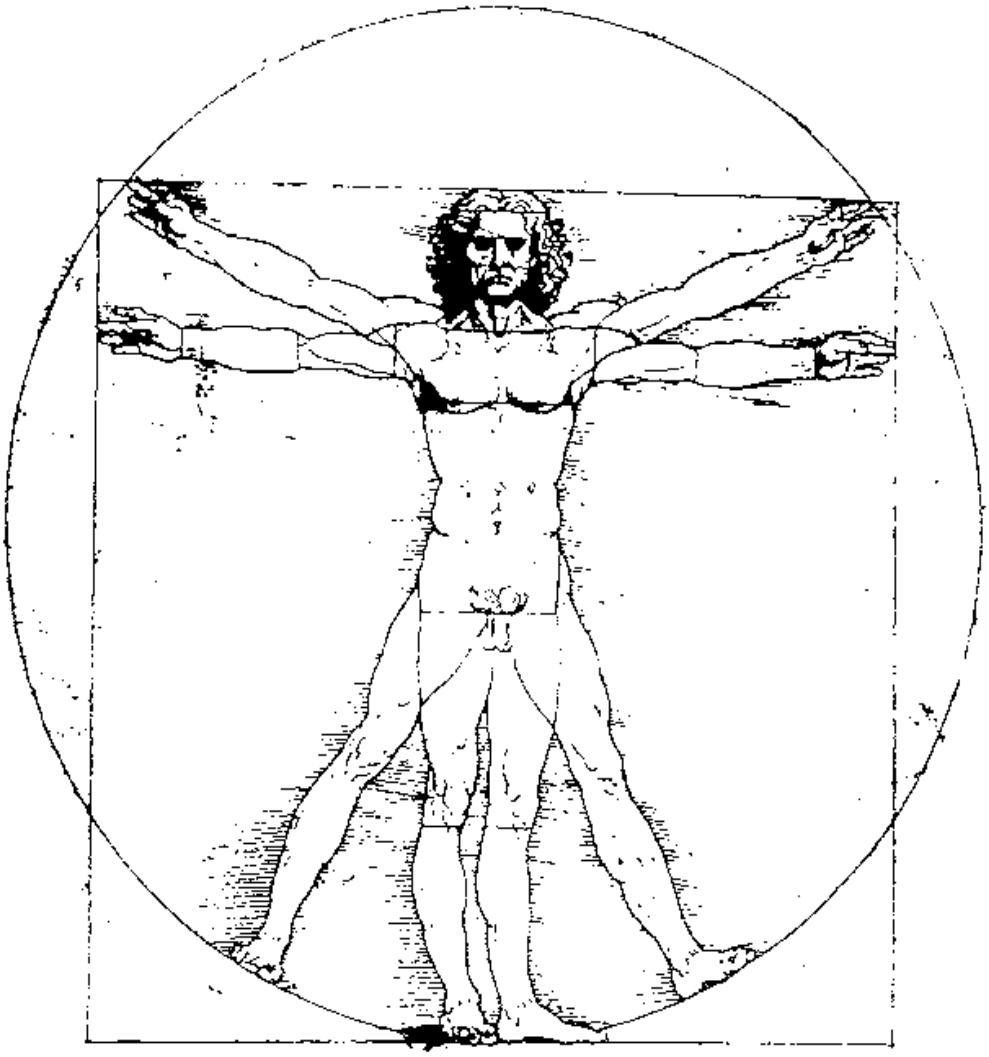


ADAM/P-µM/T6963C/2

ADAM/P-µM/T6963C/2
Messtechnik-Graphik-Toolbox

Inhaltsverzeichnis

1) ADAM/P-µM/T6963C/2 – Messtechnik-Graphik-Toolbox

ADAM:	Analog-Digital-Aquisitions-Modul
/P	Pascal
µM/T6963C	Treiber für„mikroMedia for PIC24“ und Controller T6963C
/2	Toolbox 2 erweitert gegenüber /1

Die Graphik-Toolbox **ADAM/P-µM/T6963C/2** bietet Sourcecode-Funktionen in Pascal für die Erfassung und Anzeige von Messwerten mit einem Pic-Controller der Serien PIC24... auf einem Graphik-Lcd-Schirm mit dem Controller T6963C und auf TFT.
Es können auch 8bit-PIC-Controller PIC18F2620,4620 eingesetzt werden, jedoch mit einem reduzierten Funktionsumfang der Toolbox. Vorallem entfallen dann Funktionen mit größerer Heap-Verwendung

Entwickelt wurde das System mit mikroPascal von Mikroelektronika.

Eigenschaften für T6963C

- PASCAL-Sourcen (Mikroelektronika-Compiler)
- getestet mit Prozessoren PIC18F2620 und PIC24FJ128GB110
- getestet mit Graphik-LCD EA W240-7K (240x128 Pixel) und Interface-Karte
- getestet mit mikroMedia for PIC24

- Graphik-Grundfunktionen ähnlich früherem Borland BGI
  - 8 Graphik-Seiten
  - SetPixel, Line, Get-Putimage, horizontal und vertikal schreiben, usw.

- Analog-Datenerfassung
  - Direktaufruf mit Samplerate-Vorgabe
  - Interrupt-gesteuert mit Samplerate-Vorgabe
  - Berechnung der Register-Parameter
  - Ringpuffer

- Analog-Signal-Anzeige mit Cursor-Einblendung und Werteanzeige

- Digital-Signal-Anzeige

- Editier-Fähigkeit der Analog-Digital-Anzeigen. Damit können Signale für einen Arbiträr-Generator erstellt weeden

- X,Y-Bargraph und Scrollbargraph

- 7-Segment-Anzeige in variabler Größe und Formatierung

- Spektrum-Anzeige

- Skalierfunktionen

- Listbox-Einblendung in mehreren Ebenen für Parameter-Auswahl

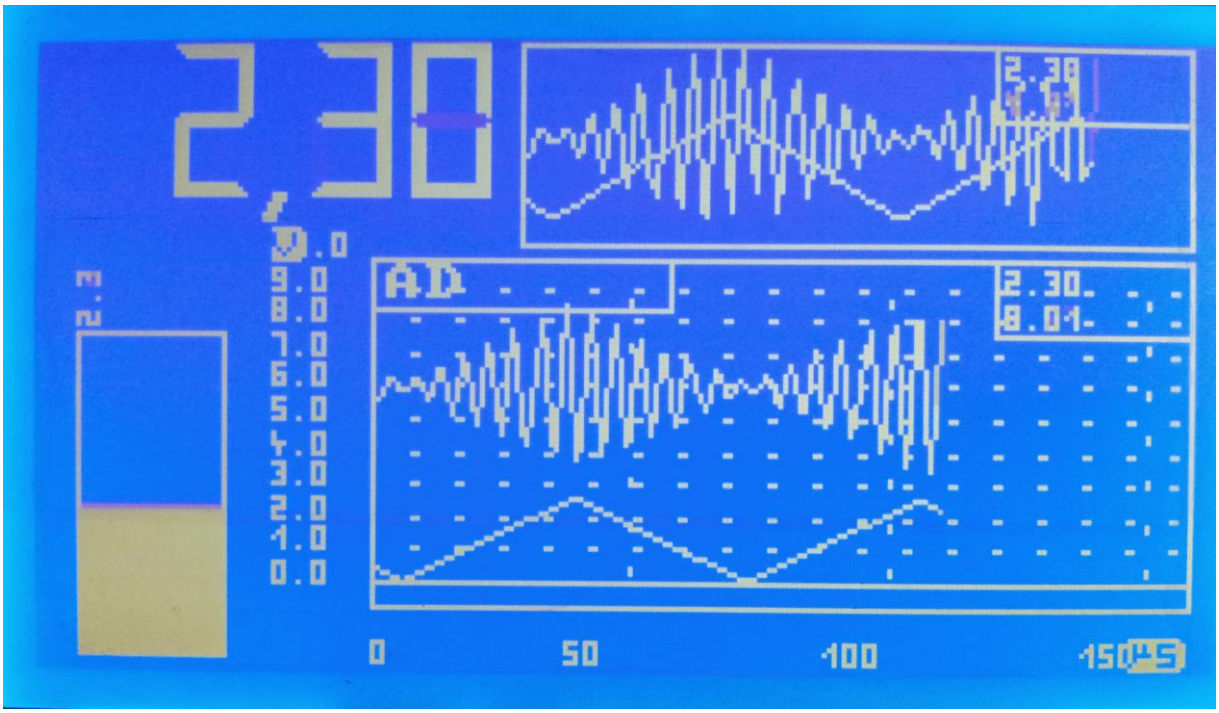


Bild 1: Testfunktionalität in Programm **Toolbox_24_Demo1** für T6963C.
Es zeigt Funktionen aus p_mdyC86, p_digC86, p_md3C86.

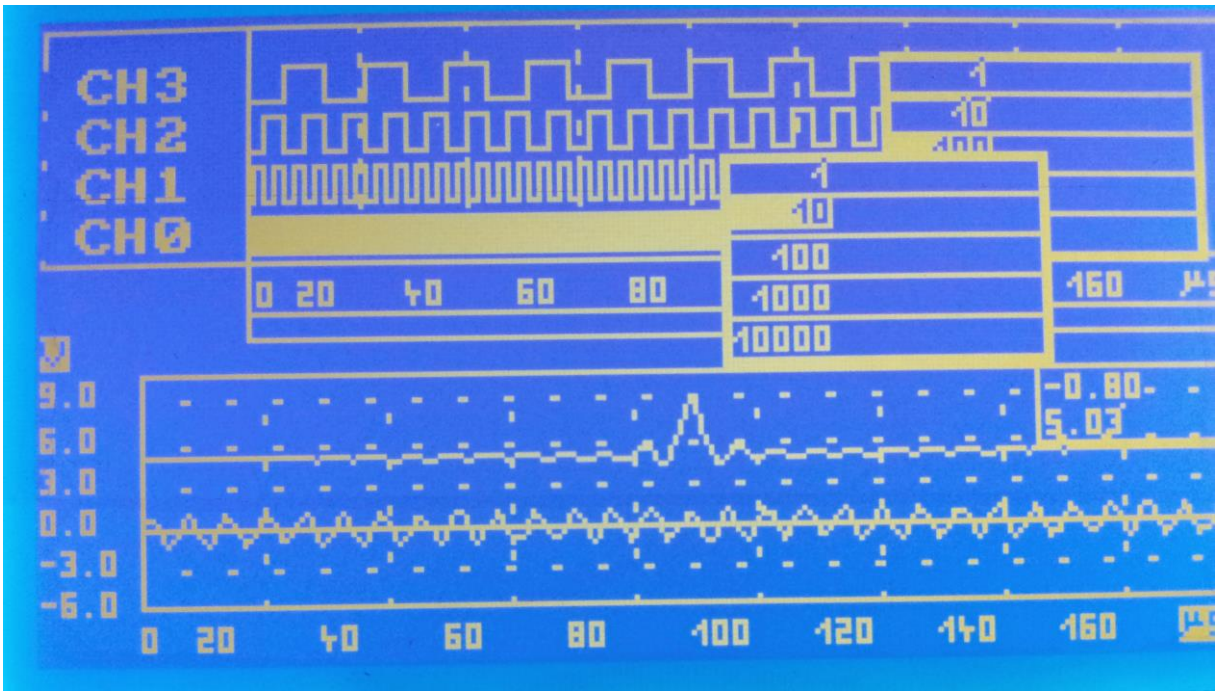


Bild 2: weitere Testfunktionalität in Programm **Toolbox_24_Demo1** für T6963C**.**
Es zeigt Funktionen aus p_larC86, p_segC86, p_md3C86.

Eigenschaften für „mikroMedia for PIC24“

- PASCAL-Sourcen (Mikroelektronika-Compiler)
- getestet mit Prozessor PIC24FJ256GB110
- getestet mit mikroMedia for PIC24

- Graphik-Grundfunktionen ähnlich früherem Borland BGI
  - 1 Graphik-Seite
  - SetPixel, Line, horizontal und vertikal schreiben, usw.

- Analog-Datenerfassung
  - Direktaufruf mit Samplerate-Vorgabe
  - Interrupt-gesteuert mit Samplerate-Vorgabe
  - Berechnung der Register-Parameter
  - Ringpuffer

- Analog-Signal-Anzeige mit Cursor-Einblendung und Werteanzeige

- Digital-Signal-Anzeige

- Editier-Fähigkeit der Analog-Digital-Anzeigen. Damit können Signale für einen Arbiträr-Generator erstellt weeden

- X,Y-Bargraph und Scrollbargraph

- 7-Segment-Anzeige in variabler Größe und Formatierung

- Spektrum-Anzeige

- Skalierfunktionen

- Touch-Funktionen

- Flash-Filesystem

Hier Bilder aus aus dem Testprogramm zur „mikroMedia for PIC24“-Toolbox

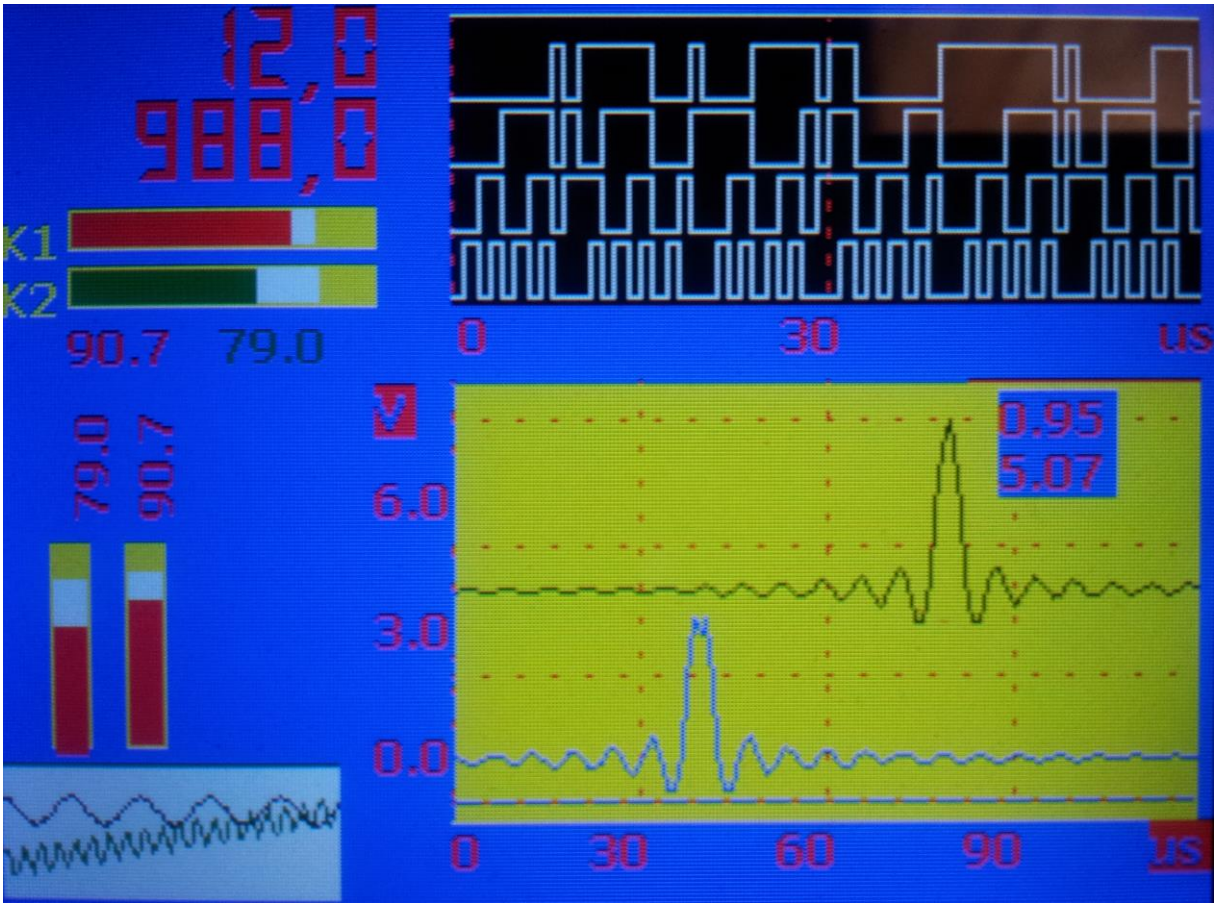


Bild3: Testfunktionalität in Programm **Toolbox_24_Demo1** für mikroMedia**.**
Es zeigt Funktionen aus p_larC86, p_md3C86, p_mdxC86, p_mdyC86 und p_digC86..

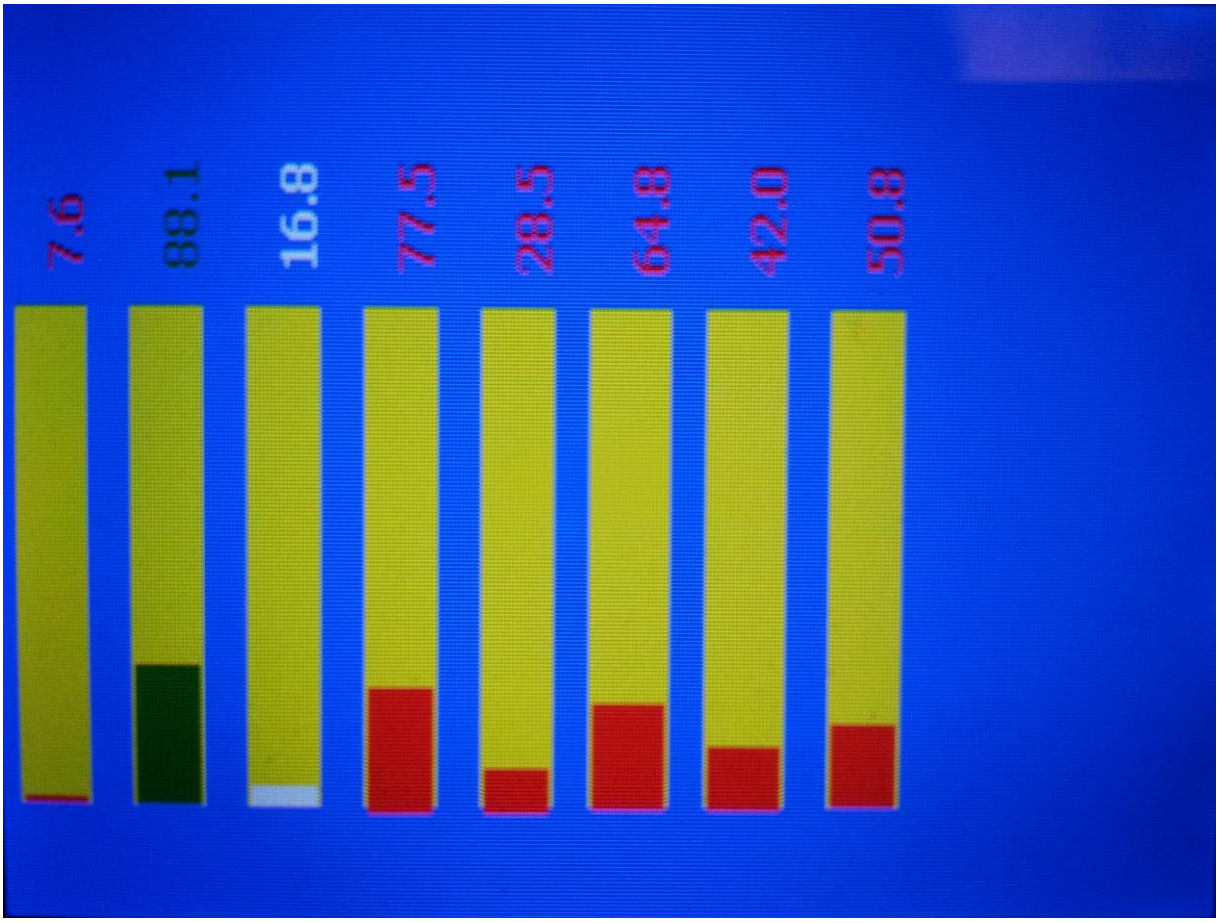


Bild4: Testfunktionalität in Programm **Toolbox_24_Demo1** für mikroMedia**.**
Es zeigt 8 parallele BarGraphen, die mit p_mdyC86 erstellt wurden...

2) Lizenz-und Garantie-Bestimmumgen

Die Software wurde umfanfreich getestet. Eine Fehlerfreiheit kann jedoch nicht garantiert werden.
Eine Haftung für eventuelle Fehler in den damit erstellten Anwendungen wird nicht übernommen.

Die Lizenzierung für das Software-Paket erfolgt mit dem Kauf.
Eine Weitergabe des Sourcecodes an Dritte ist nicht gestattet.

3) Übernahme der Software in eigenes Projekt

Auf der CD befindet sich Projekte für „mikroPascal PRO for PIC“ und „mikroPascal PRO dsPIC“.
Diese Projekte können direkt in die Mikroelektronika-IDE übernommen werden.

4) Units - Übersicht

Das Software-Paket ADAM/P-MT6963C besteht aus einzelnen Units die in ein Hauptprogramm eingebunden werden .

_

Name	Aufgabe	Anzahl Proc/Func	Seite
p_defC86	**Standard-Definitionen**	4	12
p_bitC86	**Sonderfunktionen**	**12**	14
p_matC86	**math.Funktionen**	**7**	18
p_debC86	**Debug-Funktionen**	**19**	20
p_ioPic18F2620	**8bit-PIC-Treiber für Interfacekarte**		**Siehe p_ioPic24FJ128GB110**
p_ioPic18F2620D	**8bit-PIC-Treiber ohne Interfacekarte**		**Siehe p_ioPic24FJ128GB110**
p_adPic18F2620	**8bit-AD-Treiber**		**Siehe p_adPic24FJ128GB110**
p_ioPic24FJ128GB110	**PIC-Treiber für Interfacekarte**	**14**	25
p_ioPic24FJ128GB110D	**PIC-Treiber ohne Interfacekarte** (direkte Kopllung PIC-Lcd)		**Siehe p_ioPic24FJ128GB110**
p_ioMM24	**PIC-Treiber für mikroMedia**		
p_adPic24FJ128GB110 p_adPic24FJ256GB110	**AD-Treiber**	**17**	70
p_grpC86	**Graphik-Grundfunktionen (T6963C-Treiber)**	**66**	30
p_grpMM24	**Graphik-Grundfunktionen (mikroMedia)**		
p_md3C86	**Analogsignale anzeigen**	**10**	61
p_sklC86	**Skalier-Funktionen**	**13**	51
p_mdxC86	**X-BarGraph / Scrollbar**	**6**	79
p_mdyC86	**Y-BarGraph / Scrollbar**	**6**	82
p_spgC86	**Spektrum**	**4**	67
P_digC86	**7-Segment-Anzeige**	**4**	77
p_icoC86	**Icon-Ausgabe**	**3**	69
p_larC86	**Digitalsignale anzeigen**	**11**	57
p_segC86	**Selektionen aus Listbox**	**4**	65

Das Erweiterungs-Paket ADAM/P-T6963C/2 bietet zusätzliche Funktionen

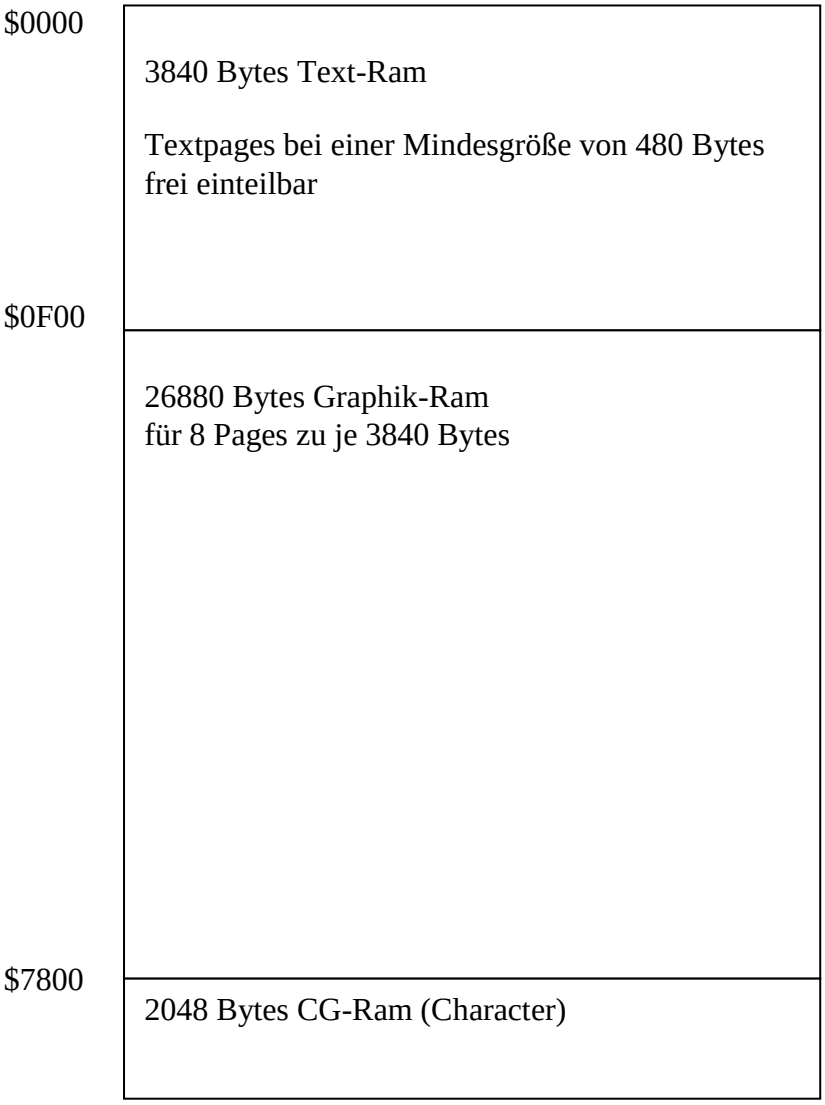
Name	Funktion
p_drsC86	**Analoginstumente (Drehspul)**
p_tchC86	**Touch-Funktionen**
p_cntC86	**Counter-Funktionen**
p_eFlash	**Flash-Filesystem**

5) Definition der Speicherbereiche im LCD-Modul

Die grundsätzliche Adressenzuteilung für Text und Graphik-Ram erfolgt in der Unit p_ioPic.

Die Größe einer Graphik-Page (max. 240*128) beträgt 3840 Bytes und ist fest.

Die Mindest-Größe einer Text-Page (30 Zeichen/Zeile bei 16 Zeilen) beträgt 480 Bytes..
Hier kann jedoch auch eine größere Page als der Sichtbereich definiert werden, womit dann Scrollen nötig wird.
Die Definition Textpage-Größen wird in der Unit p_ioPIC... festgelegt.



Wird eine kleineres LCD-Modul verwendet (z.B.128x64), so können die Ram-Bereiche entsprechend verkleinert werden.

6) Programmierhinweise:

Heap-Parameter

Im Hauptprogramm :

(* Variablen in Memory Manager Library: *)

```
unit MemManager;
  const HEAP_START  : word; external;
  const HEAP_SIZE      : word; external;
```

Variable in p_defC86:
g_HeapStart

Beispiel für Hauptprogramm::

```
const
  heap_start : word = 5500      ← aus Compile-Lauf RAM-Bedarf + Reserve
  heap_size  : word = 10000   ← geschätzter Heap-Bedarf + Reserve

begin
  g_HeapStart := heap_start;
  ...
  ...
end.
```

Beachten: für PIC24FJ128GB110 in IDE/Projekt-Edit 10000Bytes Heap anfordern

ProgrammStart

Heap-Initialisierung (siehe oben)

Units-Initialisierungen:
Hier alle p_xxx_Init-Funktionen der verwendeten Units starten

Systemvariable

SyRc Integer-Variable für Return-Codes
Codes sind in p_defC86 definiert

7) **Schichtung der Software-Module**

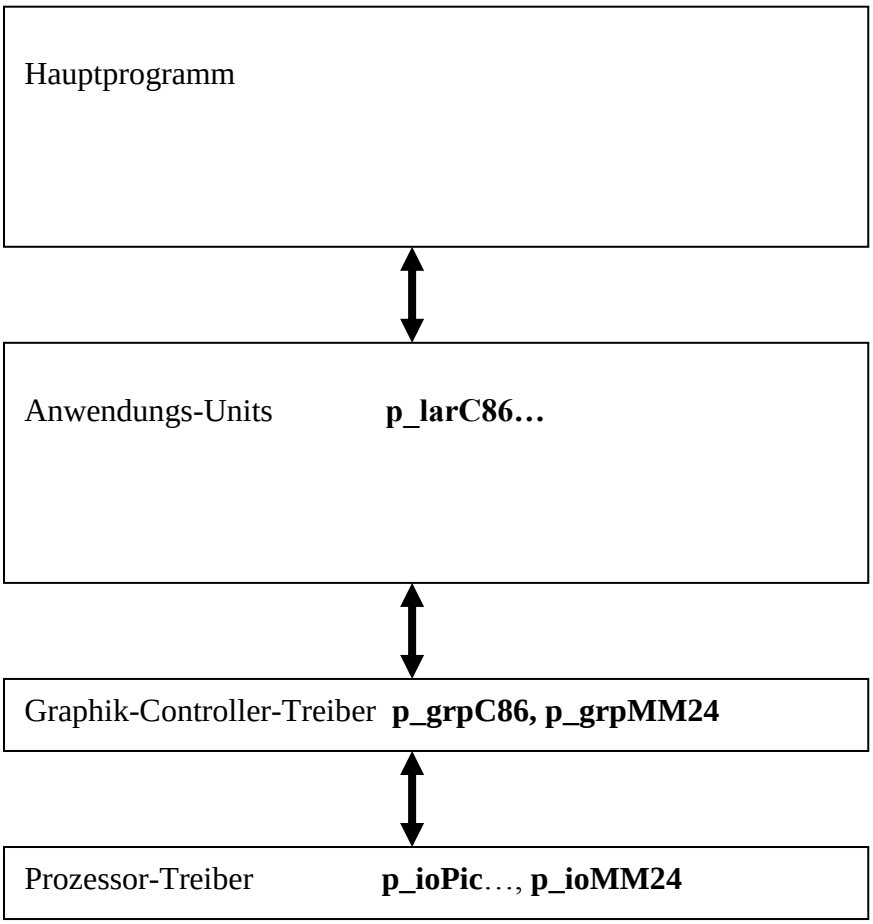
Anwendungsspezifische Funktionen /1 **p_larC86** **p_md3C86** **p_sklC86** **p_mdxC86** **p_mdyC86** **p_spgC86** **p_digC86** **p_icoC86** **p_segC86**	Anwendungsspezifische Funktionen /2 **p_larC86** **p_md3C86** **p_sklC86** **p_mdxC86** **p_mdyC86** **p_spgC86** **p_digC86** **p_icoC86** **p_segC86** **p_drsC86** **p_cntC86**
Hilfsfunktionen **p_matC86** **p_debC86**	Hilfsfunktionen **p_matC86** **p_debC86**
Graphik-Controller-spezifische Funktionen **p_grpC86** **p_grpMM24**	Graphik-Controller-spezifische Funktionen **p_grpC86** **p_grpMM24**
Prozessor-spezifische Funktionen **p_ioPIC18F2620** **p_ioPIC24FJ128GB110** **p_ioMM24**	Prozessor-spezifische Funktionen **p_ioPIC18F2620** **p_ioPIC24FJ128GB110** **p_ioMM24** **p_tchC86** **p_eFlash**
Allgemeine Funktionen **p_bitC86** **p_defC86**	Allgemeine Funktionen **p_bitC86** **p_defC86**

Der Einsatz eines anderen Graphik-Controllers als T6963C hat Auswirkungen auf **p_defC86, p_ioPIC... und p_grpC86**.

Entwickelt wurde das System mit den Prozessoren
- PIC24FJ128GB110 mit 128kB Flash und 16kB Ram..
- PIC24FJ256GB110 mit 256kB Flash und 16kB Ram..
- PIC18F2629 64kB Flash und 3,9kB Ram..

Es kann jedoch auch ein Prozessor mit weniger Speicher (z.B. PIC18F2620) eingesetzt werden, wenn nur wenige Module benötigt werden.

Der Einsatz eines anderen MikroControllers als **PIC24FJ128GB110** hat Einbfluß auf **p_defC86 und p_io....**
Auch müssen Heap-Intensive Funktionen entfernt werden.
z.B: Mirror-Funktionen aus p_grpC86



8) Unit-Dokumentationen

Unit **p_defC86**

Enthält globale Variable und Konstante für das komplette Software-Paket.

==========

procedure **p_defC86_Init**

Aufgabe: Initialisierung von globalen Variablöen

Parameter: -

==========

function **SyResult:** integer

Aufgabe: liefert Inhalt von SyRc und setzt Syrc nach dem Auslesen auf 0

Parameter: -

==========

procedure **SampleString**(SampleUnit: byte;
var SampleString: str_3)

Aufgabe: Textzuordnung zu Zeiteinheit-Konstante

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	SampleUnit	byte	Input	Konstante für Abtast-Zeiteinheit z.B. c_sec
var	SampleString	String[3]	Output	String Zeiteinheit

==========

procedure **ValueString**(ValueUnit: byte;
var ValueString: str_3)

Aufgabe: Textzuordnung zu Wert-Konstante

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	ValueUnit	byte	Input	Konstante für Werte-Einheit z.B. c_mV
var	ValueString	String[3]	Output	String

Unit **p_bitC86**

Unit enthält spezielle Konvertierfunktionen.

==========

procedure **ByteToHexadez**(bin: byte;
var hd: string[2])

Aufgabe: Byte in HexaDezimal-String wandeln

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	bin	byte	Input	Byte-Wert
var	Hd	String[2]	Output	Hexa-String

==========

procedure **WordToHexaDez**(wd: word;
var hd: string[4])

Aufgabe: Word in HexaDezimal-String wandeln

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Wd	word	Input	Byte-Wert
var	Hd	String[4]	Output	Hexa-String

==========

procedure **i_nibble_b** (var zahl : byte;
h_nibble : char;
l_nibble : char)

Aufgabe: Byte aus Hexa-String bilden

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
var	zahl	word	Output	Byte-Wert
	H_nibble	char	Input	High-Hexa-Char
	L_nibble	char	Input	Low-Hexa-Char

==========

procedure **XFloatToStr**(iFloat: real; AnzNk: byte;
var oStr: string[20])

Aufgabe: Floatzahl in String ausbereiten

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	iFloat	Real	Input	Float-Wert
	AnzNk	byte	Input	Nachkomma-Stellen
var	oStr	String[20]	Output	Zahl-String

==========

procedure **XXFloatToStr**(iFloat: real; AnzVk: byte; AnzNk: byte;
var oStr: string[20])

Aufgabe: Floatzahl in String ausbereiten

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	iFloat	real	Input	Byte-Wert
	AnzVk	Byte	Input	Vorkomma-Stellen
	AnzNk	Byte	Input	Nachkomma-Stellen
var	oStr	String[20]	Output	Zahl-String

==========

function **fx**(xv,xb: integer): word

Aufgabe: Pixeldifferenz mod 8 umrechnen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	xv	integer	Input	xv-Pixel
	xb	integer	Input	Xb-Pixel
Function	Fx	word	Output	Byte-Differenz in VideoRam

==========

function **substr**(var iTxt: string[25];
von,len: byte): string[25]

Aufgabe: Teilstring aus Quell-String kopieren

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	iTxt	String[25]	Input	Quell-String
	Von	Byte	Input	Startposition für Kopieren
	Len	Byte	Input	Kopierlänge
Function	Substr	String[25]	Output	Teilstring

==========

procedure **Xrtrim**(var iStr: string[25]; il: byte)

Aufgabe: String nach Rechts ausrichten mit definierter Längel

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
var	iStr	word	Input/ Output	Auszurichtender String
	il	String[4]	Input	Stringlänge

==========

procedure **Get_Nk**(var s : string[18];
var s_nk: string[5])

Aufgabe: Nachkommastellen aus Zahlenstring auslesen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	S	String[18]	Input	Byte-Wert
var	Hd	String[5]	Output	Nachkomma-String

==========

procedure **Get_Vk**(var s : string[18]; (* 12Vk.5Nk *)
var s_vk: string[12])

Aufgabe: Vorkommastellen aus Zahlenstring auslesen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	S	String[18]	Input	Zahlenstring
var	S_vk	String[12]	Output	Vorkomma-String

==========

procedure **iGet_Vk**(var s: string[18];
var len: byte);

Aufgabe: Stringlänge bis zum Dezimaltrenner(.) suchen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
var	s	String[18]	Input	Input-String
var	Len	Byte	Output	Stringlänge bis zum Dezimaltrenner

Unit **p_matC86**

Enthält spezielle math. Funktionen

==========

procedure **p_matC86_Init**

Aufgabe: Initialisierung der Unit

Parameter –

==========

function **iRound**(iw: real): integer

Aufgabe: Runden eines Real-Wertel

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	iw	real	Input	Float-Wert
function	IRound	Integer	Output	Gerundeter Wert

==========

procedure **Randomize**(Bereich: word)

Aufgabe: Zufallsgenerator einschalten

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
P_gr	Bereich	Word	Input	Zufallszahlen von 0..Bereich erzeugen

==========

function **Random**: real

Aufgabe: Zufallszahl generieren

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
Function	Random	Real	Output	Zufallszahl 0..Bereich

==========

function **log2N**(iw: word): byte

Aufgabe: 2er-Logarithmus einer Zahl ermitteln z.B. 3=log2(8)

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Iw	word	Input	Zahl
Function	Log2N	Byte	Output	2er-Logarithmus der Zahl

==========

function **Bit2NReal**(n: byte): real

Aufgabe: 2er-Potenz einer Zahl ermitteln (2^n)

Parameter:

ggt

Worth/Ref	Name	Typ	I/O	Kommentar
	N	Byte	Input	Exponent
Function	Bit2Nrea	Real	Output	2^n

==========

function **ggt**(m,n: longint): real

Aufgabe: größten gemeinsamen Teiler(GGT) von 2 Zahlen ermitteln

Parameter:

P_

Worth/Ref	Name	Typ	I/O	Kommentar
	m	Longint	Input	Zahl 1
	n	Longint	Input	Zahl 2
Function	ggt	Real	Output	GGT(Zahl 1, Zahl 2)

Unit **p_debC86**

Enthält Debug-Funktionen für USART (RS232)

==========

procedure **RS232_Init**(iCom: byte; iBaud: word)

Aufgabe: Initialisierung einer seriellen Schnittstelle

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	ICom	Byte	Input	Schnittstelle c_COM1, c_COM2
	IBaud	Word	Input	Baudrate

==========

procedure **WTest** (var iStr: string[50]);

Aufgabe: String-Testausgabe über Usart(RS232)
Gesteuert über Variable TESTMODE

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	IStr	String[50]	Input	Auszugebender Text

==========

procedure **WTest_gng**(var iStr: string[25]; confirm: byte)

Aufgabe: String-Testausgabe über Usart(RS232)
Wenn confirm=C_CONFIRM
Dann weiterlauf im Programm erst nach Tastendruck

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	IStr	String[50]	Input	Auszugebender Text
	Confirm	Byte	Input	Wenn C_CONFIRM Dann Taste für weiterlauf nötig

==========

procedure **Get_DebTaste**(var iTxt: string[25])

Aufgabe: String-Testausgabe über Usart(RS232)
Weiterlauf im Programm erst nach Tastendruck

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Itxt	String[25]	Input	Auszugebender Text

Die folgenden Prozeduren **DxX** ermöglichen die Ausgabe eines Variableninhaltes als Dezimalzahl mit vorausgehendem Text (i_Txt).
Ausnahme: DBXH: Ausgabe als Hexadezimalwert
Beispiel: DRX(‚Real-Zahl=’,Realvariable);

procedure **DDX**(var i_Txt: string[15];
i_b : boolean)

procedure **DBX**(var i_Txt: string[15],
i_b : byte)

procedure **DBXH**(var i_Txt: string[15],
i_b : byte)

procedure **DWX**(var i_Txt: string[15],
i_w : word)

procedure **DIX**(var i_Txt: string[15],
i_i : integer)

procedure **DLX**(var i_Txt: string[15],
i_l : longint)

procedure **DRX**(var i_Txt: string[10];
i_r : real)

==========

procedure **HexDump**(MemMode : byte;
StartAdr: word;
len: byte);

Aufgabe: Hex-Dump eines Speicherbereichs von max 256 Bytes über RS232

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	MemMode	Byte	Input	Wahl des Speichertyps c_SRAM : Ram c_FLASH: Programm-Flash für PIC18F2620 zusäzlich: c_EEPROM
	StartAdr	Word	Input	Startadresse
	Len	Byte	Input	Länge des auszugebenden Speicherbereichs

==========

Die folgenden Prozeduren ermöglichen die Erfassung von Debug-Daten in Echtzeit während des Programmablaufs.
Mit DINIT wird ein Pufferspeicher initialisiert, in den mit **Dx** Variablenwerte und zugehötige Kurztexte erfasst werden können.
In einem zeitunkritischen Programmabschnitt können die Daten mit DOUT per RS232 ausgegeben werden.

Beispiel:

```
DINIT();
.....
DB(‚Variable1=’,Byte_Variable_1);
…….
DL(‘Variable_2=’,Long_Variable_2);
……..
DOUT(c_COM1,’H’) ;  (* Ausgabe in HexaDez-Format *)
```

```
procedure DB(var i_Txt: string;
                 i_b  : byte);
procedure DW(var i_Txt: string;
                  i_w  : word);
procedure DI(var i_Txt: string;
                i_i  : integer);
procedure DL(var i_Txt: string;
                  i_l  : longint);
procedure DR(var i_Txt: string;
                  i_r  : real);
procedure DInit;
procedure DOut(i_Com: byte; DezHex: char);
```

Unit **p_ioPIC18F2620**

Diese Unit enthält Funktionen zur Ansteuerung des Lcd-Moduls durch einen 8bit-Mikrocontroller (18F2620,4620).

Der Funktionsumfang ist aus der Beschreibung der Unit **p_ioPIC24FJ128GB110** ersichtlich.
Erfolgt die Ansteuerung des Lcd-Moduls direkt durch den PIC, so sollte die Unit **p_ioPIC18F2620D** verwendet werden.

Unit **p_ioPIC24FJ128GB110**

Die Unit p_ioPIC24FJ128GB110 ist für eine spezielle Interfacekarte zwischen PIC-Rechner unf Lcd vorgesehen, welche Drehimpulsgeber und Funktionstasten enthält.
(Siehe Schaltplan im Anhang)
Zusätzlich kann mit dieser Karte die Lcd-Ansteuerung über einen X86-Rechner erfolgen.

Soll eine direkte Kopplung von PIC-Rechner und Lcd erfolgen, so muss die modifizierte Unit **p_ioPIC24FJ128GB110D** verwendet werden

==========

procedure **p_ioPic_Init**

Aufgabe: Variablen-Initialisierung für die Unit

SetTextRamStart ($0000)
SetGraphRamStart($0F00)
SetCGRamStart ($7800)

Parameter: keine

==========

procedure **InitPorts**

Aufgabe: Ports für Hardware initialisieren (siehe Schaltplan)

Parameter: keine

==========

procedure **Xreset**

Aufgabe: Reset für Lcd

Parameter: keine

==========

procedure **Check_Adr**

Aufgabe: Prüfung der Adressen-Initialisierung für T6963C

Parameter: keine

==========

procedure **SetTextRamStart** (StartAdr: word)

Aufgabe: Startadresse für TextRam im Textmodus festlegen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	StartAdr	Word	Input	Start-Adresse

==========

procedure **SetGraphRamStart** (StartAdr: word)

Aufgabe: Startadresse für GraphikRam festlegen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	StartAdr	Word	Input	Start-Adresse

==========

procedure **SetCGRamStart** (StartAdr: word)

Aufgabe: Startadresse für CharacterGenerator-Ram festlegen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	StartAdr	Word	Input	Start-Adresse

==========

procedure **MirrorGetMem** (page: byte;
var MirrorRamPtr: t_p;
PageSize: word)

Aufgabe: wenn Heap nicht reicht, dann Sonder-Speicher aktivieren (z.B. SIM-Card) → Funktion vorerst nicht nötig

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	Page	Byte	Input	Mirror-Page
Var	MirrorRamPtr	^t_p	Output	Pointer auf 3840-Byte Bereich
	PageSize	Word	Input	Hier 3840 Bytes

==========

function **Status_ok_extended**: boolean

Aufgabe: T6963C-Status abhängig vom vorausgehendem Kommando prüfen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
function	Status_ok_extended	boolean	Output	True : Funktion ok False: Funktion nicht ok

==========

function **Status_Ok**(Typ: byte): boolean

Aufgabe: T6963C-Status prüfen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	Typ	Byte	Input	STA0, STA1 bei normaler Arbeit STA2 bei Automode Read STA3 bei Automode Write STA6 bei Peek
function	Status_ok	boolean	Output	True : Funktion ok False: Funktion nicht ok

==========

procedure **Write_Command** (Cmd : byte)

Aufgabe: Kommando an T6963C ausgeben

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	Cmd	Byte	Input	T6963C-Kommando-Byte

==========

procedure **Write_Data** (Dat: byte)

Aufgabe: Daten-Byte an .T6963C ausgeben

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar

	Dat	Byte	Input	Daten-Byte

==========

procedure **Write_1_command** (Dat : byte;
Cmd : byte)

Aufgabe: Kommando mit 1 Datenbyte ausgeben

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	Typ	Byte	Input	Daten-Byte
	Cmd	Byte	Input	Kommando-Byte

==========

procedure **Write_2_command** (Dat1: byte;
Dat2: byte;
Cmd : byte)

Aufgabe: Kommando mit 2 Datenbytes ausgeben

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	Dat1	Byte	Input	Daten-Byte 1
	Dat1	Byte	Input	Daten-Byte 1
	Cmd	Byte	Input	Kommando-Byte

Unit **p_ioPIC24FJ128GB110D**

Diese Unit hat die gleichen Funktionen wie **p_ioPIC24FJ128GB110,**
nur entfällt die Interfacekarte zwischen Lcd und Prozessor.
Änderungen ergeben sich damit an der Schnittstelle von Prozessor und Lcd.

Siehe auch Schaltplan

Die folgende Tabelle zeigt die Signalzuordnung, die jedoch angepasst bwerden kann

Lcd-Signal	PIC24FJ128GB110-Pin	PIC24-Signal
D0	72	PD.0
D1	76	PD.1
D2	77	PD.2
D3	78	PD.3
D4	81	PD.4
D5	82	PD.5
D6	83	PD.6
D7	84	PD.7
/RD	69	PD.9
/WR	56	PD.10
RES	79	PD.12
/CE	71	PD.11
C/D	80	PD13
Font	47	PD14
Column	48	PD15

Unit **p_grpC86**

Funktionen für Graphikmodus

Diese Unit stellt die Basisfunktion für die graphische Oberfläche zur Verfügung
- Punkte
- Linien
- Kreise
- Textausgabe
- Paging
- Lcd-Schirm löschen / füllen
- Graphik-Ausschnitte sichern / zurückladen

Speicherorganisation

Der Controller T6963C kann 64kB Graphik-Ram addressieren. Eine Graphik-Seite von 240x128 Pixeln benötigt 3840 Bytes. Bei 32kB Graphik-Ram können damit 8 Graphik-Seiten (Page_0-7) betrieben werden.
Das Auslesen von bereits geschriebenen Graphik-Daten aus dem Ram kann bei manchen Graphik-Modulen Probleme machen. Deshalb kann parallel zum Graphik-Ram ein Mirror-Ram in der CPU eingesetzt werden. In diesen Ram erfolgen Parallel zum T6963C alle Schreibzugriffe, sodaß dieser Ram ein Abbild des Graphik-Ram darstellt. Für den PIC24FJ128GB110 können Page_0 und 1 als Mirror gepflegt werden.

benötigter Ram:
GetImage/PutImage : max 3840 Bytes Heap für eine Graphik-Page

Konstante

Name	**Wert**	**Kommentar**
NormalFont/ DefaultFont	$01	8x8 Font vertikales Schreiben möglich
SmallFont	$02	4x8 Font; Enthält nur die wichtigsten Zeichen für elektrotechnische Masseinheiten vertikales Schreiben nicht möglich
NumFont	$03	3x5 Font; numerische Zeichen vertikales Schreiben möglich
c_LCD	$01	Das Auslesen von Graphikdaten erfolgt aus T6963C-Controller
c_Mirror	$02	Das Auslesen von Graphikdaten erfolgt aus dem Mirror-Ram
HorizDir	$01	Schreibrichtung Horizontal
VertDir	$02	Schreibrichtung Vertikal
NormalPut	$08	1:1 PutImage
XorPut	$09	

NotPut	$0A	Invertierter PutImage
Page_0..7	$00-$07	Konstante für Page-Auswahl
Page_all	$FF	Alle Pages
Black	$0000	Farb-Konstante (**T6963C kann nur schwarz/weiss**)
Blue	$0001	
Green	$0002	
Cyan	$0003	
Red	$0004	
Magenta	$0005	
Brown	$0006	
LightGray	$0007	
DarkGray	$0008	
LightBlue	$0009	
LightGreen	$000A	
LightCyan	$000B	
LightRed	$000C	
LightMagenta	$000D	
Yellow	$000E	
White	$000F	
grOk	0	Fehler-Konstante
grNoInitGraph	-1	
grNotDetected	-2	
grFileNotFound	-3	
grInvalidDriver	-4	
grNoLoadMem	-5	
grNoScanMem	-6	
grNoFloodMem	-7	
grFontNotFound	-8	
grNoFontMem	-9	
grInvalidMode	-10	
grError	-11	
grIOerror	-12	
grInvalidFont	-13	
grInvalidFontNum	-14	
grMaxPage	-15	
grOutOfMaxPage	-16	
grMirrorInactive	-17	
grBarOutOfMax	-18	
grBarMinMax	-19	
grStringToLong	-20	
grOutOfMaxView	-21	
grNoImage	-22	
grNoHeap	-23	
grClip	-24	
grLineStyle	-25	

grNoMirrorPage	-26	
grMaxMirrorPage	-27	
SolidLn	0	Durchgezogene Linie
DottedLn	1	Gepunktete Linie
CenterLn	2	Punkt Strich Punkt
DashedLn	3	Gestrichelte Linie
UserBitLn	4	selbstdefiniert
NormWidth	$01	normale Linienstärke
ThickWidth	$03	Dick
Font_inv	$01	
Font_not_inv	$00	

Graphik-Funktionen/Prozeduren – Borland-Like

==========

function **GraphResult** : integer;

Aufgabe: liefert das Ergebnis einer Graphikfunktion

Parameter: -

==========

function **GraphErrorMsg**(ErrorCode: integer): string[25]

Aufgabe: liefert eine Fehlerstring zum Fehlercode

Parameter: -

Worth/Ref	Name	Typ	I/O	Kommentar
	ErrorCode	Integer	Input	Fehlercode aus GraphResult
function	GraphErrorMsg	String[25]	Output	Fehlerstring

==========

function **GetMaxX**: integer;

Aufgabe: liefert max.Pixelzahl in X-Richtung

Parameter: -

==========

function **GetMaxY**: integer;

Aufgabe: liefert max.Pixelzahl in Y-Richtung

Parameter: -

==========

procedure **SetActivePage** (Page: word)

Aufgabe: auf diese Graphik-Seite erfolgen Schreibzugriffe
Angezeigt wird jedoch die mit SetVisualPage ausgewählte Seite

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Page	Word	Input	Graphik-Seite (0..7)

==========

procedure **SetVisualPage** (Page: word)

Aufgabe: Auswahl der Anzeige-Graphik-Seite

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Page	Word	Input	Graphik-Seite (0..7)

==========

procedure **SetViewPort** (xv,yv,xb,yb: integer;
Clip: Boolean);

Aufgabe: setzt ein Graphik-Zeichenfenster
Alle Graphikbefehle arbeiten relativ zum Zeichenfenster

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	xv	Integer	Input	X-Position für Zeichenfenster (links oben)
	yv	Integer	Input	Y-Position für Zeichenfenster (links oben)
	xb	Integer	Input	X-Position für Zeichenfenster (rechts unten)
	yb	Integer	Input	Y-Position für Zeichenfenster (rechts unten)
	Clip	Boolean	Input	Wenn True dann alle zeichenaktionen am Rand des Fensters abschneiden

==========

procedure **SetColor** (Color: word)

Aufgabe: Farbauswahl für Active-Page
(für T6963C nur Black,White)

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Color	Word	Input	Farbe

==========

procedure **PutPixe**l (x,y: integer; Color: word)

Aufgabe: ein Farbpixel an die Position x,y setzen
(für T6963C nur Black,White)

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	x	Integer	Input	X-Position für Pixel
	y	Integer	Input	Y-Position für Pixel
	Color	Word	Input	Farbe des Pixels

==========

procedure **Rectangle**(xv,yv,xb,yb: Integer);

Aufgabe: ein Viereck zeichnen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	xv	Integer	Input	X-Position für Eck links oben
	yv	Integer	Input	Y-Position für Eck links oben
	xb	Integer	Input	X-Position für Eck rechts unten
	yb	Integer	Input	Y-Position für Eck rechts unten

==========

procedure **Line** (xv: integer;
yv: integer;
xb: integer;
yb: integer)

Aufgabe: eine Linie zeichnen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	xv	Integer	Input	X-Position für Endpunkt links
	yv	Integer	Input	Y-Position für Endpunkt links
	xb	Integer	Input	X-Position für Endpunkt rechts
	yb	Integer	Input	Y-Position für Endpunkt rechts

==========

procedure **Circle**(x0: integer;
y0: integer;
r : word)

Aufgabe: einen Kreis zeichnen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	X0	Integer	Input	X-Position für Mittelpunkt
	Y0	Integer	Input	Y-Position für Mittelpunkt
	r	Integer	Input	Radius in Pixel

==========

procedure **OutTextXY** (x : integer;
y : integer;
var str: string)

Aufgabe: einen Text ausgeben an der Schreibposition x,y
Font,Direction und FontStyle vorher einstellen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	x	Integer	Input	X-Position für Text
	y	Integer	Input	Y-Position für Text
	Str	String	Input	Ausgabetext

==========

procedure **MoveTo** (x,y: integer)

Aufgabe: Graphik-Cursor auf die Position x,y setzen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	x	Integer	Input	X-Position für OutText
	y	Integer	Input	Y-Position für OutText

==========

procedure **OutText** (var str: string);

Aufgabe: einen Text ausgeben an der Position des Graphik-Cursors
Font,Direction und FontStyle vorher einstellen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Str	String	Input	Ausgabetext

==========

procedure **ClearDevice**

Aufgabe: aktive Graphik-Page löschen

Parameter: -

==========

procedure **ClearViewPort**

Aufgabe: aktuellen Viewport löschen (siehe SetViewPort)

Parameter: -

==========

procedure **SetMaxViewPort**

Aufgabe: Viewport auf die gane Page ausdehnen
= SetViewPort(1,1,GetMaxX,GetMaxY,True)

Parameter: -

==========

procedure **Bar** (xv,yv,xb,yb: integer)

Aufgabe: Bar zeichnet ein gefülltes Recteck

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	xv	Integer	Input	X-Position für Endpunkt links
	yv	Integer	Input	Y-Position für Endpunkt links
	xb	Integer	Input	X-Position für Endpunkt rechts
	yb	Integer	Input	Y-Position für Endpunkt rechts

==========

procedure **Bar_Byte** (xv,yv,xb,yb: integer; AnzFillBytes: byte);

Aufgabe: Bar_byte zeichnet ein gefülltes Recteck
Wenn Farbe=Black dann Rechteck löschen
Wenn Farbe nicht Black dann das mit SetFillStyle vorgegeben Pattern verwenden

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	xv	Integer	Input	X-Position für Endpunkt links
	yv	Integer	Input	Y-Position für Endpunkt links
	xb	Integer	Input	X-Position für Endpunkt rechts
	Yb	Integer	Input	Y-Position für Endpunkt rechts
	AnzFillBytes	Byte	Input	Byte oder Word-Ausrichtung für Pattern Mögliche Werte: 1 oder 2

==========

procedure **Bar_Word** (xv,yv,xb,yb: integer)

Aufgabe: Bar_word zeichnet ein gefülltes Recteck
Wenn Farbe=Black dann Rechteck löschen
Wenn Farbe nicht Black dann das mit SetFillStyle vorgegeben Pattern verwenden

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	xv	Integer	Input	X-Position für Endpunkt links
	yv	Integer	Input	Y-Position für Endpunkt links
	xb	Integer	Input	X-Position für Endpunkt rechts
	yb	Integer	Input	Y-Position für Endpunkt rechts

==========

procedure **SetLineStyle** (LineStyle: word;
Pattern : word;
Thickness:word)

Aufgabe: stellt die Parameter für eine Linie ein

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	LineStyle	word	Input	Normale Linie: SolidLn Punktierte Linie: DottedLine
	Pattern	word	Input	nicht verwendet
	Thickness	word	Input	Dicke

				NormWidth = 1 ThickWidth = 3

==========

procedure **GetLineSettings** (var LineInfo: LineSettingsType)

Aufgabe: die mit SetLineStyle eingestellten Parameter auslesen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
Var	LineInfo	LineSettingsType	Output	LineInfo.LineStyle LineInfo.Pattern LineInfo.Thickness

==========

procedure **SetTextStyle** (Font : byte;
Direction : byte;
CharSize : byte)

Aufgabe: Parameter für Textausgabe definieren

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Font	byte	Input	Fontauswahl: DefaultFont, SmallFont, NumFont
	Direction	Byte	Input	Schreibrichtung: HorizDir,VertDir
	CharSize	Byte	Input	Schriftdicke: Nicht verwendet

==========

procedure **SetFillStyle** (Pattern: word;
Color : word)

Aufgabe: Füllmuster für Bar_byte,Bar_word einstellen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Pattern	word	Input	Zu setzendes Pixel mit 1 in word kennzeichnen $FFFF = alles white $0000 = alles black
	Color	word	Input	Farbe des Musters

==========

procedure **PutImage** (x,y : integer;
ImageNr : byte;
BitBlt : word;
freeKz : boolean)

Aufgabe: Füllmuster für Bar_byte,Bar_word einstellen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	x	Integer	Input	X-Positon für Einfüge-Bildausschnitt
	y	Integer	Input	Y-Positon für Einfüge-Bildausschnitt
	ImageNr	byte	Input	Handle des Bildausschnitts aus GetImage
	BitBlt	word	Input	NormalPut Bildausschnitt unverändert zurückladen NotPut: Bildausschnitt invertiert zurückladen
	FreeKz	Boolean	Input	True: Bildausschnitt wird auf Heap gelöscht False: Bildausschnitt bleibt für weiteren Putimage auf Heap erhalten

==========

procedure **GetImage** (xv,yv,xb,yb : integer;
var ImageNr : byte)

Aufgabe: einen Bildausschnitt der ActivePage auf dem Heap sichern

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	xv	Integer	Input	X-Position für Endpunkt links
	yv	Integer	Input	Y-Position für Endpunkt links
	xb	Integer	Input	X-Position für Endpunkt rechts
	yb	Integer	Input	Y-Position für Endpunkt rechts
Var	ImageNr	Byte	OutPut	Handle des Bildausschnitts

==========

function **ImageSize** (xv,yv,xb,yb : integer): word

Aufgabe: Bestimmen der Größe eines Bildausschnitt der ActivePage in Bytes

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	xv	Integer	Input	X-Position für Endpunkt links
	yv	Integer	Input	Y-Position für Endpunkt links
	xb	Integer	Input	X-Position für Endpunkt rechts
	yb	Integer	Input	Y-Position für Endpunkt rechts
function	ImageSize	Word	Output	Bildgröße in Bytes

Graphik-Funktionen/Prozeduren – nicht Borland

==========

procedure **InitGraph**

Aufgabe: Graphikmodus aktivieren
Es wird auch geprüft, ob der Lcd-Inhalt ausgelesen werden kann.
Ist dies nicht möglich, so wird der Mirror-Ram gestartet, der dann das Auslesen der Graphikdaten ermöglicht

Parameter: -

==========

procedure **p_grpC86_Init**

Aufgabe: Unit-Variablen initialisieren
Muss vor InitGraph erfolgen

Parameter: -

==========

procedure **ClearLCD**

Aufgabe: ganze ActivePage löschen

Parameter: -

==========

procedure **GetActivePage**(var page: byte)

Aufgabe: aktuelle ActivePage ermitteln

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
var	Page	Byte	OutPut	Aktive Page

==========

procedure **GetVisualPage**(var page: byte)

Aufgabe: aktuelle VisualPage ermitteln

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
var	Page	Byte	OutPut	VisualPage

==========

procedure **SetFont**(font: byte)

Aufgabe: Font für OutTextXY einstellen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Font	Byte	Input	NormalFont,DefaultFont,SmallFont,NumFont

==========

procedure **SetDirection**(Direction: byte)

Aufgabe: Schreibrichtung für OutTextXY einstellen
Horizontal, Vertikal
Vertikal nicht für SmallFont vorgesehen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Direction	Byte	Input	HorizDir,VertDir

==========

procedure **GetDirection**(var Direction: byte)

Aufgabe: aktuelle Schreibrichtung ermitteln

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
var	Direction	Byte	OutPut	HorizDir,VertDir

==========

procedure **SetFontStyle** (Style: byte)

Aufgabe: invertiertes , nicht invertiertes Schreiben einstellen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Style	Byte	Input	Font_not_inv, Font_Inv

==========

procedure **GetFontStyle** (var Style: byte)

Aufgabe: aktuellen Fontstyle ermitteln

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
var	Style	Byte	Output	Font_not_inv, Font_Inv

==========

procedure **SetPixel** (x,y: integer)

Aufgabe: einzelnes Pixel relativ zu Viewport in Lcd setzen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	X	Integer	Input	X-Pixelkoordinate relativ zu Viewport
	Y	Integer	Input	Y-Pixelkoordinate relativ zu Viewport

==========

procedure **ClearPixel** (x,y: integer)

Aufgabe: einzelnes Pixel relativ zu Viewport in Lcd löschen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	X	Integer	Input	X-Pixelkoordinate relativ zu Viewport
	Y	Integer	Input	Y-Pixelkoordinate relativ zu Viewport

==========

procedure **UnLine** (xv: integer;
yv: integer;
xb: integer;
yb: integer)

Aufgabe: Linie löschen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	xv	Integer	Input	X-Position für Endpunkt links
	yv	Integer	Input	Y-Position für Endpunkt links
	xb	Integer	Input	X-Position für Endpunkt rechts

	yb	Integer	Input	Y-Position für Endpunkt rechts

==========

procedure **TCircle**(x0 : integer;
y0 : integer;
r : word;
rp : byte;
lro : byte)

Aufgabe: Teilkreis zeichnen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	X0	Integer	Input	Mittelpunkt-X-Pixelkoordinate relativ zu Viewport
	Y0	Integer	Input	Mittelpunkt-Y-Pixelkoordinate relativ zu Viewport
	r	Word	Input	Radius
	rp	Byte	Input	Wieviel Prozent des Gesamtkreises soll Teilkreis sein
	lro	Byte	Input	In welche Richtung soll Teilkreis schauen: c_links, c_rechts, c_oben

Graphik-Funktionen/Prozeduren – sonstig

==========

procedure **OpenRWLcdRam** (RamAddr: word)

Aufgabe: Graphik AddressPointer in Graphik-Ram setzen
(Graph_Adress_Ptr_Set)
ab dieser Adresse erfolgen Zugriffe auf Graphik-Ram

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	RamAddr	Word	Input	Word-Adresse

==========

procedure **WriteContLcdRam** (ib: byte)

Aufgabe: Datenbyte ib in Graphik-Ram schreiben
Graphik-Pointer autom. Incrementieren für nächstes Schreiben
(AutoIncrement)

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	ib	byte	Input	Datenbyte für Graphik-Ram

==========

procedure **WriteLcdRam** (ib: byte)

Aufgabe: Datenbyte ib in Graphik-Ram schreiben
(ohne AutoIncrement)
Adresse muss vorher mit OpenRWLcdRam eingestellt worden sein

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	ib	byte	Input	Datenbyte für Graphik-Ram

==========

procedure **ReadLcdRam** (var ib: byte)

Aufgabe: Datenbyte ib aus Graphik-Ram lesen
Adresse muss vorher mit OpenRWLcdRam eingestellt worden sein

Beispiel: Berechne_RamAddr (100,100,RamAddr)
OpenRWLcdRam (RamAddr)
ReadLcdRam (ib)

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
var	ib	byte	Output	Datenbyte aus Graphik-Ram

==========

procedure **Berechne_RamAddr** (x,y: integer;
var RamAddr: word)

Aufgabe: Adresse in Graphik-Ram für x,y-Position berechnen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
var	ib	byte	Output	Datenbyte aus Graphik-Ram

==========

procedure **Berechne_GraphRamStartEnde**(Page: byte;
var PStart: word;
var PEnde: word)

Aufgabe: Start-und Ende-Adresse einer Page im Graphik-Ram berechnen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Page	byte	Input	Page_0..7
Var	PStart	Word	Output	Startadresse
Var	PEnde	Word	Output	Endeadresse

==========

procedure **Graph_Adress_Ptr_Set** (hb: byte;
lb : byte)

Aufgabe: Graphik-Adress-Pointer laden

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	hb	byte	Input	High-Byte der Adresse
	lb	byte	Input	Low-Byte der Adresse

==========

procedure **OpenMirrorRam**(page: byte)

Aufgabe: Mirror-Ram für eine Graphik-Page auf dem Heap eröffnen
Für PIC24FJ128GB110 nur Page_0,1 möglich

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Page	byte	Input	Page_0,1

==========

procedure **CloseMirrorRam**(page: byte)

Aufgabe: Mirror-Ram closen. Heap freigeben

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Page	byte	Input	Page_0,1

==========

vprocedure **StartMirrorRam**(page: byte)

Aufgabe: Open auf Mirror-Ram mit zusätzlichen Prüfungen
Nur mit dieser Funktion soll Mirror-Ram gestartet werden

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Page	byte	Input	Page_0,1

==========

procedure **StopMirrorRam**(page: byte)

Aufgabe: Close auf Mirror-Ram; Gegenstück zu StartMirrorRam

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Page	byte	Input	Page_0,1

---

procedure **InitReadWriteMirrorRam**(page: byte)

Aufgabe: Close auf Mirror-Ram; Gegenstück zu StartMirrorRam

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Page	byte	Input	Page_0,1

==========

procedure **ReadMirrorRam**(page: byte; RamAddr: word; var SByte: byte)

Aufgabe: Lesen eines Bytes aus dem Mirror-Ram

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Page	byte	Input	Page_0,1
	RamAddr	word	Input	Adresse im Mirror-Ram Die Adresse kann aus der Adresse für den Lcd-Ram berechnet werden z.B. Berechne_RamAddr(zxv,zyv,RamAddr); Berechne_MirrorAddr(RamAddr,MirrorAddr);
var	SByte	byte	Input	Gelesenes Byte

==========

procedure **WriteMirrorRam**(page: byte; RamAddr: word; b: byte)

Aufgabe: Schreiben eines bytes in den Mirror-Ram

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Page	byte	Input	Page_0,1
	RamAddr	word	Input	Adresse im Mirror-Ram Die Adresse kann aus der Adresse für den Lcd-Ram berechnet werden z.B. Berechne_RamAddr(zxv,zyv,RamAddr); Berechne_MirrorAddr(RamAddr,MirrorAddr);
	SByte	byte	Input	Zu schreibendes Byte

==========

procedure **GetReadMode** (var ReadMode: byte)

Aufgabe: Lesen Readmode
Der Readmode enthält die Information ob Lesezugriff auf Lcd-Ram oder Mirror-Ram erfolgen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
var	Readmode	byte	Input	C_Lcd, c_Mirror

==========

procedure **CopyPage**(FromPage,ToPage: byte)

Aufgabe: kompletten Inhalt einer Graphik-Seite in eine andere Seite kopieren

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Frompage	byte	Input	Quell-Seite
	ToPage	Byte	Input	Ziel-Seite

==========

procedure **CopyPageParts**(FromPage : byte;
fxv,fyv,fxb,fyb: integer;
ToPage : byte;
txv,tyv : integer)

Aufgabe: Teil-Inhalt einer Graphik-Seite in eine andere Seite kopieren

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	Frompage	byte	Input	Quell-Seite
	fxv	integer	Input	X-Position für Endpunkt links
	fyv	integer	Input	Y-Position für Endpunkt links
	fxb	integer	Input	X-Position für Endpunkt rechts
	fyb	integer	Input	Y-Position für Endpunkt rechts
	ToPage	Byte	Input	Ziel-Seite
	txv	integer	Input	Ziel-X-Position
	txb	integer	Input	Ziel-Y-Position

==========

procedure **transform_pixelvalue**(Pin : integer;
mode : byte;
var Pout: integer)

Aufgabe: Pixelangabe an Bytegrenze ausrichten

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	pin	byte	Input	Pixel-Nr z.B. Pixel 355 (Zählung beginnt bei 1)
	mode	integer	Input	C_up: Pixel auf nächst größeres Byte ausrichten C_down: Pixel auf nächst kleineres Byte ausrichten
var	pout	integer	Input	Nächstes an Byte-Grenze ausgerichtes Pixel ist entweder 352(c_down) oder 360(c_up)

Unit **p_sklC86**

Diese Unit stellt Funktionen für die Skalierung von Signalanzeigen zur Verfügung.
Es können unabhängig voneinander X-und Y-Skalierungen vorgenommen werden.

==========

procedure **p_sklC86_Init**

Aufgabe: Variablen-Initialisierung für die Unit

Parameter: keine

==========

procedure **GetSkalWin**(var Obj: string[5];
ExtWin: byte;
var IntWin: byte)

Aufgabe: Für MD2,MD3,SPG können gleichzeitig max. 8 Skalierungen definiert werden. Je nach Obj wird wird der Angabe eines externen Windows ein internes Window zugeordnet.
Mit dieser internen Window-Nr wird weitergearbeitet.
Damit können gleichzeitig skaliert werden:
2 MD3-Signal-Windows
4 MD2-Signal-Windows
1 SPG-Signal-Window

Obj	ExtWin	IntWin
MD3	1	1
MD3	2	2
MD2	1	3
MD2	2	4
MD2	3	5
MD2	4	6
SKL	1	7

Parameter:

.

Worth/Ref	Name	Typ	I/O	Kommentar
var	Obj	String[5]	Input	Textkürzel für Einsatz der Skalierung **MD2** p_md2C86 **MD3** p_md3C86 **SPG** p_spgC86
	ExtWin	byte	Input	Window-Nr innerhalb Obj
var	IntWin	byte	Output	zu verwendende Window-Nr für Funktionen

==========

procedure **Set_SkalParam_1** (w: byte;
xv,yv,xb,yb: integer;
Xmin,Ymin,Xmax,Ymax: real;
dx_skal,dy_skal: real;
x0,y0: byte;
h_farbe,k_farbe,s_farbe:word)

Aufgabe: Übernahme der Skalierungs-Parameter in int.Struktur

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	w	String[5]	Input	Window-Nr (=IntWin)
	xv	Integer	Input	xv Pixelangabe des Anzeige-Windows
	yv	Integer	Input	yv Pixelangabe des Anzeige-Windows
	xb	Integer	Input	xb Pixelangabe des Anzeige-Windows
	yb	Integer	Input	yb Pixelangabe des Anzeige-Windows
	Xmin	Real	Input	Min-Wert auf X-Achse
	Ymin	Real	Input	Min-Wert auf Y-Achse
	Xmax	Refontzal	Input	Max-Wert auf X-Achse
	Ymax	Real	Input	Max-Wert auf Y-Achse
	dx_skal	Real	Input	Schrittweite der X-Skalierung
	dy_skal	Real	Input	Schrittweite der Y-Skalierung
	x0	byte	Input	Achsabstand zu xv (siehe Skizze)
	y0	byte	Input	Achsabstand zu yb (siehe Skizze)
	H_farbe	word	Input	Hintergrund-Farbe (vorerst nicht verwendet)
	K_farbe	word	Input	Signal-Farbe (vorerst nicht verwendet)
	S_farbe	word	Input	Skalierungs-Farbe

==========

procedure **Set_XY_Nk** (XNk,YNk : byte;
X_XVersatz,X_YVersatz: integer;
Y_XVersatz,Y_YVersatz: integer);

Aufgabe: Nachkommastellen und x,y-Versatz der Skalierungswerte-Beschriftung

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	XNk	byte	Input	Nachkommastellen X-Skalierung
	YNk	byte	Input	Nachkommastellen Y-Skalierung
	X_XVersatz	integer	Input	X-Skalierung: Versatz in X-Richtung
	X_YVersatz	integer	Input	X-Skalierung: Versatz in Y-Richtung
	Y_XVersatz	integer	Input	Y-Skalierung: Versatz in X-Richtung
	Y_YVersatz	integer	Input	Y-Skalierung: Versatz in Y-Richtung

==========

procedure **YSkalierung** (w: byte)

Aufgabe: Skalierung der Y-Achse durchführen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr (=IntWin)

==========

procedure **XSkalierung** (w: byte)

Aufgabe: Skalierung der X-Achse durchführen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr (=IntWin)

==========

procedure **XRealToPixel** (w : byte;
wert : real;
var x: integer);

Aufgabe: X-Analogwert in Pixel umrechnen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr (=IntWin)
	wert	real	Input	Analog-Wert in Pixel umrechnen
Var	x	integer	Output	X-Pixelwert innerhalb Anzeige-Window

==========

procedure **YRealToPixel** (w : byte;
wert : real;
var y: integer);

Aufgabe: Y-Analogwert in Pixel umrechnen

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr (=IntWin)
	wert	real	Input	Analog-Wert in Pixel umrechnen
Var	y	integer	Output	Y-Pixelwert innerhalb Anzeige-Window

==========

procedure **X0_Line** (w: byte; var pX0: byte)

Aufgabe: Berechnet Versatz der X-0-Linie relativ zu Window-Grenzen abhängig von Xmin,Xmax-Werten

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr (=IntWin)
Var	pY0	byte	Output	Y-Pixelwert-Versatz relativ zu yb innerhalb Anzeige-Window

==========

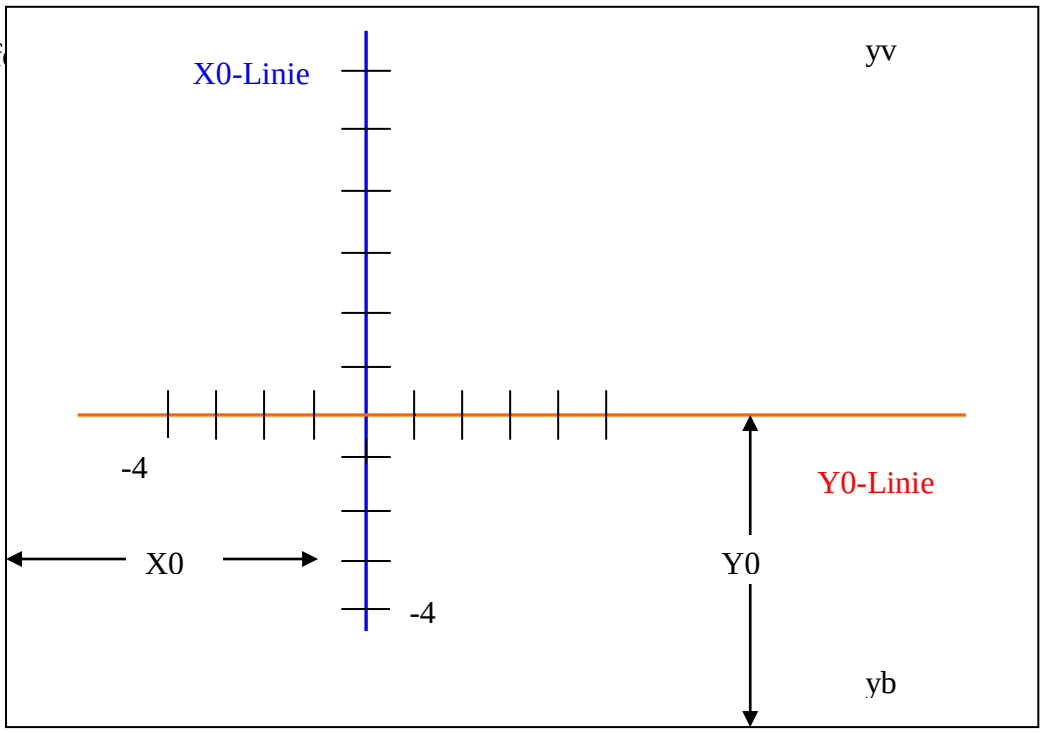
procedure **Y0_Line** (w: byte; var pY0: byte)

Aufgabe: Berechnet Versatz der Y-0-Linie relativ zu Window-Grenzen abhängig von Ymin,Ymax-Werten

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr (=IntWin)
Var	pY0	byte	Output	Y-Pixelwert-Versatz relativ zu yb innerhalb Anzeige-Window wird abhängig von Ymax,min berechnet

Das f



==========

procedure **Get_SkalParam_2** (w: byte;
var qxwert,qywert: real);

Aufgabe: Berechnet Faktor, der die Zuordnung des anzuzeigenden Wertebereichs zum Pixelbereich definiert.

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr (=IntWin)
Var	qXwert	Real	Output	qXwert = (Xmax-Xmin) / (xb-xv)
Var	qYwert	Real	Output	qYwert = (Ymax-Ymin) / (yb-yv)

==========

procedure **Neg_YSkalierung** (w: byte)

Aufgabe: Skaliert Y-Achse mit negativ-Werten.

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr (=IntWin)

==========

procedure **Pos_YSkalierung** (w: byte; mode: byte)

Aufgabe: Skaliert Y-Achse mit pos.Werten.
Wenn Ymin > 0 dann Start bei Ymin → c_OnlyPosSkal
Wenn Ymin <= 0 dann Start bei 0. → c_mixedSkal

Parameter:

Worth/Ref	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr (=IntWin)
	mode	Byte	Input	c_OnlyPosSkal = 1; c_mixedSkal = 2;

Beispiel: siehe Toolbox_24_Demo1

Unit **p_larC86**

p_larC86 enthält Routinen zur Digital-Meßwertdarstellung mit Cursoreinblendung

- max. 8 Kanäle mit Beschriftung
- max. 240 Werte auf X-Achse
- keine Kompressionsfunktion
- unterstützt Skalierung (Grid)

==========

procedure **p_larC86_Init**

Aufgabe: Variablen-Initialisierung der Unit

Parameter: keine

==========

Procedure **Open_dig_kurve** **(**w : byte;
WertY_max : real;
WertY_min : real;
XInd_max : word;
XInd_min : word;
XUnit : byte;
AnzChannels : byte;
wi_xv : integer;
wi_yv : integer;
wi_xb : integer;
wi_yb : integer;
hinter_farbe : word;
kurven_farbe1 : word;
kurven_farbe2 :word;
skalen_farbe :word;
var UeText :string[10];
Skalierung :byte;
dx_skal :real;
dy_skal :real)

Aufgabe: Window für Digital-Anzeige (Bitmuster) öffnen
Es können max. 2 Windows eröffnet werden

Parameter:

.

Worth/Ref.	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr max. 2 Windows möglich

	WertY_max	real	Input	max. Y-Wert
	WertY_min	real	Input	Min. Y-Wert
	XInd_max	real	Input	max. X-Wert
	XInd_min	real	Input	Min. X-Wert
	Xunit	Byte	Input	Sampling-Zeiteinheit C_sec, c_ms, c_us, c_ns
	AnzChannels	byte	Input	Anzahl der Digital-Kanäle / max. 8
	w_xv	integer	Input	xv-Koordinaten des Anzeige-Window
	w_yv	integer	Input	yv- --„--
	w_xb	integer	Input	xb- --„--
	w_xb	integer	Input	yb- --„--
	hinter_farbe	word	Input	Hintergrund-Farbe
	kurven_farbe1	word	Input	Signal-Farbe 1
	kurven_farbe2	word	Input	Signal-Farbe 2 (vorerst nicht verwendet)
	skalen_farbe	word	Input	Skalierungs-Farbe
var	UeText	String[20]	Input	Überschrift
	skalierung	byte	Input	Welche Skalierung wird gewünscht c_skal_no = 0 keine Skalierung c_skal_x = 1 X-Skalierung c_skal_y = 2 Y-Skalierung nur sinnvol bei nur 1 Kanal c_skal_xy = 3 XY-Skalierung
	dx_skal	real	Input	X-Skalierung (z.B. 10 ➔ 10 uS-Schritte)
	dy_skal	real	Input	Y-Skalierung (zB. 1 ➔ 1Volt-Schritte)

==========

Procedure **Set_BitText** (var bs0,bs1,bs2,bs3,bs4,bs5,bs6,bs7: string[5]);

Aufgabe: Bezeichnungen der Kanäle (max. 8)
Muss vor Open aktiviert werden

Parameter:

.

Worth/Ref.	Name	Typ	I/O	Kommentar
var	bs0	String[5]	Input	Bezeichnung Kanal 0
var	bs1	String[5]	Input	Bezeichnung Kanal 1
var	bs2	String[5]	Input	Bezeichnung Kanal 2
var	bs3	String[5]	Input	Bezeichnung Kanal 3
var	bs4	String[5]	Input	Bezeichnung Kanal 4
var	bs5	String[5]	Input	Bezeichnung Kanal 5
var	bs6	String[5]	Input	Bezeichnung Kanal 6
var	bs7	String[5]	Input	Bezeichnung Kanal 7

==========

procedure **Close_dig_kurve** (w: byte)

Aufgabe: Window für Digitalanzeige schließen

Parameter:

.

Worth/Ref.	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr (1 oder 2)

==========

procedure **dig_kurve** (w : byte;
XInd : word;
DigByte : byte)

Aufgabe: Bezeichnungen der Kanäle (max. 8)

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	W	byte	Input	Window-Nr
	Xind	Word	Input	Anzeigeindex
	DigByte	Byte	Input	Digitaler Wert für 8 Kanäle

==========

procedure **Open_dig_cursor** (w: byte)

Aufgabe: Open graphische Indexanzeige unter der Digitalanzeige

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr (1 oder 2)

==========

procedure **dig_cursor**(w : byte;
var XInd : word)

Aufgabe: graphische Indexanzeige setzen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr

	Xind	Word	Input	Position in Anzeigewindow

==========

procedure **Close_dig_cursor** (w: byte)

Aufgabe: graphische Indexanzeige löschen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr

==========

procedure **open_lar_cursor** (w : byte)

Aufgabe: Cursorlinie über alle Kanäle eröffnen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr

==========

procedure **lar_cursor** (w : byte;
rel_x : integer)

Aufgabe: Cursorlinie auf zeichnen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	w	Byte	Input	Window-Nr
	Rel_x	Integer	Input	Linie auf Index rel_x eichnen

==========

procedure **close_lar_cursor** (w: byte);

Aufgabe: Cursorlinie löschen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	w	byte	Input	Window-Nr

Unit **p_md3C86**

Die Unit p_md3C86 enthält Routinen zur analogen Meßwertdarstellung
inclusive Cursorfunktion .

Funktionalität

- max. 240 Werte auf X-Achse
- 2 Windows möglich
- 2 Kurven pro Window möglich
- Messcursor einblendbar
- unterstützt Grid

==========

procedure **p_md3C86_Init**

Aufgabe: Variablen-Initialisierung der Unit

Parameter: keine

==========

```
procedure open_md3_kurve(w              : byte;
                         WertY_max      : real;
                         WertY_min      : real;
                         WertX_max      : real;
                         WertX_min      : real;
                         w_xv,w_yv      : integer;
                         w_xb,w_yb      : integer;
                         hinter_farbe   :  word;
                         kurven_farbe1  : word;
                         kurven_farbe2  : word;
                         skalen_farbe   : word;
                         dx_skal        : real;
                         dy_skal        : real;
                         SampleIntervall : real;
                         SampleUnit     : byte;
                         X_ValueUnit    : byte;
                         Y_ValueUnit    : byte;
                         Store          : boolean;
                         XY_Display     : boolean;
                         ShowNum        : boolean;
                         font           : byte;
                         Save_X_1       : boolean;
                         Save_Y_1       : boolean;
                         Save_X_2       : boolean;
                         Save_Y_2       : boolean;
                         var UeText     : string[20]);
```

Aufgabe: Öffnen der Analoganzeige

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	W	byte	Input	Window-Nr
	WertY_max	real	Input	Max.Wert auf Y-Achse
	WertY_min	real	Input	Min.Wert auf Y-Achse
	WertX_max	real	Input	Max.Wert auf X-Achse
	WertX_min	real	Input	Min.Wert auf X-Achse
	wxv	integer	Input	Pixel-Koordinate
	wyv	integer	Input	Pixel-Koordinate
	wxb	integer	Input	Pixel-Koordinate
	wyb	integer	Input	Pixel-Koordinate
	hinter_farbe	word	Input	Hintergrund-Farbe (black)
	kurven_farbe1	word	Input	Zeichen-Farbe-1 (white)
	kurven_farbe2	word	Input	Zeichen-Farbe-2 (white)
	skalen_farbe	word	Input	Skalier-farbe (white)
	dx_skal	real	Input	Rasterung der X-Skalierung
	dy_skal	real	Input	Rasterung der Y-Skalierung
	SampleIntervall	real	Input	Abtastintervall (z.B. 10000/sec)
	SampleUnit	byte	Input	Zeiteinheit
	X_ValueUnit	byte	Input	X-Anzeige-Zeiteinheit (c_sec,c_ms,...)
	Y_ValueUnit	byte	Input	X-Anzeige-Zeiteinheit (c_V,...)
	Store	boolean	Input	Wert-speichern aktivieren
	XY_Display	boolean	Input	XY-Display; Werte als Punkte anzeigen (Funktion vorerst nicht unterstützt)
	ShowNum	boolean	Input	Werte im Window rechts oben anzeigen
	font	byte	Input	Font für Werteanzeige
	Save_X_1	boolean	Input	X1-Wert speichern ; nötig für XY_Display
	Save_Y_1	boolean	Input	Y1-Wert speichern ; nötig für Cursor
	Save_X_2	boolean	Input	X2-Wert speichern ; nötig für XY_Display
	Save_Y_2	boolean	Input	Y2-Wert speichern ; nötig für Cursor
var	UeText	String[20]	Input	Überschrifts-Text

==========

procedure **close_md3_kurve** (w: byte)

Aufgabe: Analog-Anzeige löschen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	W	byte	Input	Window-Nr

==========

procedure **write_md3_kurve** (w : byte;
mode : byte;
wert_x1 : real;
wert_y1 : real;
wert_x2 : real;
wert_y2 : real)

Aufgabe: Analog-Signal-Werte zeichnen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	W	byte	Input	Window-Nr
	mode	byte	Input	Darstellungsmode ➔ c_NextValue : komplette Signalkurve beginned bei WertX_min bis WertX_max ausgeben ➔c_RandomValue neuen Signalwert an beliebige X-Position ausgeben. Der alte Wert wird überschrieben. Die Lines werden angepasst
	wert_x1	real	Input	X-Samplingzeit
	wert_y1	real	Input	Y1-Analogwert
	wert_x2	real	Input	X-Samplingzeit
	wert_y2	real	Input	Y2-Analogwert

==========

procedure **open_cursor** (w : byte)

Aufgabe: Cursorlinie eröffnen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	W	byte	Input	Window-Nr

==========

procedure **write_cursor** (w : byte;
xr: real);

Aufgabe: Cursorlinie auf X-Wert xr positionieren

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	W	byte	Input	Window-Nr

	Xr	Real	Input	Sampling-Zeit

==========

procedure **close_cursor** (w: byte)

Aufgabe: Cursorlinie löschen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	W	byte	Input	Window-Nr

==========

procedure **open_anl_cursor** (w: byte)

Aufgabe: Cursor für Samplingzeit eröffnen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	W	byte	Input	Window-Nr

==========

procedure **write_anl_cursor**(w : byte;
xr: real)

Aufgabe: Cursor für Samplingzeit positionieren

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	W	byte	Input	Window-Nr
	Xr	Real	Input	Sampling-Zeit

==========

procedure **close_anl_cursor** (w: byte)

Aufgabe: Samplingzeit-Cursor löschen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	W	byte	Input	Window-Nr

Unit **p_segC86**

Zur Auswahl von Parameter für Messvorgänge können Listboxen verwendet werden.
Eine Listbox kann max.5 Einträge als String enthalten. Der Gerade aktive Eintrag wird invertiert angezeigt und sein Index zurückgegeben.

Beispiel Pseudocode:
```
  Open_Listbox
   While (kein Select-Tastendruck am Drehimpulsgeber)
      Get_Drehimpulsgeber
      If c_up or c_down then Update_Listbox
      If Select-Tastendruck then Listind übernehmen
                                    Close_Listbox
                                         Oder
                                    Weitere Listbox öffnen
    Endwhile
```

==========

procedure **p_segC86_Init**

Aufgabe: Initialisierung der Unit
Muss vor dem ersten Open erfolgen

Parameter –

==========

procedure **Open_Listbox**(var handle : byte;
xv : integer;
yv : integer;
xb : integer;
AnzLines: byte;
Font : byte;
var z1 : string[10];
var z2 : string[10];
var z3 : string[10];
var z4 : string[10];
var z5 : string[10

Aufgabe: Listbox für Parameterauswahl eröffnen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
Var	Handle	Byte	Output	Handle für aktuelle Listbox Max.3 Listboxen gleichzeitig möglich
	xv	byte	Input	Xvon-Pixel des Listbox-Windows

	yv		Input	Yvon-Pixel des Listbox-Windows
	xb		Input	Xbis-Pixel des Listbox-Windows
	Anzlines		Input	Anzahl Parameterzeilen (max. 5)
	Font		Input	Font für Parameterzeile
var	Z1		Input	Parameterteile 1
var	Z2		Input	Parameterteile 2
var	Z3		Input	Parameterteile 3
var	Z4		Input	Parameterteile 4
var	Z5		Input	Parameterteile 5

==========

procedure **Close_Listbox**(handle : byte)

Aufgabe: Listbox für Parameterauswahl schliessen
Listboxen dürfen nur nach den Stackprinzip geschlossen werden. Zuletzt geöffnete muss als erste geschlossen werden.

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	Handle	Byte	Input	Handle für Listbox

==========

procedure **Update_Listbox**(handle : byte;
UpDown : byte;
var ListInd: byte

Aufgabe: Parameter aus Z1..Z5 aktivieren. Wird invertiert dargestellt.

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	Handle	Byte	Input	Handle für Listbox Nur die zuletzt geöffnete Listbox kann aktiv sein
	UpDown	Byte	Input	Ausgehend vom gerade aktiven Parameter wird der nächste oberhalb oder unterhalb von Zi aktiviert C_Up oder c_Down kann von einem Drehimpulsgeber oder Tastatur kommen
Var	ListInd	Byte	Output	Nach Close Index des zuletzt aktiven Parameters

Unit **p_spgC86**

Diese Unit unterstützt die Darstellung eines Spektrums..
Auf der X-Achse wird die Frequenz skaliert, auf der Y-Achse die Spannung.

==========

procedure **p_spgC86_Init**

Aufgabe: Initialisierung der Unit
Muss vor dem Open erfolgen

Parameter –

==========

procedure **OpenXSpectrum** (w_xv : integer;
w_yv : integer;
w_xb : integer;
w_yb : integer;
nl_x0 : byte;
nl_y0 : byte;
hinter_farbe : word;
kurve_farbe : word;
skalen_farbe : word;
WertX_max : real;
WertX_min : real;
WertY_max : real;
WertY_min : real;
XUnit : byte;
YUnit : byte;
mo_skal : byte;
dx_skal : real;
dy_skal : real;
var UeText : string[20]);

Aufgabe: Spektrum-Anzeige öffnen.

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	w_xv	integer	Input	Window-Koordiunate
	w_yv	integer	Input	Window-Koordiunate
	w_xb	integer	Input	Window-Koordiunate
	w_yb	integer	Input	Window-Koordiunate
	nl_x0	byte	Input	Siehe Set_SkalParam_1
	nl_y0	byte	Input	Siehe Set_SkalParam_1
	hinter_farbe	word	Input	Hintergrund-Farbe (black bei s/w)g
	kurve_farbe	word	Input	Zeichen-farbe (white)

	skalen_farbe	word	Input	Skalierfarbe (white bei s/w)
	WertX_max	real	Input	max.X-Wert
	WertX_min	real	Input	min X-Wert
	WertY_max	real	Input	max.X-Wert
	WertY_min	real	Input	min Y-Wert
	XUnit	byte	Input	Einheit auf X-Achse (z.B. c_kHz)
	YUnit	byte	Input	Einheit auf X-Achse (z.B. c_V)
	mo_skal	byte	Input	Mit/ohne Skalierung Mit: Skalierung über p_sklC86 Ohne: keine Skalierung
	dx_skal	real	Input	Skalierabstand auf X-Achse Beispiel: WertX_min = 0 WertX_max= 100 kHz Dx_skal = 20 ➔ Skalierabstand 20kHz
	dy_skal	real	Input	Skalierabstand auf Y-Achse Beispiel: WertY_min = 0 WertY_max= 5V Dy_skal = 1 ➔ Skalierabstand 1 Volt
var	UeText	String[20]	Input	Überschtift für Window

==========

procedure **WriteXSpectrum**(XWert: Real;
YWert: Real);

Aufgabe: einen Spektrumswert (Frequenzlinie) anzeigen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	Xwert	Real	Input	Frequenzangabe Muss innerhalb von WertX_min und WertX_max liegen
	Ywert	Real	Input	Amplitude Muss innerhalb von WertY_min und WertY_max liegen

==========

procedure **CloseXSpectrum**(xvo,yvo,xbo,ybo: integer);

Aufgabe: Spektrum-Anzeige schliessen

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	xvo	Integer	Input	xvo- Koordinate des Windows
				yvo- Koordinate des Windows

				xbo-Koordinate des Windows
				ybo- Koordinate des Windows

Unit **p_icoC86**

Diese Unit unterstützt die Darstellung von max. 255 Icons.
Jedes Icon wird in Form einer Bitmap als Include eingebunden.

Für ein 240x128-Lcd werden max 3840+2 bytes benötigt. Die ersten 2 Bytes enthalten Anzahl Pixel in X-und Y-Richtung.

X-Anzahl Pixel
Y-Anzahl Pixel
Bitmap-Byte
...
Bitmap-Byte

==========

procedure **p_icoC86_Init**

Aufgabe: Initialisierung der Unit
Muss vor dem Open erfolgen

Parameter –

==========

procedure **WriteIcon** (x,y: integer;
IconIndex: byte);

Aufgabe: Ausgabe des Icons

Parameter:

Worth/Ref.	Name	Typ	I/O	Kommentar
	x	Integer	Input	Linke obere X-Bildkoordinate
	y	Integer	Input	Linke obere Y-Bildkoordinate
	IconIndex	byte		Nr des Icons in Icontabelle **IconAdrTab**

==========

procedure **WriteIcon_invert** (x,y: integer;
IconIndex: byte);

Aufgabe: Invertierte Ausgabe des Icons; sonst wie WriteIcon

Parameter wie WriteIcon

Unit **p_adPIC24FJ128GB110**

Diese Unit unterstützt die Erfassung und Pufferung von Analogspannungen.
Die Pin-Ansteuerung und Registerbelegung ist hier speziell für den Prozessor PIC24FJ128GB110 codiert, kann aber für andere PIC24-Prozessoren modifiziert werden.

Für die Datenerfassung bieten sich 2 Varianten an:
- „manuelle“ AD-Erfassung durch zeitlich nicht exakt gesteuerten Aufruf im Programm
- Timer-Interrupt-gesteuerte AD-Erfassung

Für die schnelle Pufferung der Daten können dynamisch max 2 Ringpuffer angelegt werden.
Eine größere Anzahl Ringpuffer ist abhängig von der Heapgröße.
Ein Ringpuffer hat die Breite WORD und unterstützt jeweils 1 Analogkanal.

„Manuelle“ Erfassung:
:Diese Art der Datenerfassung ist für schnelle AD-Erfassungen sinnvoll. Mit einer Delay-Schleife kann annähernd ein exaktes Timing erzielt werden.

Beispiel einer schnellen Erfassung in Pseudocode:

```
Open AD
Open Ringpuf
(* Erfassen in Ringpuffer *)
While (Ringpuf  noch nicht gefüllt)
        Get_AD (mit/ohne Delay)
        Save in Ringpuf
Endwhile

(* Auslesen aus Ringpuffer *)
While (Ringpuf  nicht ausgelesen)
        Read Wert
        Anzeigen in Graphik
Endwhile
```

Interrupt-gesteuerte Erfassung:

Für zeitlich größere Abstände einer AD-Erfassung ist eine Timer-Steuerung sinnvoll.
Hier kann über den Ringpuffer ein verzahntes Schreiben und Lesen erfolgen.

Beispiel einer Erfassung in Pseudocode:

```
Interrupt:
   Get_ AD
   Save in Ringpuf

Main:
  Open AD
  Open Ringpuf
  While (Ringpuf  nicht ausgelesen)
```

Read Wert
Anzeigen in Graphik
Endwhile

==========

procedure **p_ad_Init**

Aufgabe: Initialisierung der Unit
Muss vor dem Open erfolgen

Parameter –

==========

procedure **Open_Random_AD**(UsedChannels : word;
SamplFreq_kz : boolean;
TimeUnit : byte;
maxSamplingRate: word;
PosVRefType : byte;
PosVrefValue : real;
NegVRefType : byte;
NegVrefValue : real);

Aufgabe: Den AD-Wandler für alle benötigten Kanäle eröffnen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	UsedChannels	word	Input	Für jeden verwendeten Kanal das Bit in der WORD-Variable setzen. z.B. 0000000000001100 Kanal2,3 verwendet
	SamplFreq_kz	boolean	Input	Kennzeichen, ob über Delay eine annähernde Samplingzeit eingehalten werden soll
	TimeUnit	byte	Input	Zeiteinheit für Angabe einer Samplingfrequenz c_sec, c_ms
	maxSamplingRate	word	Input	Samplinfrequenz z.B. 10000 pro c_sec intern berechneter Delay ca. 100µs
	PosVRefType	byte	Input	Typ der pos.Referenzspannung Intern: c_Vdd (+3,3V) oder Extern: c_extVdd
	PosVrefValue	real	Input	Wert für c_extVdd
	NegVRefType	byte	Input	Typ der neg.Referenzspannung Intern: c_Vss (GND) oder

				Extern: c_extVss
	NegVrefValue	real	Input	Wert für c_extVss

==========

procedure **Get_Random_AD_Value**(Channel: byte;
var ADword : word;
var ADvalue: real

Aufgabe: Analogwert eines Kanals einlesen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Channel	Byte	Input	Kanal: 0..15
var	Adword	Word	Output	10bit-wert des Analogsignals
var	ADValue	Real	Output	Real-Wert des Analogsignals

==========

procedure **Check_Wait_for_Random**;

Aufgabe: Einhalten der Samplingfrequenz
nicht berücksichtigt ist hier die Verarbeitungszeit
zwischen 2 Sampling-Vorgängen

wenn Open_Random_AD (…SamplFreq_kz =true,…
dann Delay über Check_Wait_for_Random

Parameter -

==========

procedure **Open_Sample_AD** (UsedChannels : word;
PosVRefType : byte;
PosVrefValue : real;
NegVRefType : byte;
NegVrefValue : real)

Aufgabe: Interrupt-gesteuerte Analogerfassung eröffnen
Die einzelnen Kanäle müssen noch separat eröffnet werden,
da unterschiedliche Samplingzeiten möglich sind
(siehe Open_Sample_AD_Channel)

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	UsedChannels	word	Input	Für jeden verwendeten Kanal das Bit in der WORD-Variable setzen. z.B. 0000000000001100 Kanal2,3 verwendet

	PosVRefType	byte	Input	Typ der pos.Referenzspannung Intern: c_Vdd (+3,3V) oder Extern: c_extVdd
	PosVrefValue	real	Input	Wert für c_extVdd
	NegVRefType	byte	Input	Typ der neg.Referenzspannung Intern: c_Vss (GND) oder Extern: c_extVss
	NegVrefValue	real	Input	Wert für c_extVss

==========

procedure **Open_Sample_AD_Channel**(Channel : byte;
TimeUnit : byte;
SampleRate : word);

Aufgabe: Eröffnen eines einzelnen Interrupt-gesteuerten Analog-Kanals.
Die einzelnen Kanäle müssen noch separat eröffnet werden,
da unterschiedliche Samplingzeiten möglich sind
(siehe Open_Sample_AD_Channel)

Jeder Kanal kann mit einer eigenen Samplingzeit definiert werden. Da aber alle Kanäle von nur einem Timer (TIMER1) bedient werden muss eine gemeinsame kleinste Samplingzeit berechnet werden. Dies Erfolgt über die Ermittlung eines kleinsten gemeinsamen Teilers GGT für alle definierten Samplingzeiten. Auf Basis dieser GGT wird für jeden Kanal ein Multiplikations-Faktor (FACTOR) für seine eingene Abtastzeit berechnet

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Channel	Byte	Input	Kanal-Nr (0..15)
	Timeunit	Byte	Input	Zeiteinheit für Sampligangabe c_sec, c_ms
	Samplerate	Word	Input	Abtastungen pro Timeunit z.B. 2 pro c_sec

==========

procedure **ADwordToReal**(ADword: word; var ADValue: real);

Aufgabe: den vom AD-Wandler gelieferten 10bit-Wert in einen Real-Wert als Spannungsangabe konvertieren.
Der Spannungswert wird auf Basis der Referenzspannungen berechnet

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Adword	Word	Input	10bit-Wert von AD-Wandler
	Advalue	Real	Input	Gemessene Spannung

==========

procedure **Start_Sampling**

Aufgabe: Start der Interruptbetriebenen AD-Wandlung

Parameter: -

==========

procedure **WordIndToSampleTime**(TimeInd: word; var SampleTime: real;
var SampleUnit: byte);

Aufgabe: die allen Samplingkanälen gemeinsame min.Samplingzeit wird zur Verfügung gestellt

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	TimeInd	word	Input	Index der n.ten Abtastung (AD-Wandlung)
var	SampleTime	real	Output	Berechneter Zeitpunkt der n-ten Abtastung
var	Sampleunit	byte	Output	Zeiteinheit

==========

procedure **ChangeSampleTime**(SampleTime: real; SampleUnit: byte;
var NewSampleTime: real;
NewSampleUnit: byte)

Aufgabe: Samplingzeit von einer Zeiteinheit in eine andere Zeiteinheit umrechnen
z.B. 5 sec = 5000 ms

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	SampleTime	real	Input	Urspr.Samplingzeit
	SampleUnit	Byte	Input	Urspr.Zeiteinheit
Var	NewSampleTime	Real	Output	Berechnete Samplingzeit
	NewSampleTime	Byte	Input	Neue Zeiteinheit

==========

procedure **Open_RingPuf**(Handle: byte; NumValues: word)

Aufgabe: Eröffnen Ringpuffer für eine Anzahl von WORD-Werten
Der Puffer wird auf dem Heap angelegt.
2 Ringpuffer können eröffnet werden.

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Handle	Byte	Input	Nr des Ringpuffers (derzeit 0 und 1)
	NumValues	Word	Input	Anzahl der möglichen Werte in Puffer

==========

procedure **Reset_RingPuf**(Handle: byte)

Aufgabe: Reset des Ringpuf-Read-Pointer auf Anfang. Ein erneutes Auslesen ist mögöich.

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Handle	Byte	Input	Nr des Ringpuffers (derzeit 0 und 1)

==========

procedure **Read_RingWord**(Handle: byte; var WordVal: word)

Aufgabe: Nächsten Wert aus dem Puffer lesen
Über SyRc kann geprüft werden, ob das Lesen erfolgreich war.

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Handle	Byte	Input	Nr des Ringpuffers (derzeit 0 und 1)
Var	WordVal	Word	Output	Wert aus Puffer

==========

procedure **Write_RingWord**(Handle: byte; WordVal: word)

Aufgabe: Nächsten Wert in Puffer schreiben
Über SyRc kann geprüft werden, ob das Schreiben erfolgreich war.

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Handle	Byte	Input	Nr des Ringpuffers (derzeit 0 und 1)
	WordVal	Word	Output	Zu schreibender Wert

==========

procedure **Close_RingPuf**(Handle: byte)

Aufgabe: Ringpuffer löschen; Heap freigeben

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Handle	Byte	Input	Nr des Ringpuffers (derzeit 0 und 1)

procedure Init_Timer1 (TimeIntervall: real); (* µs *)
procedure ReInit_Timer1;
procedure Start_Timer1 ;
procedure Stop_Timer1 ;

Unit **p_digC86**

Unit zur numerischen Wertaausgabe in Form einer 7-Sement-Anzeige.
Die Größe und Strichstärke kann definiert werden.
Die Darstellungsform eines Wertes in Vor-und Nachkommastellen kann definiert werden..
5 Anzeigen können gleichzeitig eröffnet werden

==========

procedure **p_digC86_Init**

Aufgabe: Initialisierung der Unit
Muss vor dem Open erfolgen

Parameter –

==========

procedure **OpenDig** (x,y : integer;
x_rlb : byte;
Groesse : byte;
Dicke : byte;
farbe : word;
invFarbe : word;
var PicForm : string[22];
var DigNr: byte);

Aufgabe: 7-Sement-Anzeige eröffnen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	X	Integer	Input	X-Pixel-Koordinate für Beginn der Anzeige
	Y	Integer	Input	Y-Pixel-Koordinate für Beginn der Anzeige
	groesse	byte	Input	Zeichengrösse(1..n; sinnvoll 1..3)
	dicke	byte	Input	Strichdicke (1..n; sinnvoll 1..3)
	farbe	word	Input	Strichfarbe (nur weiss);
	invFarbe	word	Input	Hintergrundfarbe (nur Black)
	picform	String[22]	Input	Format der Ausgabe Z: führendes Blank wenn 0 P: Ziffer ausgeben (auch führende 0) V: definiert Dezimaltrenner PicForm-Beispiele Z.ZZZ.ZZP ZZPV,PPP ZZP,P ZZ.ZZPV,PP000 ZZ,ZZPV.P Dezimaltrenner Punkt oder Komma

var	DigNr	byte	Output	zurückgegebenes handle

==========

procedure **CloseDig** (DigNr: byte)

Aufgabe: Anzeige löschen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	DigNr	Byte	Input	Handle-Nr der Anzeige

==========

procedure **WriteDig** (DigNr: byte;
r: real;
var ErrorKz: boolean)

Aufgabe: Anzeigewert ausgeben

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	DigNr	Byte	Input	Handle-Nr der Anzeige
	R	Real	Input	Anzeigewert
Var	ErrorKz	Bnoolean	Output	GenerellerFehlerhinweis Genaue Fehlerursache über SyRc ermitteln

Unit **p_mdxC86**

Die Unit p_mdcC86 bietet Bargraph und Scrollbar-Funktionen in X-Richtung.
Gleichzeitig können 8 Bargraph und 2 Scrollbar-Graphen dargestellt werden.

==========

procedure **Open_X_BarGraph** (bargr_nr : byte;
xvon,xbis,yvon,ybis: integer;
xwert,ywert : integer;
wert_min,wert_max : real;
font : byte;
fontstyle : byte;
h_farbe, l_farbe : byte)

Aufgabe: Bargraph öffnen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	bargr_nr	Byte	Input	Bargraph-Nr (Handle-Nr) der Anzeige (1-8)
	Xvon	Integer	Input	Pixel-Koordinate
	Yvon	Integer	Input	Pixel-Koordinate
	Xbis	Integer	Input	Pixel-Koordinate
	Ybis	Integer	Input	Pixel-Koordinate
	Xwert	Integer	Input	X-Pixel-Koordinate für Wertanzeige
	Ywert	Integer	Input	Y-Pixel-Koordinate für Wertanzeige
	Wert_min	Real	Input	Min.Anzeigewert
	Wert_max	Real	Input	Max.Anzeigewert
	Font	Byte	Input	Anzeige-Font (Default, Num)
	FontStyle	Byte	Input	Anzeige-Style(invertiert,…)
	H_farbe	Byte	Input	Hintergrund-Farbe (black)
	L_farbe	Byte	Input	Zeichenfarbe(white)

==========

procedure **X_BarGraph** (bargr_nr : integer;
wert_neu : real)

Aufgabe: Bargraph zu aktuellem Wert anzeigen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	bargr_nr	Byte	Input	Bargraph-Nr (Handle-Nr) der Anzeige (1-8)
	Wert_neu	Real	Real	Anzeigewert(Wert_min <= Wert <= Wert_max)

==========

procedure **Close_X_BarGraph** (bargr_nr: byte);

Aufgabe: Beenden des X-Bargraph

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	bargr_nr	Byte	Input	Bargraph-Nr (Handle-Nr) der Anzeige (1-8)

==========

procedure **Open_X_ScrollBarGraph** (bargr_nr : byte;
xvon,xbis,yvon,ybis: integer;
len : byte;
wert_min,wert_max : real;
font : byte;
fontstyle : byte;
h_farbe : byte;
l_farbe : byte)

Aufgabe: ScrollBarGraph öffnen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	bargr_nr	Byte	Input	ScrollBargraph-Nr (Handle-Nr) der Anzeige (1-2)
	Xvon	Integer	Input	Pixel-Koordinate
	Xbis	Integer	Input	Pixel-Koordinate
	Yvon	Integer	Input	Pixel-Koordinate
	Ybis	Integer	Input	Pixel-Koordinate
	Len	byte	Input	Scrollbalken-Länge (Pixel)
	Wert_min	Real	Input	Min.Anzeigewert
	Wert_max	Real	Input	Max.Anzeigewert
	Font	Byte	Input	Anzeige-Font (Default, Num)
	FontStyle	Byte	Input	Anzeige-Style(invertiert,…)
	H_farbe	Byte	Input	Hintergrund-Farbe (black)
	L_farbe	Byte	Input	Zeichenfarbe(white)

==========

procedure **X_ScrollBarGraph** (bargr_nr : byte;
wert_neu : real)

Aufgabe: ScrollBargraph zu aktuellem Wert anzeigen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	bargr_nr	Byte	Input	ScrollBargraph-Nr (Handle-Nr) der Anzeige (1-2)
	Wert_neu	Real	Real	Anzeigewert(Wert_min <= Wert <= Wert_max)

==========

procedure **Close_X_ScrollBarGraph** (bargr_nr: byte)

Aufgabe: Beenden des X-ScrollBargraph

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	bargr_nr	Byte	Input	ScrollBargraph-Nr (Handle-Nr) der Anzeige (1-2)

p_mdyC86

Die Unit p_mdcC86 bietet Bargraph und Scrollbar-Funktionen in Y-Richtung.
Gleichzeitig können 8 Bargraph und 2 Scrollbar-Graphen dargestellt werden.

==========

procedure **Open_Y_BarGraph** (bargr_nr : byte;
xvon,xbis,yvon,ybis: integer;
xwert,ywert : integer;
wert_min,wert_max : real;
font : byte;
fontstyle : byte;
h_farbe, l_farbe : byte)

Aufgabe: Bargraph öffnen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	bargr_nr	Byte	Input	Bargraph-Nr (Handle-Nr) der Anzeige (1-8)
	Xvon	Integer	Input	Pixel-Koordinate
	Yvon	Integer	Input	Pixel-Koordinate
	Xbis	Integer	Input	Pixel-Koordinate
	Ybis	Integer	Input	Pixel-Koordinate
	Xwert	Integer	Input	X-Pixel-Koordinate für Wertanzeige
	Ywert	Integer	Input	Y-Pixel-Koordinate für Wertanzeige
	Wert_min	Real	Input	Min.Anzeigewert
	Wert_max	Real	Input	Max.Anzeigewert
	Font	Byte	Input	Anzeige-Font (Default, Num)
	FontStyle	Byte	Input	Anzeige-Style(invertiert,…)
	Direction	Byte	Input	Richtung des Anzeigewertes
	H_farbe	Byte	Input	Hintergrund-Farbe (black)
	L_farbe	Byte	Input	Zeichenfarbe(white)

==========

procedure **Y_BarGraph** (bargr_nr : integer;
wert_neu : real)

Aufgabe: Bargraph zu aktuellem Wert anzeigen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	bargr_nr	Byte	Input	Bargraph-Nr (Handle-Nr) der Anzeige (1-8)
	Wert_neu	Real	Real	Anzeigewert(Wert_min <= Wert <= Wert_max)

==========

procedure **Close_Y_BarGraph** (bargr_nr: byte)

Aufgabe: Anzeige löschen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	bargr_nr	Byte	Input	Bargraph-Nr (Handle-Nr) der Anzeige (1-8)

==========

procedure **Open_Y_ScrollBarGraph** (bargr_nr : byte;
xvon,xbis,yvon,ybis: integer;
len : byte; (* ScrollBarGraph-Länge *)
wert_min,wert_max : real;
font : byte;
fontstyle : byte;
h_farbe : byte;
l_farbe : byte)

Aufgabe: ScrollBarGraph öffnen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	bargr_nr	Byte	Input	ScrollBargraph-Nr (Handle-Nr) der Anzeige (1-2)
	Xvon	Integer	Input	Pixel-Koordinate
	Xbis	Integer	Input	Pixel-Koordinate
	Yvon	Integer	Input	Pixel-Koordinate
	Ybis	Integer	Input	Pixel-Koordinate
	Len	byte	Input	Scrollbalken-Länge (Pixel)
	Wert_min	Real	Input	Min.Anzeigewert
	Wert_max	Real	Input	Max.Anzeigewert
	Font	Byte	Input	Anzeige-Font (Default, Num)
	FontStyle	Byte	Input	Anzeige-Style(invertiert,…)
	Direction	Byte	Input	Richtung des Anzeigewertes
	H_farbe	Byte	Input	Hintergrund-Farbe (black)
	L_farbe	Byte	Input	Zeichenfarbe(white)

==========

procedure **Y_ScrollBarGraph** (bargr_nr : byte;
wert_neu : real)

Aufgabe: ScrollBargraph zu aktuellem Wert anzeigen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	bargr_nr	Byte	Input	ScrollBargraph-Nr (Handle-Nr) der Anzeige (1-2)
	Wert_neu	Real	Real	Anzeigewert(Wert_min <= Wert <= Wert_max)

==========

procedure **Close_Y_ScrollBarGraph** (bargr_nr: byte)

Aufgabe: Anzeige löschen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	DigNr	Byte	Input	Handle-Nr der Anzeige

Unit **p_drsC86**

Die Unit p_drsC86 bietet Funktionen für Analogzeiger.
Diese Funktionalität kann nur für mikroMedia for PIC24 verwendet werden

---

procedure **TCircle1** (xm : integer; (* x-Mittelpunkt *)
ym : integer; (* y-Mittelpunkt *)
r : real; (* Radius *)
ScaleColor: word);

Aufgabe: Kreis zeichnen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Xm	Byte	Input	x-Mittelpunkt
	Ym	Integer	Input	y-Mittelpunkt
	R	real	Input	Radius
	ScaleColor	word	Input	Kreisfarbe

.

---

procedure **TCircle2**
(xm : integer;
ym : integer
r : real;
alph1max : real;
alph1min : real;
Color1 : word;
alph2max : real;
alph2min : real;
Color2 : word;
alph3max : real;
alph3min : real;
Color3 : word;
BackColor : word;
ScaleColor : word);

Aufgabe: obere Kreishälfte zeichnen
mit max 3 farbig markierten Kreisausschnitten

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Xm	byte	Input	x-Mittelpunkt

	Ym	Integer	Input	y-Mittelpunkt
	r	real	Input	Radius
	alph1max	real	Input	Startwinkel 1.Segment
	alph1min	real	Input	Stopwinkel 1.Segment
	color1	word	Input	Farbe des Segmentes
	alph2max	real	Input	Startwinkel 2.Segment
	alph2min	real	Input	Stopwinkel 2.Segment
	Color2	word	Input	Farbe des Segmentes
	alph3max	real	Input	Startwinkel 3.Segment
	alph3mon	real	Input	Stopwinkel 3.Segment
	color3	word	Input	Farbe des Segmentes
	BackColor	word	Input	Standardfarbe
	ScaleColor	word	Input	Skalierfarbe

---

procedure **Open_Hinstr**

```
(w            :  byte;
 sc           :  byte;
 xv,yv        :  integer;
 xb,yb        :  integer;
 BackColor    :  word;
 ScaleColor   :  word;
 ZeigerColor:  word;
 Umin,Umax    : real;

 Smin1,Smax1: real;
 Color1       :  word;
 Smin2,Smax2: real;
 Color2       : word;
 Smin3,Smax3: real;
 Color3       : word;
 dUSkal       : real);
```

Aufgabe: Horizontales Instrument zeichnen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	W	byte	Input	Nr des Instruments (max.4)
	Sc	byte	Input	Instrumenttyp c_simple / c_complex c_simplex: nur einfaches Instrument ihne farbige Abschnitte c_complex: farbig markierte Kreissegmente
	Xv	integer	Input	xv des Windows
	Yv	integer	Input	yv des Windows
	Xb	integer	Input	xb des Windows
	Yb	integer	Input	yb des Windows

	BackColor	word	Input	Standardfarbe
	ScaleColor	word	Input	Skalierfarbe
	ZeigerColor	word	Input	Farbe des Zeigers
	Umin	real	Input	Min.Anzeigewert
	Umax	real	Input	Max.Anzeigewert
	Smin1	real	Input	Startwert 1.Segment
	Smax1	real	Input	Stopwert 1.Segment
	Color1	word	Input	Farbe 1 Segment
	Smin2	real	Input	Startwert 2.Segment
	Smax2	real	Input	Stopwert 2.Segment
	Color2	word	Input	Farbe 2 Segment
	Smin3	real	Input	Startwert 3.Segment
	Smax3	real	Input	Stopwert 3.Segment
	Color3	word	Input	Farbe 3 Segment
	dUSkal	real	Input	Skalierwert-Delta

Beispiel: Volt-Instrument 0..10V mit den Segmenten 2-5V / 6-8V / 9-10V

```
p_Init_drsC86();
Open_Hinstr
        (1,             // Instr.-Nr
        c_komplex,
        1,1,140,66,
        yellow,         // background
        red,            // Scaling
        black,          // Pointer
        0,10,           // Umin,Umax
        2,5,            // Smin,Smax
        green,
        6,8,            // Smin,Smax
        brown,
        9,10,           // Smin,Smax
        blue,
        1);             // Scaling-Delta
```

---

procedure **Wert_HInstr**(w : byte;
wert: real);

Aufgabe: Wert auf Instrument anzeigen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	W	byte	Input	Nr des Instruments (max.4)
	wert	Real	Input	Anzeigewert

---

procedure **Close_HInstr**(w : byte);

Aufgabe: Instrument löschen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	W	byte	Input	Nr des Instruments (max.4)

---

procedure **p_Init_drsC86**

Aufgabe: Heapspeicher zur Verfügung stellen
Variablen initialisieren

---

procedure **p_Close_drsC86**

Aufgabe: Heapspeicher löschen

Unit **p_eFlash**

Die Unit p_eFlash bietet ein komplettes Filesystem für den Zugriff auf Recordebene.
Diese Funktionalität wurde für „mikroMedia for PIC24“ entwickelt.

==========

general functions

procedure **p_minStart_eFlashFileSystem**;

Aufgabe: Hardware-Initialisierung des Flash-Zugriffs

Parameter -

==========

procedure **p_Start_eFlashFileSystem**(maxFileNum: byte);

Aufgabe: Hardware-Initialisierung des Flash-Zugriffs
Initialosieren Puffer, Directory usw.

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	maxFileNum	byte	Input	Anzahl der max. verwendeten Files (-> Puffergröße)

==========

procedure **p_Stop_eFlashFileSystem**;

Aufgabe: Löschen aller Puffer

Parameter -

==========

function **eFlashActive**: boolean;

Aufgabe: Prüfen, ob Filesystem gestartet ist

Parameter -

==========

procedure **Clear_eFlash**; //Dangerous !!!

Aufgabe: kompletten Flashspeicher löschen (alle Bytes auf $FF setzen)

Parameter -

File functions

```
procedure eOpenFile(var Name: array[0..9] of char;
                    var Handle: byte);

procedure eNewFile(var Name : array[0..9] of char;
                   Size     : word;
                   RecLen   : byte;
                   var Handle: byte);

procedure eDeleteFile(handle: byte);

procedure eCloseFile(handle: byte);

procedure eCloseAllFiles;
```

Record functions

Ein Record (z.B. t_fb) muss folgenden Aufbau haben:

```
Type
  t_fb = record
       //internal data  (don't update)
       dummy     : byte;     //0          //Word-Allocation !!!
       DletFlag  : byte;     //1
       RecNr     : word;     //2
       RecNrOld  : word;     //4

       //User data
       //ab hier beliebige Daten
    end          ...
```

```
procedure eWriteRecord(handle: byte; RecAddr: ^byte);

procedure eUpdateRecord(handle: byte; RecAddr: ^byte);

procedure eReadRecord(handle: byte; RecAddr: ^byte);
```

procedure **eReadNextRecord**(handle: byte; RecAddr: ^byte);

procedure **eDeleteRecord**(handle: byte);

procedure **eReadRecordWithDlet**(handle: byte; RecAddr: ^byte);

procedure **eReadNextRecordWithDlet**(handle: byte; RecAddr: ^byte);

Variable functions

procedure **Read_eFlashByte**(iAddr: longint; var iByte: byte);

procedure **Write_eFlashByte**(iAddr: longint; iEnableFlag: byte; iByte: byte);

procedure **Read_eFlashWord**(iAddr: longint; var iWord: word);

procedure **Write_eFlashWord**(iAddr: longint; iWord: word);

procedure **Read_eFlashText**(iAddr: longint; var iStr: string[10]);

procedure **Write_eFlashText**(iAddr: longint; var iStr: string[10]);

internal functions

Die Beschreibung der folgenden Funktionen ist ist aus dem Datenblatt für M25P80 zu entnehmen.

function **eFlash_IsWriteBusy**(): boolean;

procedure **Read_eFlashStatus**(var Stat1: byte);

procedure **Read_eFlash_Identification**;

procedure **Init_eFlashSPI**;

procedure **Init_eFlash;**

procedure **Write_eFlashEnable**;

procedure **Set_eFlash_CS**;

procedure **Reset_eFlash_CS**;

Debug / Admin functions

Für die Erstellung eines Flash-Directories auf dem TFT-Screen wird die folgenden 4 Funktionen in der darfestellten Reihenfolge aufgerufen (siehe p_afu_tch)

procedure **Open_eDirLine**;
procedure **eFlashDir**(AnzFiles: byte);
loop
  procedure **Read_eDirLine**(Ind: byte; var Line: string[50]);
endloop
procedure **Close_eDirLine**;

==========

procedure **eDumpDir;**

Aufgabe: Zusammenfassung der 3 folgenden Funktionen
RDump, Edump, HandleDump

==========

procedure **RDump**;

Aufgabe: Dump des erweiterten Flash-Directories im Ram per RS232

==========

procedure **EDump**;

Aufgabe: Dump des Flash-Directories direkt aus Flash per RS232

==========

procedure **HandleDump**;

Aufgabe: Dump der Handle-Daten per RS232

==========

procedure **eFlash_HexDump**(Mode: byte; StartAdr: longint; len: byte);

Aufgabe: Hex-Dump ab einer vorgegeben Adresse im Ram-/Flash-Memory

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Mode	byte	Input	1: Zugriff auf Sram 3: Zugriff auf Flash-Memory
	StartAdr	longint	Input	Start-Adresse
	Len	Byte	Input	Anzahl anzuzeigende Bytes (max.$FF)

==========

procedure **Insert_StrucField**(var FileName : array[0..9] of char;
var Fieldname: string[10];
pos : byte;
KeyType : boolean;
SType : byte;
ArrCnt : byte;
HexDez : char);

Aufgabe: Definition eines Record-Feldes für den FileDump

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	FileName	Char[0..9]	Input	Filename
	Fieldname	String[10]	Input	Feldname
	Pos	byte	Input	Offset innerhalb Record
	KeyType	boolean	Input	Key-Feld? Nicht benutzt
	SType	byte	Input	Feld-Typ c_Type_char = 1 c_Type_byte = 2 c_Type_word = 3 c_Type_integer = 4 c_Type_longint = 5 c_Type_dword = 6 c_Type_real = 7
	ArrCnt	byte	Input	Array-Counter
	HexDez	char	Input	H: Hex- D: Dezinal-Ausgabe

==========

procedure **Clear_StrucTab**;

Aufgabe: Aufgabe: Speicher auf dem Heap freigeben

==========

procedure **eFileDump**(Handle: byte);

Aufgabe: Hex-Dump des zum Handle gehörigen Files
Aufbereitung der Ausgabe eines Records entsprechend den mit Insert_StrucField vorgegebenen Daten

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Handle	byte	Input	Handle-Nr (max.4) des Files

==========

procedure **eCountFiles**(var FileCnt_active: byte;
var FileCnt_all : byte);

Aufgabe: Anzahl aller Files im Flash ermitteln

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
var	FileCnt_active	byte	Input	Anzahl der noch nicht gelöschten Files
var	FileCnt_all	longint	Input	Anzahl aller Files(gelöscht und nicht gelöscht)

Unit **p_tchC86**

Die Unit p_tchC86 bietet Funktionen für das Anzeigen und Auswerten von Touch-Objekten. Diese Funktionalität wurde für „mikroMedia for PIC24“ entwickelt

Beispiele: **siehe p_afu_tch**

==========

procedure **Open_Color_Buttons**;

Aufgabe: Farbauswahl eröffnen und anzeigen

Parameter -

==========

procedure **Get_Color_Buttons**(var oVal : word;
var oEndMode: byte);

Aufgabe: Farbe auswählen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
Var	oVal	word	Output	Nr der gewählten Farbe
Var	oEndMode	byte	Output	Returncode aus Farbauswehl

==========

procedure **Open_NChar_Buttons**;

Aufgabe: Anzeige Numerik-Auswahl 0..9

Parameter -

procedure **Get_NChar_Buttons**(var oStr: string[20];
var oVal: real;
var oEndMode: byte);

Aufgabe: Zahl eintippen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
Var	OStr	String[20]	Output	Zahl als String
Var	oVal	Real	Output	Rahl als Wert
Var	oEndMode	Byte	Output	Returncode der Zahleneingabe

==========

procedure **Open_TChar_Buttons**(var iStr: string[20]);

Aufgabe: Zeichenauswahl eröffnen

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	iStr	String[20]	Input	Vorbelegung

==========

procedure **Open_NumCharInput**(var oText: string[20];
var EndMode: byte);

Aufgabe: Stringerfassung mit Zahl

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
var	oText	String[20]	Input/ Output	Vorbelegung / Erfasster String
Var	oEndMode	Byte	Output	Returncode der Stringeingabe

==========

procedure **Get_TChar_Buttons**(var oStr: string[20];
var oEndMode: byte);

Aufgabe: Zeichenstring erfassen (ohne Zahl)

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar

Var	OStr	String[20]	Output	Zahl als String
Var	oEndMode	Byte	Output	Returncode der Stringeingabe

==========

procedure **OpenTouch**;

Aufgabe: Touch-Funktionalität aktivieren
Falls noch keine Kalibrierung erfolgte, kalibrieren

==========

procedure **CloseTouch**;

Aufgabe: Touch-Funktionalität beenden

==========

function **Touch: boolean**;

Aufgabe: Prüfen, ob Touch-Druck erfolgte

==========

procedure **Open_GraphTouch**;

Aufgabe: Speicher für Graphische Touch-Objekte anfordern

==========

procedure **Set_TouchField**(Fnr: byte; xv,yv,xb,yb: integer; FuncNr: byte);

Aufgabe: Touch-Feld definieren
(nur Koordinaten, noch kein Graph.Element)

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Fnr	Byte	Input	Feldnr
	xv	Integer	Input	XV-Koordinate Touch-Feld
	yv	Integer	Input	YV-Koordinate Touch-Feld
	xb	Integer	Input	XB-Koordinate Touch-Feld
	yb	Integer	Input	YB-Koordinate Touch-Feld
	FuncNr	Byte	Input	Nummer der zugehörigen Funktion

procedure **Set_TouchFieldMode**(FldNr: byte; Active_InActive: byte);

Aufgabe: Touch-Feld aktivieren / deaktivieren

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	FldNr	Byte	Input	Nr des Touch-Feldes
	Active_InActive	Byte	Input	C_active C_inActive

==========

procedure **Set_GraphTouchField**(Fnr: byte;
wxv,wyv,wxb,wyb: integer;
t1xv,t1yv: integer;
var Text1: string[20];
t2xv,t2yv: integer;
var Text2: string[20];
WinColor : word;
TextColor: word;
RectColor: word);

==========

procedure **Set_GraphIOTouchField**(Fnr: byte;
wxv,wyv,wxb,wyb: integer;
t1xv,t1yv: integer;
var Text1: string[30];
t2xv,t2yv: integer;
var Text2: string[30];
IOxv : integer; //rel. zu wxv
IOyv : integer; //rel. zu wyv
var IOBer: string[20];
WinColor : word;
TextColor: word;
IOColor : word;
RectColor: word);

Aufgabe: Touch-Menu-Feld definieren

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Fnr	Byte	Input	Nr des Touch-Menu-Feldes
	wxv	integer	Input	xv-Koordinate des Touch-Feldes
	wyv	integer	Input	yv-Koordinate des Touch-Feldes
	wxb	integer	Input	xb-Koordinate des Touch-Feldes
	wyb	integer	Input	yb-Koordinate des Touch-Feldes

	t1xv	integer	Input	Zeile1-Text-Start rel. zu wxv
	t1yv	integer	Input	Zeile1-Text-Start rel. zu wyv
	text1	String[30]	Input	Button-Text1
	t2xv	integer	Input	Zeile2-Text-Start rel. zu wxv
	t2yv	integer	Input	Zeile2-Text-Start rel. zu wyv
	Text2	String[30]	Input	Button-Text2
	IOxv	Integer	Input	IO-Feld-Start rel. zu wxv
	IOyv	Integer	Input	IO-Feld-Start rel. zu wyv
var	IOBer	String[20]	Output	IO-Text
	Wincolor	word	Input	LCD-Hintergrund-Farbe
	TextColor	word	Input	Textfarbe
	IOColor	word	Input	IO-Text-Fill-Farbe
	RectColor	word	Input	Randfarbe

==========

procedure **Write_IOField**(Fnr: byte; var IOber: string[20]);

Aufgabe: Touch-Menu-IO-ausgeben

Parameter

Worth/Ref.	Name	Typ	I/O	Kommentar
	Fnr	Byte	Input	Nr des Touch-Menu-Feldes
	IOber	String[20]	Input	Text für Menu-Button

==========

procedure **Write_IOField_Err**(Fnr: byte; var IOber: string[20]);

Aufgabe: Fehlertext zu Feld Fnr ausgeben

==========

procedure **Field_Blink**(Fnr: byte);

Aufgabe: Feld kurz aufblonken lassen
Nötig bei Touch-Eingabe

==========

procedure **Check_Touch**(var oFunc: byte);

Aufgabe: prüfen, welcher Button berührt wurde

procedure **Temp_Tch_ADC_Init**;

Aufgabe: Open ADC für Touch

==========

procedure **GetColorText**(Color: word; var oStr: string[12]);

Aufgabe: Text zu Farb-Nr ermitteln

==========

procedure **SetCColor**(ii: byte; var oColor: word);

Aufgabe: Farbe zu Index setzen

==========

procedure **DumpFCoordTab**(var iText: string[10]);

Aufgabe: Feld-Koordinaten per RS232 ausgeben

==========

procedure **DumpNCoordTab**;

Aufgabe: Numerik-Feld-Koordinaten per RS232 ausgeben

==========

procedure **DumpTCoordTab**;

Aufgabe: Alpha-Feld-Koordinaten per RS232 ausgeben

Unit **p_cntC86**

Die Unit p_cntC86 bietet Funktionen für Frequenzzähler.
Diese Funktionalität wurde für „mikroMedia for PIC24“ entwickelt.

==========

procedure **Init_Imp_RF0**;

Aufgabe: Port RF.0 für Puls-Pausen-Messung aktivieren

Parameter -

==========

procedure **Get_PP_Random_RF0**;

Aufgabe: Port RF.0 einlesen
Puls-/Pausen-Verhältnis bestimmen

==========

procedure **Start_Counter23**;
procedure **Stop_Counter23**;

==========

procedure **Init_Counter23**;

Aufgabe: Timer2,3 verketten zu 32bit-Counter
RA.15 :ist Clock-Input

==========

procedure **Read_Counter23**(var oFreq: longint);

Aufgabe: 32bit-Wert aus Counter lesen

procedure **Init_Counter45**;

Aufgabe: Timer4,5 verketten zu 32bit-Counter
RA.14 :ist Clock-Input

==========

procedure **Start_Counter45**;
procedure **Stop_Counter45**;
procedure **Read_Counter45**(var oFreq: longint);

9) Unterschiede der Toolboxen

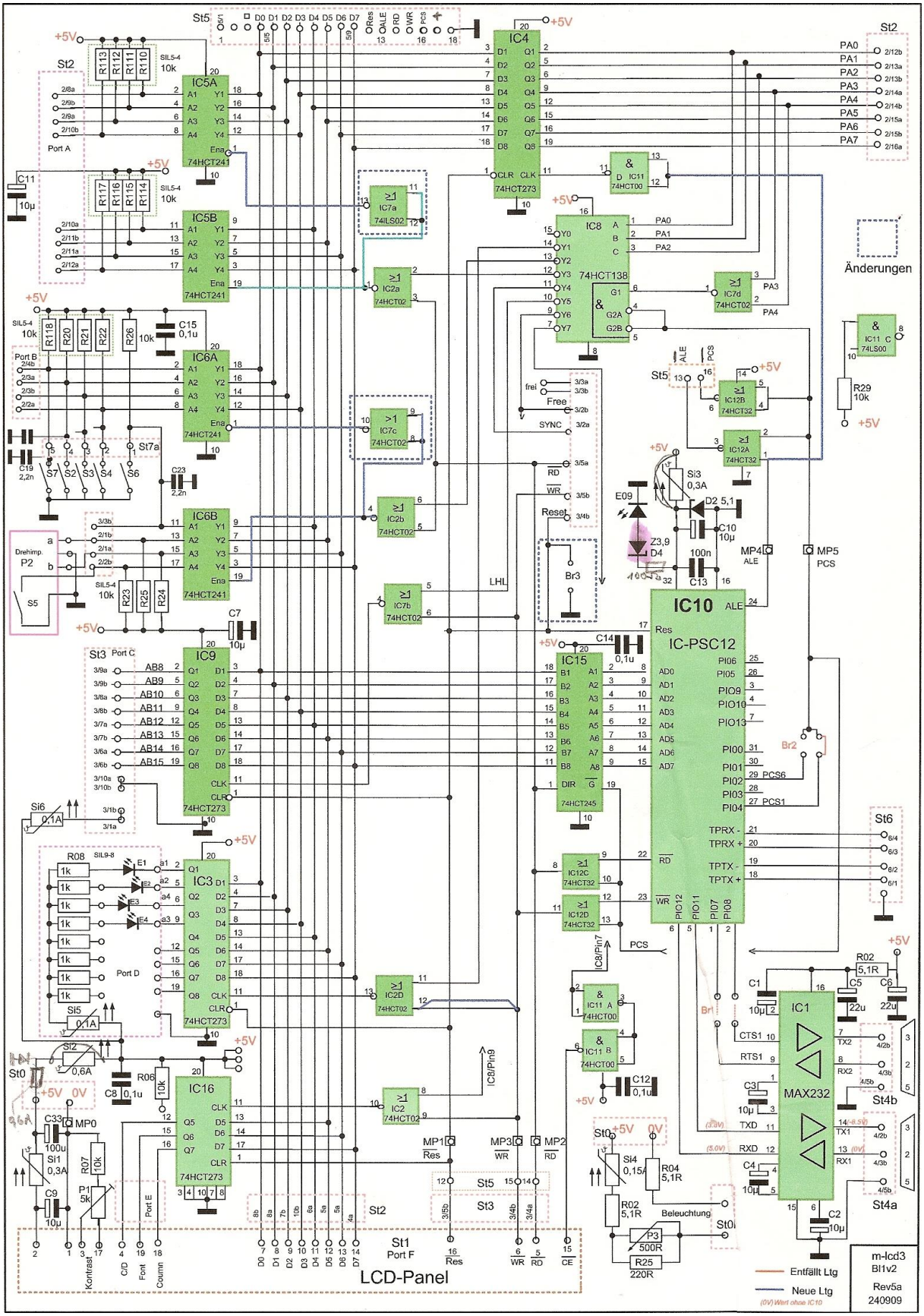
Je nach verwendetem Prozessor und Graphik-Modul ergeben sich Unterschiede in Speicherausstattung und Fähigkeiten des Graphik-Controllers. Dies hat Auswirkungen auf Ausführbare Funktionen.

Die folgende Tabelle soll Unterschiede aufzeigen:

Unit	PIC18F2620,4620 mit Graphik-LCD EA W240-7K	PIC24FJ128GB110 mit Graphik-LCD EA W240-7K	„mikroMedia for PIC24“ von Mikroelektronika
p_grpC86 p_grpMM24	kein **GetImage** kein **PutImage**	ok	kein **SetActivePage** kein **GetImage** kein **PutImage**
p_icoC86		ok	Unit nicht nötig, da Library zur Verfügung steht
p_io...	p_ioPIC18F2620	p_ioPIC24FJ128GB110	p_ioMM24
p_segC86	entfällt	ok	entfällt; Graphik-Daten müssten auf SDCard ausgelagert werden
p_drsC86	entfällt	entfällt	ok
p_cntC86	entfällt	ok	ok
p_eFlash	entfällt	entfällt	ok

Anhang:

A1) Schaltplan der Interface-Karte zwischen PIC und Lcd:


IC4
IC8
74HCT138
IC10
IC-PSC12
IC15
74HCT245
IC9
IC3
IC16
74HCT273
IC1
MAX232
IC5A
IC5B
IC6A
IC6B
74HCT241
IC7a
74LS02
IC2a
IC2b
IC2c
IC2d
IC7b
IC7c
74HCT02
IC11
74HCT00
IC12A
IC12B
IC12C
IC12D
74HCT32
LCD-Panel
Änderungen
Entfällt Ltg
Neue Ltg
m-lcd3
Bl1v2
Rev5a
240909

Einsatz von **PIC18F2620** als IC10

IC10-Pin	Pin-Bezeichnung	PIC18F-Pin	PIC-18F-Bezeichnung
8	AD0	21	PB.0
9	AD1	22	PB.1
10	AD2	23	PB.2
11	AD3	24	PB.3
12	AD 4	25	PB.4
13	AD5	26	PB.5
14	AD6	27	PB.6
15	AD7	28	PB.7
22	/RD	5	PA.3
23	/WR	6	PA.4
27	/PCS1	4	PA.2
24	ALE	2	PA.0
17	RES	7	PA.5
IC11-4	/CE	11	PC.0

Einsatz von **PIC24FJ128GB110**

IC10-Pin	Pin-Bezeichnung	PIC24FJ-Pin	PIC-24FJ-Bezeichnung
8	AD0	72	PD.0
9	AD1	76	PD.1
10	AD2	77	PD.2
11	AD3	78	PD.3
12	AD 4	81	PD.4
13	AD5	82	PD.5
14	AD6	83	PD.6
15	AD7	84	PD.7
22	/RD	69	PD.9
23	/WR	56	PD.10
27	/PCS1	47	PD.14
24	ALE	80	PD.13
17	RES	79	PD.12
IC11-4	/CE	71	PD.11

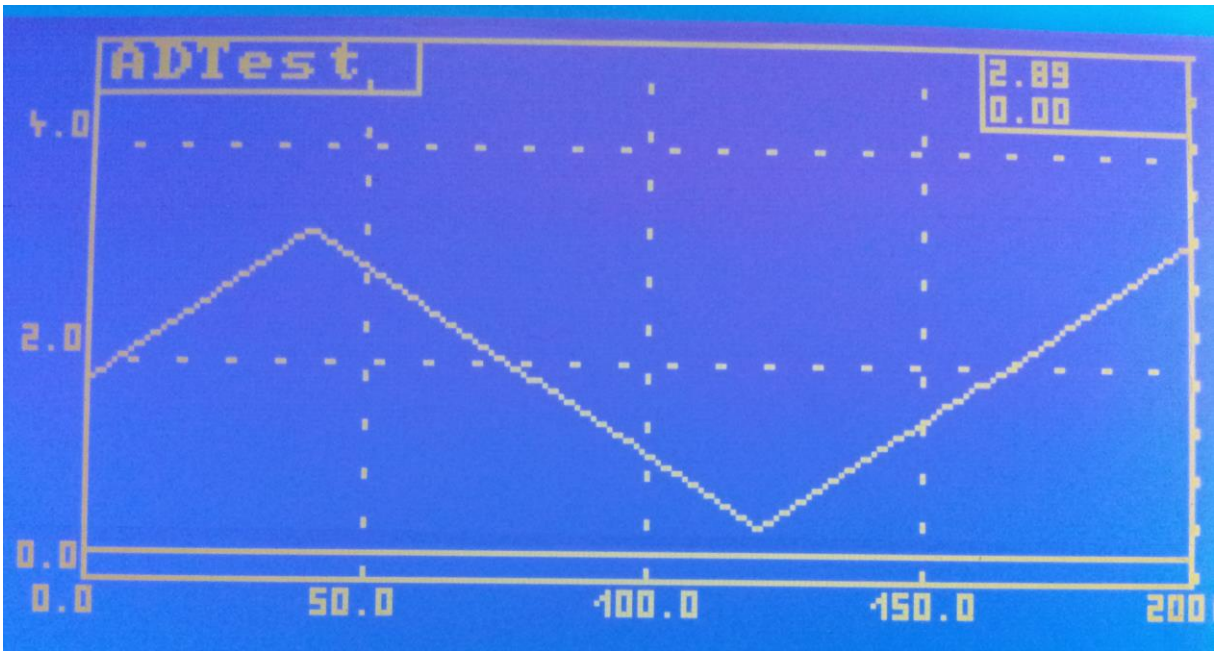
A2) Source-Beispiel für Datenerfassung und Anzeige

```
(*-----------------------------------------------------------*)
(* nicht Timergesteuertes AD-Wandeln  und Anzeige            *)
(*-----------------------------------------------------------*)
program ADTest_3;
uses
  p_defC86,
  p_debC86,
  p_md3C86,
  p_grpC86,
  p_ioPIC18F2620,
  p_adPIC18F2620;

const
  heap_start : word = 2500;
  heap_size  : word = 1000;

var
  save_STATUS      : byte;
  save_PCLATH      : byte;
  ii               : word;
  ic               : byte;
  SamplingTime_C1  : real;
  SamplingTime_C4  : real;
  RingCounter      : word;
  iADWord          : integer;
  ADvalue1         : real;
  ADvalue4         : real;

  RingInd          : word;

  SampleTime       : real;
  SampleUnit       : byte;

  iTxSample        : real;

(*----------------------------------------*)
procedure InitVars;
begin
  TestMode     := true;

  g_HeapStart  := Heap_Start;
  g_HeapSize   := Heap_Size;
  g_StatusType := c_No_F_Read;

  g_factor     := 0;     (* für Interrupt / Kanalzuordnung *)

  iLATC1       := 0;
end;
```

```
(*----------------------------------------*)
begin
  InitPorts();     (* PORT-IO definieren  *)
  InitVars();
  XReset();        (* Lcd RESET           *)

  (* Heap initialisieren *)
  MM_Init();

  RS232_Init(c_COM1,9600);
  Delay_ms(10);
  WTest('ADTest');

  (* Graphik initialisieren *)
  DefaultInitGraph();

  SetActivePage(Page_0);
  SetVisualPage(Page_0);
  SetViewPort(1,1,GetMaxX(),GetMaxY(),true);
  ClearLcd();

  (* Ringpuffer 0 initialisieren max. 210 Werte *)
  Open_RingPuf(0,210);
  IntRingUsed[0]    := true;
  IntRingUsed[1]    := false;

  (* Analog-Anzeige initialisieren *)
  p_md3C86_Init();
  open_md3_kurve       (1,
                        5,                  (* YMax  5 Volt          *)
                        0,                  (* YMin                  *)
                        200,                (* XMax  200 ms          *)
                        0,                  (* XMin                  *)
                        30,5,               (* xv,yv Pixel           *)
                        235,105,            (* xb,yb                 *)
                        Black,              (* Farbe Hintergrund     *)
                        White,              (* Farbe Kurve 1         *)
                        White,              (*       Kurve 2         *)
                        White,              (* Farbe Skalierung      *)
                        true,               (* interne Skalierung    *)
                        50,                 (* dx-Skalierung  50ms   *)
                        2,                  (* dy-Skalierung  2Volt  *)
                        1,                  (* SampleRate            *)
                        c_ms,               (* msec                  *)
                                            (* siehe Open_Random...  *)
                        c_ms,               (* X_ValueUnit           *)
                        c_V,                (* Volt                  *)
                        false,              (* Store                 *)
                        False,              (* XY-Display            *)
                        true,               (* Werteanzeige als String*)
                        NumFont,            (* num.Font 3x5          *)
                        false,              (* X1 Store              *)
                        false,              (* Y1 Store              *)
```

```
                    false,              (* X2 Store              *)
                    false,              (* Y2 Store              *)
                    'ADTest');         (* Überschrift           *)


  (* AD-Wandler initialisieren *)
  p_Init_AD();
  Open_Random_AD(2,       (* UsedChannels                    *)
                 false,   (*Check_Wait_for_Random aktivieren *)
                 c_sec,   (* Timeunit                       *)
                 1000,    (* Samples/sec Samples/Timeunit    *)
                          (* Test mit 10Hz-Sägezahn          *)
                 c_Vdd,   (* PosVRefType                    *)
                 5,       (* PosVrefValue                   *)
                 c_Vss,   (* NegVRefType                    *)
                 0);      (* NegVrefValue                   *)

  (* AD-Erfassung                      *)
  (* 200 Werte in Ringpuffer einlesen  *)
  for ii := 1 to 200 do
  begin
    Get_AD_Value(1,            (* AD-Kanal 1                *)
                 ADword,       (* Ergebnis als WORD-Wert    *)
                 ADvalue);     (* Ergebnis als REAL-Wert    *)
    Write_RingWord (0,ADWord);
    Check_Wait_for_Random();  (* Timing einhalten          *)
  end;

  (* AD-Erfassung ist abgeschlossen *)
  (* Werte aus Ringpuffer anzeigen  *)
  while true do
  begin
    (* Kanal-1-Werte einlesen *)
    Read_RingWord(0,ADWord);
    if SyRc = 0
    then begin
      ADwordToReal(ADword,ADvalue1);

      (* SampleUnit wie bei Open_Random_AD  *)
      RingInd := word(log_ReadAdr[0]);
      WordIndToSampleTime(RingInd,SampleTime,SampleUnit);

      (* Samplezeit umrechnen für SampleUnit wie für MD3 angegeben  *)
      ChangeSampleTime(SampleTime, SampleUnit, SamplingTime_C1, c_ms);

      (* Anzeige(SamplingTime,ADvalue) *)
      SamplingTime_C4 := SamplingTime_C1;
      write_md3_kurve(1,                  (* Window 1    *)
                      c_NextValue,
                      SamplingTime_C1,    (* X1-Time_C1  *)
                      ADValue1,           (* Y1          *)
                      SamplingTime_C4,    (* X2-Time_C2  *)
                      0);                 (* Y2          *)
```

```
    end;
  end;

  Close_RingPuf(0);
end.
```



Lcd-Photo zum obigen Programm

A3) Bezug der Toolbox ADAM/P-µM/T6963C/1

Versendet wird eine CD mit folgendem Inhalt:

- **Toolbox_18F (Source für PIC18F2620/4620)**
- **Toolbox_24 (Source für PIC24FJ128GB110)**
- **Toolbox_24_MM (Source für PIC24FJ256GB110)**
- **Doku**
- **Datenblätter**

Der Preis beträgt €17,50 incl.Mwst.

Versand nach Vorauszahlung auf unser Konto, das Ihnen nach Bestellung mitgeteilt wird.

Koller GmbH
Bergham 5
84533 Marktl
kollergmbh@gmx.de