständig eine Abfrage auf Fehler ein und verzweigt im Falle eines Fehlers auf Ihre Fehlerbehandlungs-Routine. Um die Art des Fehlers zu identifizieren beinhaltet die Variable **ERR.NUMBER** einen Fehlercode. Der Fehlercode wird vor der nächsten Gleitkommaoperation selbstständig gelöscht.

```
0  No error
1  Overflow error
2  Underflow error
3  Division by Zero error
4  Square root of negative number
5  Number too large/small to convert to integer
```

Dieses kleine Beispiel demonstriert die Anwendung. Es erzwingt durch fortgesetzte Multiplikation einen Überlauf. Der Fehler wird angezeigt und der Überlauf in der Variablen RESULT beseitigt.

```
FUNCTION FPE()
ON ERROR GOTO ER
#x
RESULT=1
DO
RESULT=RESULT*100
LCD.POS 1,1
LCD.PRINT RESULT & "            "
pause 20
LOOP
'-------- DISPLAY AN ERROR ----------
#ER
LCD.POS 2,1
LCD.PRINT "ERROR:" & ERR.NUMBER
BEEP 5,5,50
LCD.POS 2,1
LCD.PRINT "                "
goto x
END FUNCTION
```

DIE FLAOTING POINT MATH BASIC++ LIBRARY

Das Control Unit FLOATING POINT MODULE (FPM), das im Betriebssystem integriert ist, kann neben den Umrechnungen der Datentypen FLAOT,BYTE und WORD auch die Grundrechenarten MULTIP, DIVIDE, SUBTRACT und ADD. Daneben ist auch die Wurzelberechnung SQR und Sinusberechnung SIN und COS verfügbar. Auch Vergleichsoperationen sind möglich Einige oft benötigte Funktionen sind darin allerdings nicht enthalten, werden aber in der **FLOATMATH.BLIB** zur als Funktionen zur Verfügung gestellt. Die Library selbst benötigt einige Variablen die dort auch bereits definiert sind:

```
define BVALUE1 as byte
define BVALUE2 as word
define FVALUE1 as float
define FVALUE2 as float
define FVALUE3 as float
define FVALUE4 as float
define FVALUE5 as float
define FVALUE6 as float
define RESULT ref FVALUE2
```

Sie dürfen nicht doppelt definiert werden, können aber referenziert werden. Die Floats FVALUE5 und FVALUE6 werden nur für die FUNCTION PWR(x,y) benötigt und können zusammen mit der Funktion gelöscht werden wenn sie nicht benötigt werden.

Die Library stellt Ihnen folgende Funktionen zur Verfügung:

LN(x)	
LOG(x)	
TAN(x) (x in degrees)	
EXPO(x,y)	x^y (y as Byte)
POWER(x,y)	x^y (y as FLOAT)
E(x)	exp(x),e^x (e,x as FLOAT)
Nth_ROOT(x,y)	y-te Wurzel aus x
ARCSIN(x)	Umkehrfunktion des Sinus (x as FLOAT)
ARCCOS(x)	Umkehrfunktion des Cosinus (x as FLOAT)

Alle Funktionen werden mit einem oder mehreren Parametern aufgerufen und liefern das Ergebnis in der Flaotvariablen RESULT zurück. z.B. **LOG(MyFloat)**

Ab der Library Version 1.4 sind zusätzlich Zuweisungen möglich: **MyFloat=LN(MyFloat)**
Damit lassen sich auch und auch kompiziertere Terme formulieren **FV1=LOG(POWER(3,nth_ROOT(27,3)))**
Das Arbeiten mit dem Rechenergebnis jeweils in RESULT dürfte allerdings bedeutend übersichtlicher sein, als solche Terme

Sehen sie sich dazu bitte die Beispiele im Programm 7_EXTENDED_MATH.BAS an. Sie zeigen Gebrauch der Funktionen und Parameterübergabe.

LN(MyFloat)

Natürlicher Logarithmus einer Variablen oder eines konstanten Wertes

Er wird aus einer Reihe entwickelt und mit einer Genauigkeit von mindestens 4 Stellen hinter dem Komma berechnet. Bei höheren Ansprüchen bez. der Präzision können sie die FUNCTION LN entsprechend modifizieren. Ändern sie dort den Schleifenendwert 9 auf eine höhere ungerade Zahl z.B. 13
FOR BVALUE1=3 TO 9 STEP 2

```
LN(MyFloat)
LN(12.345)
LN((FLOAT(MyWord))
Das Ergebnis steht dann in der Variablen RESULT
```

LOG(MyFloat)

Dekadischer Logarithmus einer Variablen oder eines konstanten Wertes
Er wird aus dem LN entwickelt und hat deshalb die gleiche Präzision. Modifizieren Sie die FUNCTION LN wie beschrieben, wenn sie eine höhere Präzision benötigen.

```
LOG(MyFloat)
LOG(12.345)
LOG((FLOAT(MyWord))
Das Ergebnis steht dann in der Variablen RESULT
```

EXPO(MyFloatX,MyWordY)

Exponent Y zur Basis X einer Variablen oder eines konstanten Wertes

```
EXPO(MyFloat, MyWord)
EXPO(12.345,3)
EXPO(MyFloat,3)
Das Ergebnis steht dann in der Variablen RESULT
```

Der Algorithmus beruht auf einer fortgesetzten Multiplikation und lässt als Exponenten nur geradzahlige Werte in Byte Grösse (aber mit Vorzeichen) zu. Er ist wesentlich schneller als POWER.

POWER(MyFloatX,MyFLOATY)

Exponent Y zur Basis X einer Variablen oder eines konstanten Wertes

```
POWER(MyFloatx, MyFloaty)
POWER(12.345,3.33)
POWER(MyFloat,3.45)
Das Ergebnis steht dann in der Variablen RESULT
```

Die Berechnung beruht auf x^y = e ^ (y * LN x) und damit aus zwei Reihenentwicklung.

Die Genauigkeit kann verbessert werden wenn die Anzahl der Iterationen bei der Reihenentwicklung vergrössert wird.
Ändern Sie dazu die Zeile in der Library entsprechend ab: BVALUE1=30 'ITERATIONS
(in der Function E)

E(MyFloatX)

Exponent X zur Basis e

```
E(MyFloatx)
E(12.345)
Das Ergebnis steht dann in der Variablen RESULT
```

Die Berechnung beruht auf einer Reihenentwicklung. Die Berechnung ist im Bereich -5 bis 15 für den Exponenten auf mindestens 4 Stellen hinter dem Dezimalpunkt genau, wird aber ausserhalb dieses Bereichs zunehmend ungenau.
Die Genauigkeit kann verbessert werden wenn die Anzahl der Iterationen bei der Reihenentwicklung vergrössert wird.

Ändern Sie dazu die Zeile in der Library entsprechend ab: BVALUE1=30 'ITERATIONS

Nth_ROOT(MyFloatX,MYByteY)

Berechnet die n-te Wurzel y aus der Zahl x

```
Nth_ROOT(MyFloat, Mybyte)
Nth_ROOT(12.345,3)
Nth_ROOT(MyFloat,3)
Das Ergebnis steht dann in der Variablen RESULT
```

FAK(MyByte)

Berechnet die Fakultät einer Zahl

```
FAK(MyByte)
FAK(12)
FAK((INT(MyFLOAT))
Das Ergebnis steht dann in der Variablen RESULT
```

TAN(MyFloatX)

Tangens einer Variablen oder eines konstanten Wertes (x=degrees)
Der Tangens wird aus dem Sinus berechnet und hat bei 89° einen Fehler von 0.5 Bei kleineren Winkeln nimmt die Genauigkeit sehr schnell zu. Bereits bei 80° ist der berechnete Wert auf 3 Stellen hinter dem Komma genau. Der Grund dafür ist die begrenzte Genauigkeit der 32 Bit Rechnung welche im Bereich grosser Funktionssteigungen (und diese ist bei 90 Grad unendlich) natürlich zu grossen Fehlern führt.

```
TAN(MyFloat)
TAN(12.345)
Das Ergebnis steht dann in der Variablen RESULT
```

ARCSIN(MyFloatX) ARCCOS(MyFloatX)

Die Umkehrfunktionen von Sinus und Cosinus werden aus einer Reihe berechnet. Sie liefern Grad zurück und werden ab einem Wert von 0.9 (hier sind noch mehrere Stellen nach dem Komma genau) zunehmend ungenau. 0.9999 weisen bereits mehrere zehntel Grad Abweichung auf. Der Grund dafür ist die begrenzte Genauigkeit der 32 Bit Rechnung welche im Bereich grosser Funktionssteigungen (und diese ist bei 1.0 unendlich) natürlich zu grossen Fehlern führt.
Die Iteration wird dynamisch auf den Übergabeparameter abgestimmt und liegt bei Werten >0.99 bei 500ms, sonst zwischen 10 und 250ms

```
ARCSIN(MyFloat)
ARCSIN(0.5)
Das Ergebnis steht dann in der Variablen RESULT
```

ARCTAN(MyFloatX) ARCCOT(MyFloatX)

Die Umkehrfunktionen von Tangens und Cotangens werden mit einer Näherung berechnet. Sie liefern Grad zurück. Im Bereich |x|<1 ist die Berechnung auf mehrere Stellen hinter dem Komma genau.
Für |x|>1ist die Genauigkeit besser als eine Stelle hinter dem Komma.

```
ARCTAN(MyFloat)
ARCCOT(0.5)
Das Ergebnis steht dann in der Variablen RESULT
```

DIE FLOATING POINT TOOLS LIBRARY

Diese Library stellt ihnen einige Hilfsmittel zur Verfügung die oft benötigt werden und für den Anfänger nur schwer selbst zu programmieren sind. Die Programme in der Library selbst benutzen oftmals spezielle Token da die zugehörigen Schlüsselwörter normal uninteressant und deshalb auch dem Compiler nicht bekannt sind.

GET_FPVALUE()

Eingabe einer FP-Variablen von Tastatur
Dieser Aufruf der Funktion GET_FPVALUE weit der FP-Variablen RESULT1 einen Wert zu der über die Tastatur am Application-Board eingegeben werden kann. Die Tasten 0 bis 9 geben den entsprechenden numerischen Wert ein. Die Taste F2 erzeugt ein Minuszeichen, die Taste F1 den Dezimalpunkt und wenn der Dezimalpunkt gesetzt ist, das Zeichen E für den Exponenten. Das heißt wenn eine Zahl mit Exponenten dargestellt werden soll muss die Mantisse einen Dezimalpunkt haben.
Beispiel:

-123E-02

Eingabe: <F2> <1> <2> <3> <F1> <0> <F1> <F2> <0> <1> <E> entspricht -123.0E-02

Auch hier ist das Eingabeformat recht flexibel, aber es gibt einige Regeln zu beachten:

1) die führende Null vor dem Dezimalpunkt muss angegeben werden: 0.001 (nicht: .001)
2) Der Exponent muss immer zweistellig geschrieben werden: 0.234E03 (nicht: E3)
3) Nach einem Dezimalpunkt muss mindestens eine Stelle folgen: 1234.0 (nicht:1234.)
4) Die Gesamtzahl aller Zeichen (inkl. MINUS,E und Punkt) darf nicht mehr als 15 sein, was aber bei richtigem Gebrauch die Genauigkeit nicht einschränkt

Beispiele

0.0000000000001
1.1234567891234
123456789123456
123.12345678912
3456.7891234E09
-12345678.1E-12
12.3E12
-1234.56789E-01

Eine Eingabe ist abgeschlossen wenn die Taste E gedrückt wird. Die Taste C löscht eine ganze Eingabe. Das Löschen einzelner Digits ist nicht möglich.
Beachten Sie bitte, dass sämtliche Eingaben entsprechend der Formatierung im Betriebssystem (siehe Kapitel "Ausgabeformat") bei der Ausgabe anders erscheinen können.

Starten Sie das Programm GET_FPVALUE.BAS um ein Beispiel zur Anwendung zu sehen.

Achtung:
Diese Funktion verwendet den Zeichenbuffer der seriellen Schnittstelle. während der Tastatureingabe muss daher sichergestellt werden, dass von der seriellen Schnittstelle keine Daten empfangen werden.

PUTFLOAT(MyFloat)

Datenspeicherung von Float-Variablen im Flash (entspricht PRINT#)

Die Control Unit hat die Möglichkeit Daten auf den verbleibenden freien Speicher im Flash zu speichern. Dafür sind ursprünglich die Befehle PRINT# und INPUT# vorgesehen. Ein Float jedoch benötigt aber 2 Worte
zur Darstellung so dass sich PRINT#/INPUT# nicht ohne Modifikation verwenden lassen. Diese modifizierten Versionen stehen jetzt für Floats als Funktionen in der Library bereit.

GETFLOAT()

Float-Variablen aus dem Flash lesen (entspricht INPUT#)
Der gelesene FP Wert steht nach Aufruf in der Variablen RESULT1

Starten sie das Programm FP_DATENSPEICHERUNG.BAS um ein Beispiel zur Anwendung zu sehen.

ANHANG ZUM FPM

FPM QUICKGUIDE

Fliesskomma Eingabeformate (Variablen)

```
FV1=3
FV1=2.3
FV1=0.023
FV1=2.12345678
FV1=exp(1.602,-19)    -              > entspricht 1.602E-19
```

Fliesskomma Eingabeformate (INPUT MyFloat)

1) die führende Null vor dem Dezimalpunkt muss angegeben werden: **0.001 (nicht: .001)**
2) Der Exponent muss immer zweistellig geschrieben werden: **0.234E03 (nicht: E3)**
3) Nach einem Dezimalpunkt muss mindestens eine Stelle folgen: **1234.0 (nicht:1234.)**

```
0.00000000000000000000000001
1.123456789123456789
12345678912345678912345678
123456789123456789.123456789123456789
123456789123456.7891234E09
-12345678912345.123456789123456789E-12
12.3E12
0.0000000012345E13
```

Fliesskomma Ausgabeformate

12345.123456	wird dargestellt	12345.12
123456.1234567	wird dargestellt	123456.1
1234567.1234567	wird dargestellt	1.23456E06
-1234.1234567	wird dargestellt	-1234.123
1.234566E03	wird dargestellt	1234.123
0.0123456	wird dargestellt	0.012345
0.001234567	wird dargestellt	1.234567E-03

Ausgabe Anweisungen

FPPRINT (Term oder Variable,Stellenzahl)
- Formatierte Ausgabe von Termen oder Variablen

```
FPPRINT(MyFloat*FLOAT(MyWord),5)
FPPRINT(MyFloat*MyFloat,5)
FPPRINT(MyFloat,3)
```

PRINT
- Standard Ausgabe von Variablen

```
LCD.PRINT MyFloat & "VOLT"
```

Wandlung von Datenformaten

FLOAT(Term oder Variable)
- Wandlung eines Term oder einer Variable von INTERGER zu FLOAT

```
MyFloat = FLOAT(MyWord)
MyFloat = MyFloat*FLOAT(MyWord)
MyFloat = MyFloat*FLOAT(MyWord*MyWord)
MyFloat = MyFloat*FLOAT(MyWord*MyByte)
```

INT(Variable)
- Wandlung einer Variablen von FLOAT nach INTEGER

```
MyWord = INT(MyFloat)
MyWord = 3*INT(MyFloat)/MyWord
MyByte = 3*INT(MyFloat)/MyWord
```

Schleifen mit reinen Float Variablen

```
COUNTER=123.4567
ENDVALUE=999.567

'---- FOR COUNTER=123.4567 TO 999.567 STEP 0.123 ----

DO
COUNTER=COUNTER+0.123
LCD.POS 1,1
LCD.PRINT COUNTER & "         "
LOOP UNTIL (COUNTER>ENDVALUE)
```

Schleifen mit gemischten Datentypen

```
FOR LOOPCOUNTER=0 TO 3000
RESULT=FLOAT(LOOPCOUNTER)/10.9
LCD.POS 2,1
LCD.PRINT RESULT & "      "
NEXT
```

FUNCTIONS mit Parameterübergabe und Rückgabe

```
FUNCTION MyFunction(FVAL1 as FLOAT,FVAL2 as FLOAT)
. . . . . .
MyFunction=FVAL1*FVAL2
. . . . . .
END FUNCTION


FLOATVAR=MyFunction(10.9, 33.0)
```

Mathematische Operationen

MULTIPLY
DIVIDE
ADD
SUBTRACT
SQRT
SIN
COS
ABS
<, >, =, <=, >=

MULTIPLY, DIVIDE, ADD, SUBTRACT

- Bildung von Termen mit den Grundrechenarten

```
MyFloat=MyFloat/FLOAT(ADC8*MyByte)*0.0196+MyFloat*MyFloat
MyFloat= FLOAT(MyWord-MyByte*MyWord)/(MyFloat+MyFloat/5)
```

SIN, COS

- Sinus/Cosinus einer Variablen oder konstanten Wertes (Grad oder Bogenmaß)

```
MyFloat=SIN(ANGLE,bppdegrees)
MyFloat=SIN(30.33,bppdegrees)
```

```
MyFloat=COS(ANGLE,bppdegrees)
MyFloat=COS(30.33,bppdegrees)
```

SQRT

- Quadrat-Wurzel einer Variablen oder eines konstanten Wertes

```
MyFloat=SQRT(VALUE)
MyFloat=SQRT(12.345)
```

ABS

- Absolutwert eines Terms oder einer Variablen

```
MyFloat=ABS(FLOAT(ADC8*MyByte)*0.0196+MyFloat*MyFloat)
MyFloat= ABS(FLOAT(MyWord-MyByte*MyWord))/ABS(MyFloat+MyFloat/5)
```

COMPARES <, >, =, <=, >=

- Vergleichen von zwei Variablen

```
IF MyFloat1>MyFloat2 THEN.....
IF FLOAT(MyWord) > MyFloat1 THEN.....
LOOP UNTIL MyFloat1>MyFloat2
LOOP UNTIL FLOAT(MyWord) > MyFloat1
IF NOT(MyFloat1=MyFloat2) THEN.....
```

DIE FLAOTING POINT MATH BASIC++ LIBRARY

Alle Funktionen werden mit einem oder mehreren Parametern aufgerufen und liefern das Ergebnis in der Flaotvariablen RESULT zurück. z.B. **LOG(MyFloat)**

Ab der Library Version 1.4 sind zusätzlich Zuweisungen möglich: **MyFloat=LN(MyFloat)**
Damit lassen sich auch und auch kompiziertere Terme formulieren **FV1=LOG(POWER(3,nth_ROOT(27,3)))**

LN(MyFloat)

- Natürlicher Logarithmus einer Variablen oder eines konstanten Wertes

LN(MyFloat)
LN(12.345)
LN((FLOAT(MyWord))
Das Ergebnis steht dann in der Variablen RESULT

LOG(MyFloat)

- Dekadischer Logarithmus einer Variablen oder eines konstanten Wertes

Er wird aus dem LN entwickelt und hat deshalb die gleiche Präzision. Modifizieren Sie die FUNCTION LN wie beschrieben, wenn sie eine höhere Präzision benötigen.

LOG(MyFloat)
LOG(12.345)
LOG((FLOAT(MyWord))
Das Ergebnis steht dann in der Variablen RESULT

EXPO(MyFloatX,MyByteY)

- Exponent Y zur Basis X einer Variablen oder eines konstanten Wertes

EXP(MyFloat, Mybyte)
EXP(12.345,3)
EXP(MyFloat,3)
Das Ergebnis steht dann in der Variablen RESULT

TAN(x)

- Tangens einer Variablen oder eines konstanten Wertes (x=degrees)

TAN(MyFloat)
TAN(12.345)
Das Ergebnis steht dann in der Variablen RESULT

ARCSIN(MyFloatX) ARCCOS(MyFloatX)

- Die Umkehrfunktionen von Sinus und Cosinus

```
ARCSIN(MyFloat)
ARCSIN(0.5)
Das Ergebnis steht dann in der Variablen RESULT
```

ARCTAN(MyFloatX) ARCCOT(MyFloatX)

Die Umkehrfunktionen von Tangens und Cotangens

```
ARCTAN(MyFloat)
ARCCOT(0.5)
Das Ergebnis steht dann in der Variablen RESULT
```

DIE FLOATING POINT TOOLS LIBRARY

GET_FPVALUE()

- Eingabe einer FP-Variablen von Tastatur - Wert in RESULT1

-123E-01
Eingabe: <F2> <1> <2> <3> <F1> <0> <F1> <F2> <0> <1> <E> entspricht -123.0E-01

F1= DEZIMAPUNKT/EXPONENT
F2= MINUS
E = ENTER
C= EINGABE LÖSCHEN

1) die führende Null vor dem Dezimalpunkt muss angegeben werden: 0.001 (nicht: .001)
2) Der Exponent muss immer zweistellig geschrieben werden: 0.234E03 (nicht: E3)
3) Nach einem Dezimalpunkt muss mindestens eine Stelle folgen: 1234.0 (nicht:1234.)
4) Die Gesamtzahl aller Zeichen (inkl. MINUS,E und Punkt) darf nicht mehr als 15 sein, was aber bei richtigem Gebrauch die Genauigkeit nicht einschränkt

PUTFLOAT(MyFloat)

- Datenspeicherung von Float-Variablen im Flash (entspricht PRINT#)

GETFLOAT()

- Float-Variablen aus dem Flash lesen (entspricht INPUT#) - Wert in RESULT1

DAS VOICE AND SOUND MODUL

Ab den OS Versionen 2.28 steht in der ADVANCED Ausführung der BASIC Computer ein Soundmodul zur Verfügung. Es beinhaltet einen File Player um eigene aufgezeichnete Sound Dateien abzuspielen und einen Text To Speech Synthesizer der eine PRINT Anweisungen von Text oder Variableninhalten als Sprache ausgibt. Für den Betrieb ist ein externes EEPROM als Datenspeicher erforderlich. Vorzugsweise ein 24LC 256 mit 64 Kbyte Speichervermögen. Es sind bis zu 8x64 KByte EEPROM adressierbar, so dass man bis zu 60s gespeicherte Klang Files abrufen kann (der Datenstrom ist 8 kByte/s). Es können bis zu 256 verschiedene Files auf einem EEPROM direkt abgespielt werden.
Die Ausgabe erfolgt über den DA1 Anschluss. Es ist ein 26 kHz PWM Signal mit einem Pegel von 5V und kann direkt an einen kleinen 8 Ohm Lautsprecher angeschlossen werden. Wenn man dem Lautsprecher noch etwas Resonanzraum bewilligt, so ist die Lautstärke für den Betrieb in ruhiger häuslicher Umgebung bereits ausreichend.

Ältere OS Versionen können ohne Einschränkung auf die OS Version 2.28 (Station ADV) und 2.29 Unit M ADV aktualisiert werden:
http://www.spiketronics.com/downloads/UPDATE_228_229.zip

ACHTUNG:
Wenn ein Audioverstärker angeschlossen wird, muss das 26Khz Signal unbedingt soweit unterdrückt werden, dass ein angeschlossenern Hochtonlautsprecher nicht überlastet wird. Die 26 kHz sind unhörbar, können aber den Verstärker (je nach Ausführung) voll aussteuern und damit zur Zerstörung von angeschlossenen Lautsprechern führen.

TEXT TO SPEECH SYNTHESIZER

Mit dem TTSS lassen sich alle Ausgaben die mit PRINT Anweisungen auf das LCD oder die Schnittstelle geschrieben werden können auch direkt als Sprache ausgeben.

DIE PHONEM DATEI AUF DAS EEPROM LADEN

Die Sprachausgabe benötigt die Buchstaben als Stimmsamples auf einem externen 64 kByte EEPROM. Im VOICE AND SOUNDPACK finden Sie neben anderen Tools auch ein Laderprogramm um die Stimmsamples (PHONEM_FILE.TXT) auf das EEPROM zu laden. Laden Sie das Programm LOAD_PHONEM_FILE.BAS in den BASIC-Computer und starten sie es. Öffnen Sie das Terminal in der Workbench, wählen Sie 1200Bd als Baudrate und „Textdatei senden“
Wählen Sie die Datei PHONEM_FILE.TXT aus, und klicken Sie „Datei senden“. Das Programm zeigt die aktuell auf das EEPROM geschrieben Anzahl von Bytes an. Der Ladevorgang ist nach etwa 55.000 Bytes beendet. Das EEPROM muss dafür auf 160 (dez) adressiert sein. Das TTSS ist funktionsbereit, wenn das EEPROM geladen wurde.

INITIALISIERUNG

Das TTSS ist als default auf die Adresse 160 (dez) eingestellt, kann aber mit

```
VOICEBASE=174
```

auf jede beliebige Adresse eingestellt werden (hier z.B. 174) Es ist die EEPROM WRITE Adresse und ist immer geradzahlig. Stellen Sie VOICEBASE auf die Adresse des EEPROMS welches das Phonem File enthält.

Die PLM DA-Converter (DAC) laufen mit 2 Khz Repeat Rate zu langsam für das Soundmodul. Stellen Sie das CONFIG2 Register auf die Option VOICE MODE (26 kHz). Damit wird auch der DA2 mit 26kHz betrieben, was aber normal nicht störend ist. Er kann weiter wie gewohnt benutzt werden

```
CONFIG2.INIT
CONFIG2.PUT 00010000b
CONFIG2.OFF
```

Bit 4 in CONFIG2 aktiviert den VOICE MODE

SPRACHAUSGABE VON TEXTEN

Für die Sprachausgabe von Texten wird prinzipiell jeder Buchstabe eines Textes nacheinander als Laut ausgegeben.

SAY „MY TEXT IS HERE“

Das Programm wird für die Dauer der Sprachausgabe angehalten, Interrupts werden aber weiterhin bedient. Während einer Sprach oder Soundausgabe darf kein Interrupt ausgeführt werden der auf die Schnittstelle oder das LCD schreibt.

Die Laute sind im EEPROM als digitale Stimmsamples gespeichert. Es sind alle Buchstaben a bis z hinterlegt.

a b c d e f g h i j k l m n o p q r s t u v w x y z

Zusätzlich zu den Buchstaben a bis z sind manche Buchstaben als eine gross geschriebene Variante hinterlegt, da sie anders klingen. So muss z.B. das „e“ in „dank**e**“ anders klingen als das „e“ in B**e**ben.

A E I O U M N Ä Ö Ü

Diese Laute lassen sich in der Regel auch vervielfachen, was bei a bis z nur bedingt möglich ist.

Um die Verständlichkeit zu erhöhen sind Laute zusammengefasst die sich nicht reibungslos aneinander reihen lassen. Sie werden alle gross geschrieben.

AU EE EI IE ER EU CH SCH

Um die Aussprache soweit als mit einem solch kleinen System möglich an den geschriebenen Text anzupassen, werden vom TTSS manche Laute auch (für den Anwender unsichtbar) im Hintergrund gewandelt. Wie z.B. „qu“ das ja wie „kw“ gesprochen wird oder auch das „äu“ in Mäuse. Es wird vom TTSS in ein gleich klingendes „EU“ verwandelt. Auch das „v“ wird intern (wie weitere Laute) in ein „f“ gewandelt, um Ihre Rechtschreibung nicht zu ruinieren, wenn Sie länger mit dem TTSS arbeiten.

Anders als TTSS auf PCs, welche tausende von Wörtern komplett in hoher Qualität gespeichert haben und hunderte Megabyte an Speichern dafür brauchen, stehen für die hinterlegten Stimmsamples dieses kleinen BASIC Computersystems nur insgesamt 64KByte zur Verfügung. Deshalb ist vom Anwender ein wenig Kreativität erforderlich um gute Resultate zu erzielen. Mit ein bisschen Erfahrung hat man aber in der Regel nach 2 bis 3 Versuchen eine gut verständlich klingende Variante gefunden.

1)
Schreiben Sie den Text zunächst in kleinen Buchstaben.
SAY „die erde ist eine sehr grosse scheibe“
Lassen sie das TTSS diesen Text sprechen und sie werden sehen, dass er komplett unverständlich ist.

2)
ersetzen sie alle Buchstaben die zusammengefasst gespeichert sind durch gross geschriebene Buchstaben.
SAY „ dIE ERde ist EIne sehr grosse SCHEIbe“
Das klingt dann bereits wesentlich verständlicher

3)
Sie werden feststellen, dass das das „o“ zu kurz gesprochen wird. Ersetzen sie es durch die Variante „O“. Das „eh“ wird im normalen Sprachgebrauch als langes „e“ gesprochen, das „h“ ist nicht zu hören. Streichen Sie es also und fassen sie „er“ zusammen.
SAY “dIE ERde ist EIne sER grOsse SCHEIbe“

4)
Um die Grösse der Scheibe zu betonen kann man das ER dehnen. Fügen sie vorher ein langes EE ein
SAY “dIE ERde ist EIne sEEER grOsse SCHEIbe“
Sie können auch die Varianten sEEr versuchen.
sEER geht nicht, da zuerst EE gelesen wird und R nicht als
Sample abgelegt ist.

5)
Die Buchstaben a bis z sind in der Regel alle kurz. Im Fall der Banane werden aber sowohl „a“ als auch „n“ gedehnt gesprochen, was sicherlich an der Krümmung liegt.
SAY "EIne banane SCHwimmt AUf dER SCHElbe"

6)
Ersetzen Sie diese Buchstaben durch die alternative Variante A und N und verlängern sie das zweite „a“ das deutlich länger gesprochen wird als das erste mit einer Verdoppelung des A
"EIne bANAANe SCHwimmt AUf dER SCHElbe"
Sie können auch mm durch MM ersetzen was noch etwas besser klingt.

7)
Pausen können die Verständlichkeit (vor allem vor den Buchstaben d g k t b) erhöhen.
"EIne bA NAA Ne SCHwimmt AUf dER SCHEl be"

SPRACHAUSGABE VON VARIABLENINHALTEN

Während Sie bei der Ausgabe von Texten weitgehend Gestaltungsfreiheit haben, können Sie die Ausgabe von Werten nicht beeinflussen. Zahlen von 0 bis 9 sind als komplette Worte gespeichert und werden nicht aus einzelnen Lauten zusammengesetzt.

MyFloat=-1.03E02
MyWord= 1234

SAY
FPPRINT (MyFloat,3) führt zur Ausgabe MINUS – EINS – KOMMA - NULL – DREI – E – NULL - ZWEI
SAY MyWord führt zur Ausgabe von EINS – ZWEI – DREI - VIER
SAY „grAd“ Physikalisch Grössen können Sie natürlich an die Wertansage anhängen.

FILE PLAYER – Abspielen von eigenen Klängen

Eigene Klänge wird man wohl am Häufigsten benutzen um die Ausgaben auf den Beeper durch wärmere und vielseitigere Varianten zu ersetzen. Man kann aber die Tastatureingaben z.B. auch mit DTMF Tönen hinterlegen und hat damit fast schon ein Telefonwählgerät. Besonders bei Robotern lassen sich Interaktionen mit der Umwelt gut in Klängen ausdrücken. Mit bis zu 60s Abspielzeit kann man die Control Unit auch in einen einzigartigen Soundgenerator (z.B. für die Modelleisenbahn) verwandeln, dessen verschiedene Klänge auch über Ereignisse getriggert werden können. Auch können natürlich einzelne Klänge endlos schleifen und nur sporadisch von anderen unterbrochen werden. Der Phantasie sind hier mit ein bisschen Geschick bei der Soundbearbeitung keine Grenzen gesetzt.

Um eigene Klänge abzuspielen sind folgende Schritte erforderlich:

SAMPELN EINES SOUNDFILES

Benuzen Sie dazu eines der vielfältig als Freeware erhältlichen Tools. Sampeln sie den Klang mit 7333 kHz Abtastrate (mono) und speichern sie es als WAVE Datei (.wav) im Format 16Bit, Mono, signed.
Vorhandene Files können mit diesen Tools meistens auch resampled werden um das Format anzupassen.
Beep oder DTMF Töne sind in riesiger Anzahl im Internet zu finden.

BEARBEITEN EINES SOUNDFILES

Öffnen Sie das Sound File mit einem HEX-Editor (ebenfalls als Freeware erhältlich).

Tragen Sie an der Adresse 0x030 (Hexadezimal) den ASCII Code ein, unter dem Sie das File mit einem Buchstaben oder Zahlenwert aufrufen wollen. Der Wert der vorher an 0x030 stand muss ersetzt werden, Sie dürfen kein zusätzliches Byte einfügen.
Wollen Sie z.B. das File mit SAY „A“ abspielen, so tragen sie an 0x030 den Wert 41 ein (das ist der ASCII Code für A). Es ist die ID für dieses File

Entfernen Sie das letzte Byte aus dem File. Speichern sie das File als Text File (z.B. MyFile.txt)

LADEN EINES EIGENEN FILES AUF DAS EEPROM

!! Das EEPROM muss dafür auf Adresse 160 (dez) adressiert werden !!

Sie können zu beliebigen Zeiten weitere Files auf das EEprom laden, solange der Speicherplatz dafür ausreichend ist. Dazu muss das EEPROM vorher jedoch vollständig gelöscht sein. Laden Sie das Programm CLEAR_EEPROM.BAS in den BASIC Computer und löschen Sie das EEPROM, was einige Minuten dauert.

Das Sound File wird mit einem speziellen Ladeprogramm, das im SOUND AND VOICE PACK enthalten ist, auf das EEPROM geladen. Laden Sie das Programm SOUNDLOADER.BAS in den BASIC-Computer und starten sie es. Es sucht sich selbst das Ende des vorher geladenen Files und hängt das neue File an.
Wenn sie das Programm auffordert den Download zu starten, öffnen Sie das Terminal in der Workbench (nicht das HYPERTERMINAL, es hat einen Bug). Wählen Sie 1200Bd als Baudrate und „Textdatei senden“ Wählen Sie Ihre Datei z.B. MyFile.txt aus, und klicken Sie „Datei senden“.
Je nach Grösse des Files kann es mehrere Minuten dauern bis der Ladevorgang beendet ist.
Dann zeigt das Programm die Grösse der Original Datei an (es dürfen keine Bytes verloren gegangen sein!) und die zuletzt beschriebene Adresse im EEPROM (sie können feststellen wieviel Platz für weitere Files verbleibt)
Ein geladenes File beansprucht auf dem EEPROM rund die Hälfte der originalen Dateigrösse, da diese mit 16Bit gesampelt ist, und das Ladeprogramm dies auf 8Bit umrechnet.
Sie können das Programm jetzt fortsetzen, wenn Sie weitere Files laden wollen.Ein Löschen ist jetzt nicht mehr erforderlich. Drücken Sie dazu die Taste 9 auf der Tastatur am Application Board.

ERSETZEN VON FILES

Sie können ein File auf dem EEPROM jederzeit ersetzen wenn sie ein File mit der gleichen ID laden.
Das alte File wird ignoriert, jedoch nicht gelöscht (da das neue File angehängt wird) und beansprucht weiterhin Speicherplatz. Das ist nur zu vermeiden wenn das EEPROM komplett gelöscht wird und alle Files neu geladen werden.

SPIELEN EINES FILES AUF DEM EEPROM

Wie auch der TTSS benötigt der FILE PLAYER den DAC VOICE MODE. Zusätzlich aber muss dem Betriebssystem gesagt werden, dass es die Kompression abschaltet die für das PHONEM FILE benutzt wird.

CONFIG2.INIT	Bit 4 aktiviert den VOICE MODE
CONFIG2.PUT 00110000b	Bit 5 deaktiviert die Kompression
CONFIG2.OFF	

Der FILEPLAYER benötigt unbedingt ein 100ms Pause-File, das auf das EEPROM neben Ihren anderen Klängen geladen werden muss. Ein passendes File finden sie neben einigen anderen (als Beispiele für verschiedene Beep Töne) im VOICE AND SOUND PACK.

Auch der FILEPLAYER hat als default Adresse 160. Stellen Sie den FILE PLAYER auf die Adresse an der sie das EEPROM betreiben.

z.B.

VOICEBASE = 174

Wenn Sie Ihr File unter der ID „A“ also 41 HEX gespeichert haben, rufen sie es mit

SAY „A“

auf. Beachten Sie, dass einige Buchstaben vom OS gewandelt werden oder eine Pause hinzugefügt wird.

Das Programm wird für die Dauer der Soundausgabe angehalten, Interrupts werden aber weiterhin bedient. Während einer Sprach oder Soundausgabe darf kein Interrupt ausgeführt werden der auf die Schnittstelle oder das LCD schreibt.

DIE EIGENE STIMME FÜR DAS TTSS

Sie sehen, dass sie auf diese Weise dem TTSS Ihre eigene Stimme geben können. Wenn Sie die Samples der von Ihnen gesprochenen Buchstaben so im EEPROM speichern, dass das Sample zum Buchstaben „a“ auch die ID „a“ bekommt (und ebenso für die weiteren Buchstaben), wird das TTSS im OS ohne jede Probleme auch mit Ihrer Stimme funktionieren.

SAY „hallo“

wird also dann Ihre Stimmsamples zusammenfügen und sprechen.
Für erste Versuche ist es ratsam mit den Zahlen von 0 bis 9 zu beginnen, da es nur 10 Files sind, die erstellt werden müssen. Sie bekommen die IDs 0 bis 9, also die ASCII CODES 30 bis 39 (HEX) die im File an Stelle 0x030 stehen müssen.

Da Ihre Files keine Kompression haben, muss diese (wie auch für alle anderen Ihrer Files) abgeschaltet werden (Bit 5 in CONFIG2 ist dann 1)

Wenn sie wirklich eigene Stimmsamples verwenden wollen, müssen Sie wissen nach welchen Regeln das OS die Buchstaben verarbeitet. Hier ist eine Zusammenstellung:

Wie bereits eingangs erwähnt werden Buchstaben zusammengefasst und unter einer speziellen ID abgelegt:
EI = 01, CH = 02, SCH = 03, ER = 04, EU = 05, AU = 06

Ausserdem werden einige Buchstaben vom OS gewandelt:
IE → II, äu → EU, qu → kw, ck → k, e → ä, E → Ä, v → f, y → Ü, c → k

Eine Leerstelle ist unter der ID 20 (hex) abgelegt und ist eine 100ms lange Pause. Diese Pause wird bei der Ausgabe allen Zahlen und folgenden Buchstaben vorangestellt:
t, g, k, d, b

LAUSPRECHER ANSCHLUSS

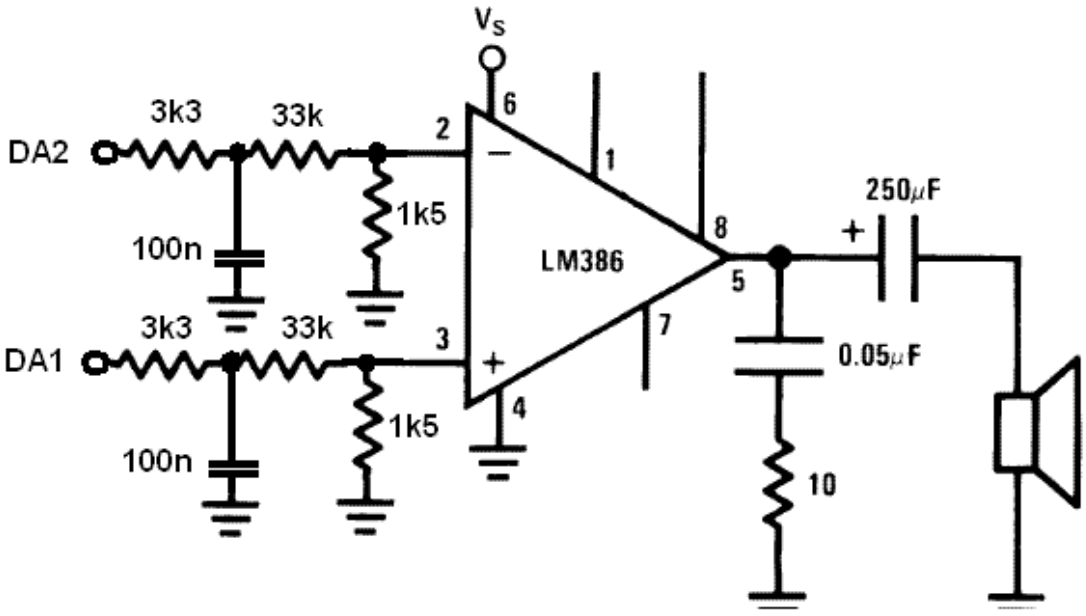
Obwohl ein kleiner Lautsprecher direkt an den DA1 angeschlossen werden kann und keine nachteiligen Folgen für den DA Ausgang zu erwarten sind, ist der Lautsprecher doch eine grosse (und permanente) Last an diesem Ausgang. Entsprechend steigt auch die Stromaufnahme der Unit.
Wesentlich schonender ist der Betrieb des Lautsprechers mit einem Differenzsignal. Stellen Sie dazu den DA2 auf einen Wert von 128 und schliessen Sie den Lautsprecher an DA1 und DA2 an. Die Wiedergabe ist etwas leiser, aber DA1 und DA2 haben in Sprechpausen praktisch das gleiche Signal und der Lautsprecher sieht an beiden Anschlüssen das gleiche Potenzial und ist damit stromlos.

Gleichzeitig werden damit auch die unvermeidlichen Hintergrundgeräusche bei der Sprachausgabe weitgehend eliminiert. Sie entstehen, da andere Systeminterrupts die Bedienung der DAC Interrupts verzögern. Zwar nur wenige Mikrosekunden, aber dennoch als Störgeräusche hörbar.

Auch beim Anschluss eines Audioverstärkers ist es daher sinnvoll den Eingang mit dem Differenzsignal zu speisen. Die 26kHz werden allein damit schon weitgehend unterdrückt, ebenso wie die Störgeräusche

Die nachfolgend gezeigte Schaltung arbeitet als Differenzverstärker und bringt sehr gute und saubere Ergebnisse. Die Betriebsspannung ist 5V.
Stellen sie den DA2 unbedingt auf einen Wert von 128. Die Schaltung ist DC gekoppelt und der Verstärker würde wegen der unterschiedlichen Gleichspannungsanteile an DA1 und DA2 in die Begrenzung gehen.
Das Tiefpassfilter ist auf Sprachwiedergabe eingestellt. Für die Verstärkung von Klängen ist der Wert der

Ausgangsverstärker Differenzbetrieb



Tiefpass Kondensatoren von 100nF auf 10nF zu ändern. Die Grenzfrequenz liegt dann bei ca. 5kHz

Wenn Sie den Verstärker single ended betreiben wollen, legen Sie den Anschluss für DA2 mit 33k an 5V.

Für eine reine Verstärkung genügen bereits ein paar Bauteile aus der Bastelkiste. Bild 1 zeigt einen kleinen Verstärker für das PWM Signal. Bild 2 zeigt die gleiche Schaltung für den Differenzbetrieb und bringt auch hier die besseren Ergebnisse.

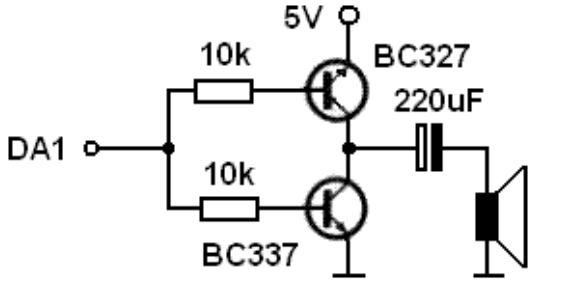


Bild 1

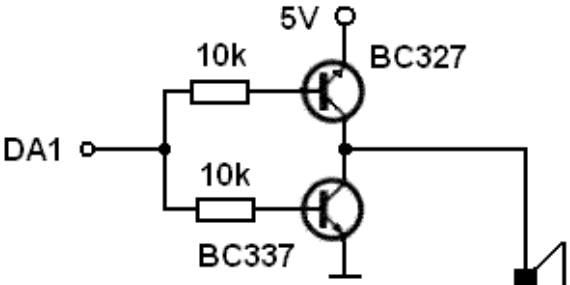
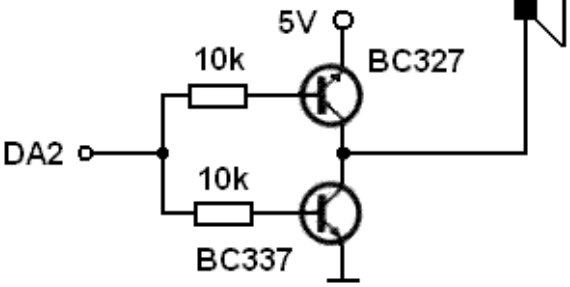


Bild 2



Beim Anschluss an Leistungsverstärker ist unbedingt zu beachten:

Die 26 kHz sind unhörbar, können aber den Verstärker (je nach Ausführung) voll aussteuern und damit zur Zerstörung von angeschlossenen Lautsprechern führen.

Wir hoffen, dass Ihre Rechtschreibung nicht von der Arbeit mit dem TTSS beeinflusst wird, und hoffen dass Sie nicht zu der Überzeugung gelangt sind, die Erde sei tatsächlich eine Scheibe.

BASIC CONTROL UNITs

http://www.Spiketronics.com

D

Impressum

GB

Imprint