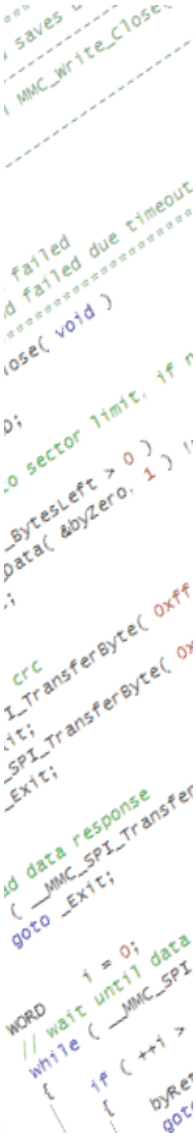
Masterarbeit

Fachhochschule Augsburg
Studienrichtung Informatik

Fachhochschule Augsburg
University of Applied Sciences

CMG – GUI-Framework

Entwicklung eines GUI-Frameworks für Embedded Systems zur plattform- und controllerunabhängigen Ansteuerung und Simulation von LCDs.



Verfasser: Christian Merkle
Abgabe: Wintersemester 2oo7/2oo8

Erstprüfer: Prof. Dr. Hubert Högl
Zweitprüfer: Prof. Dr. Christian Märtin

Erstellungserklärung

Masterarbeit gemäß §31 der Rahmenprüfungsordnung für die Fachhochschulen in Bayern (RaPO) vom 18.09.97 mit Ergänzung durch die Prüfungsordnung (PO) der Fachhochschule Augsburg vom 15.12.94.

Ich erkläre hiermit, daß ich die Arbeit selbstständig verfaßt, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt und wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum	Unterschrift

Christian Merkle
Dipl. Inf. (FH)
Füssener Str. 61 ½
86343 Königsbrunn
http://www.cmerkle.de

ÜBERSICHT

INHALTSVERZEICHNIS

1. Einleitung

1.1. Aufgabenstellung

Embedded Systems und LCD-Displays werden sowohl immer leistungsfähiger, als auch immer günstiger. Die Ansteuerung dieser Displays wird jedoch immer umfangreicher. Dazu kommt die nahezu unüberschaubare Anzahl verschiedener Controllertypen, die jeweils komplett unterschiedliche Ansteuerungen verlangen.

Für kleine Projekte ist spätestens beim Thema Plattformunabhängigkeit sowie Controllerunabhängigkeit Schluß. Weiter ist es notwendig standardisierte Schnittstellen zu verwenden, um wiederverwendbaren Code zu schreiben.

Auch Hobbyentwickler sollen mit CMG die Möglichkeit bekommen, ohne großen Aufwand, grafische Displays in ihre Projekte einzubauen und sich die damit verbundene Arbeit einzusparen.

1.2. Zielsetzung

Die Hauptziele von **CMG** – dem in dieser Arbeit entwickelten, grafischen Framework – sind:

- Einfache API und damit Ansteuerung
- Unabhängigkeit vom Display-Controller
- Unabhängigkeit von der Plattform
- Einfache Erweiterbarkeit um neue Controller und Plattformen
- Emulation und Entwicklung der Programme am PC ohne direkte Hardwareanbindung

Zusätzlich soll CMG optimiert sein für:

- Codegröße
- Ausgabegeschwindigkeit
- Modularität

1.3. Anwendungsgebiete

Die Einsatzmöglichkeiten sind nahezu unbegrenzt. Vom einfachen, portablen Gerät mit winzigem OLED-Display, welches nur wenig Text ausgibt, bis hin zum großen, farbigen Display mit Touch-Screen, welches die komplette Konfiguration komplexer Steuerungen übernimmt.

1.4. Vorhandene Lösungen

Es existieren bereits einige Lösungen zu diesem Thema. Allerdings setzen alle bisherigen Ansätze nicht die von CMG verlangten Ziele um.

Entweder sind es kleine Projekte, die nur auf einer Plattform bzw. mit nur einem Displaycontroller laufen sowie nicht erweiterbar oder modular sind.

Oder es sind sehr große Projekte, die nicht auf Codegröße und Ausführungsgeschwindigkeit optimiert sind und somit nicht auf kleinen – meist von Hobbyentwicklern verwendeten – Mikrocontrollern verwendbar sind.

1.5. Aufbau der Masterarbeit

Die Arbeit gliedert sich in insgesamt 14 Kapitel:

Nr.	Kapitel	Beschreibung
1	**Einleitung**	Diese Einleitung: Hier werden die Aufgabenstellung, die möglichen Einsatzgebiete, die Zielsetzung und die Situation der Konkurrenzprodukte näher erläutert.
2	**Grundlagen**	In diesem Kapitel wird auf die Grundlagen eingegangen.
3	**Hardware**	Vorstellung und Entwicklung der verwendeten Hardware .
4	**Softwarebasis**	Grundaufbau und Organisation des Quellcodes.
5	**LLIO – Low Level IO**	Direkte Kommunikation mit dem Display über physikalische Anschlüsse, dem Bus.
6	**LLIO – Emulatoren**	Emulation des Displays am PC.
7	**CTRL – Controller**	Ansteuerung der Displaycontroller.
8	**DRV – Driver**	Die untere Schicht des Grafiktreibers mit grundlegenden Grafikfunktionen.
9	**DRAW – Drawing**	Alle Zeichenoperationen. Diese Schicht ist die direkte Benutzerschnittstelle
10	**TEXT – Text Rendering**	Alle Funktionen zum Umgang mit Text.
11	**GUI – User Interface**	Grafisches Benutzerinterface.
12	**Zusammenfassung**	Verbesserungsvorschläge, Anmerkungen, Problembeschreibungen und ein abschließendes Fazit.

13	**Begleit-CD**	Inhalt und Aufbau der Begleit-CD.
14	**Copyright und Lizenz**	Informationen zum Copyright und den verwendeten Lizenzen zu dieser Arbeit.
	Anhang	Abbildungsverzeichnis, Tabellenverzeichnis, Listingverzeichnis, Literaturverzeichnis und der Index.

Tabelle 1 – Aufbau der Masterarbeit

1.6. SYNTAX

Das Dokument besitzt folgende Syntax:

Element	Beschreibung
Standard Blockschrift	Zusammenhängender, normaler Text
Kursiv	Hervorhebungen im Text und neu eingeführte Begriffe
`Feste Breite`	Befehle, Code, Funktionen, Variablen
`Codeblock`	Listings oder Tabellen
Fußnote[1]	Zusätzliche Erklärungen zum Begriff am Seitenende
[BEZ1]	Quellenangabe aus dem Literaturverzeichnis

Tabelle 2 – Syntax

[1] Eine Fußnote

2. Grundlagen

2.1. Was soll unterstützt werden

CMG soll universell sein und so viele Komponenten wie möglich unterstützen. Jedoch muß man die Liste der zu unterstützenden Komponenten den sinnvollen Anwendungsgebieten anpassen. CMG richtet sich primär an Systeme des unteren Endes mit kleineren Displays.

Die folgenden Aufzählungen sind nicht vollständig, sondern sollen die Hauptanwendungsgebiete zeigen. Wichtig ist nur die Möglichkeit, Erweiterungen mit möglichst geringem Aufwand realisieren zu können. Alle bereits implementierten und unterstützen Teile werden in den folgenden Kapiteln genauer beschrieben.

2.1.1. Plattformen

Direkte Ansteuerung durch einen 8-Bit Bus oder durch SPI aus Embedded Systems:

- 8-Bit Mikrocontroller (z.B. die AVR-Mega-Serie von Atmel[2])
- 16-Bit Mikrocontroller (z.B. von Siemens)
- 32-Bit Mikrocontroller (z.B. die ARM-Serie, speziell ARM7 von NXP[3] oder Atmel)

Indirekte Ansteuerung aus der X86-Architektur:

- Parallele Schnittstelle
- USB Schnittstelle mit Zusatzbaustein (z.B. von FTDI[4])

Emulation im PC:

- GDI unter Microsoft Windows
- GTK+ zur Plattformunabhängigen Ansteuerung

2.1.2. Display-Controller

CMG soll die gebräuchlichsten 8-Bit Display-Controller unterstützen:

- T6963C von Toshiba[5]
- SED1335 / S1D13305 von Epson[6]
- S1D13700 von Epson
- LDS176 von Leadis[7]
- LH155 von Sharp[8]

[2] Atmel Corporation, San Jose, Ca, USA, http://www.atmel.com/
[3] NXP Semiconductors, Eindhoven, Niederlande, ehemals Philips, http://www.nxp.com/
[4] Future Technology Devices International Limited, Glasgow, United Kingdom, http://www.ftdichip.com/
[5] Toshiba Semiconductor GmbH, Braunschweig, Deutschland, http://www.toshiba-components.com/
[6] Epson Europe Electronics GmbH, München, Deutschland, http://www.epson-electronics.de/
[7] Leadis Technology Inc., Sunnyvale, CA, USA, http://www.leadis.com/
[8] Sharp Electronics (Europe) GmbH, Hamburg, Deutschland, http://www.sharp-world.com

2.1.3. KOMMUNIKATIONSPROTOKOLLE

Die Ansteuerung zwischen Mikrocontroller und Displaycontroller geschieht über verschiedene Protokolle. Jeder Display-Controller muß mindestens eines unterstützen, manche bieten sogar mehrere, auswählbare Modi an. Im Großen und Ganzen sind dies für 8-Bit Controller folgende:

- 8080-Busprotokoll (Intel[9]) mit acht bidirektionalen Datenleitungen und vier Steuerleitungen mit /CS, A0, /RD und /WR
- 6800-Busprotokoll (Motorola[10]) mit ebenfalls acht bidirektionalen Datenleitungen und vier Steuerleitungen mit jedoch /CS, A0, R/W und E
- SPI (Serial Peripheral Interface) mit nur vier Datenleitungen mit MOSI, MISO, SCK und /SS

2.2. PROGRAMMIERSPRACHE

Die Wahl der Programmiersprache fällt anhand der Vorgaben recht einfach: Es soll möglichst systemnah, jedoch noch voll portabel sein. Da auch kleine Mikrocontroller unterstützt werden sollen, kann nur eine schlanke Sprache verwendet werden.

Assembler ist nicht plattformunabhängig, C++ zu komplex und speicherhungrig. Somit fiel die Wahl auf C.

Leider bietet einem C nicht den Komfort einer Hochsprache wie C++, C# oder D. Es muß also genau auf mögliche Namenskonflikte und Modulabhängigkeiten aufgepaßt werden. Ein großer Vorteil ist aber, daß es praktisch für jede Plattform einen C-Compiler gibt, meist sogar freie.

2.3. VERWENDETE UND BENÖTIGTE SOFTWARE

2.3.1. EDITOR

Die Wahl des Editors ist im Grunde frei. Der Arbeit liegen jedoch schon Projekte für *Programmer's Notepad*[11] – einem Open-Source Texteditor für Programmierer – und *Microsoft Visual Studio* bei.

2.3.2. BUILD-PROZEß

Die benötigte Software hängt von der oder den gewünschten Zielplattformen ab. Zu jeder unterstützten Plattform existiert eine make-Datei mit der sich das Projekt erstellen lassen kann.

---

[9] Intel Corporation, Santa Clara, CA, USA, http://www.intel.com/
[10] Motorola, Inc., http://www.motorola.com/
[11] http://www.pnotepad.org/

Für Hardwareprojekte benötigt man einen Cross-Compiler, einen Linker und einen Programmer für die Zielplattform – für Emulationsprojekte nur einen Compiler und einen Linker sowie die nötigen Windows- oder GTK-Bibliotheken.

Zur Zeit werden folgende Plattformen unterstützt:

- **AVR:**
  Compiler und Linker aus *WinAVR*[12] (GNU C), Programmer aus *WinAVR* (`avrdude`) oder aus *Atmel Studio* [13](`stk500`) wenn das *STK500* oder kompatible verwendet wird.

- **ARM7:**
  Compiler, Linker und Programmer aus *OpenOCD*[14] oder *YAGARTO*[15].

- **Emulator mit GDI:**
  Für reine Windows-Emulation kann direkt die Visual-Studio-Projektdatei verwendet werden. Damit sind keine weiteren Abhängigkeiten vorhanden. Soll CMG mit einem anderen Compiler übersetzt werden, müssen lediglich die Windows-Header mit `<window.h>` zugänglich sein.

- **Emulator mit GTK+:**
  Für eine plattformunabhängige Emulation kann *GTK+*[16] – das GIMP-Toolkit – verwendet werden. CMG verwendet nur die Include-Dateien `<gtk/gtk.h>` und `<glib.h>`. Um *GTK+* verwenden zu können, müssen für die Zielplattform die Thread-Funktionen aus *GTK+* verfügbar sein (siehe GTK-Dokumentation).

[12] WinAVR, http://winavr.sourceforge.net/
[13] Atmel STK500, ATSTK500, http://www.atmel.com/
[14] OpenOCD, Open On-Chip Debugger, Dominic Rath, http://openocd.berlios.de/web/
[15] YAGARTO, Yet another GNU ARM toolchain, Michael Fischer, www.yagarto.de
[16] GTK+, GIMP-Toolkit, http://www.gtk.org/

2.4. EINFÜHRUNG IN CMG

CMG kann in zwei verschiedenen Betriebsmodi verwendet werden:

- Für kleine Controller und einfacher Grafikausgabe den **direkten Modus**.
- Für aufwendige Grafiken, mehreren Fenstern und Dialogen den **GUI-Modus**.

CMG ist in folgende Module aufgeteilt:

Direkter Modus:

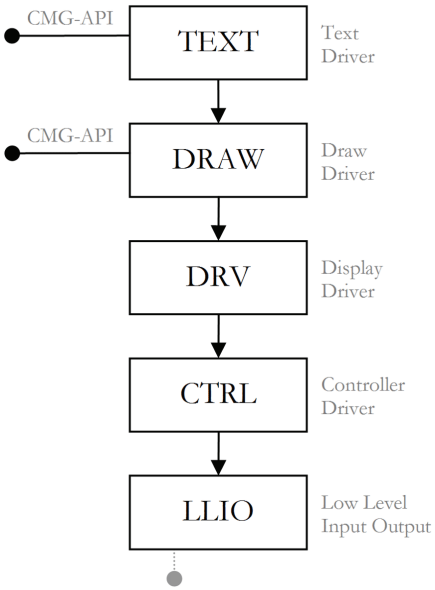


Abbildung 1 – CMG Aufbau 1

GUI-Modus:

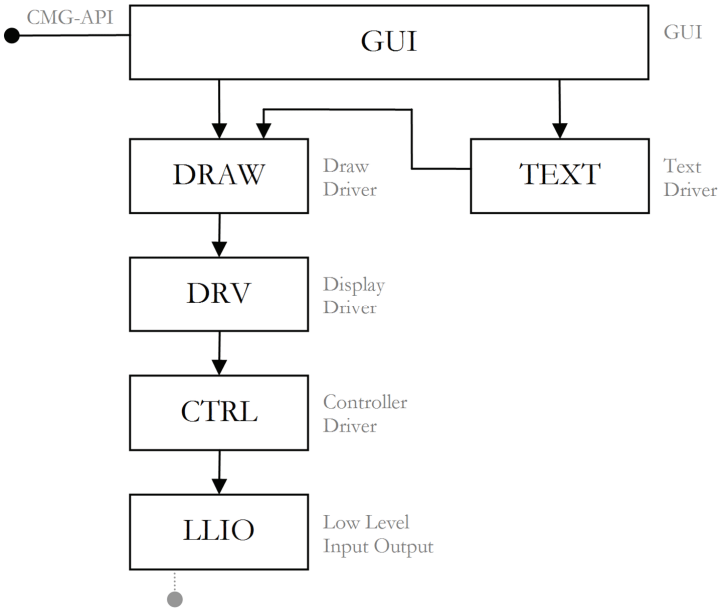


Abbildung 2 – ARM Board, Rückseite

Hier ist zu sehen, daß der GUI-Modus nur die eigentliche Schnittstelle des direkten Modus' kapselt und erweitert.

Der GUI-Modus ist in CMG noch nicht implementiert. Es ist aber wichtig, die Abhängigkeiten und notwendigen Anpassungen bereits früh zu erkennen. Das Kapitel 11 ist dem Thema GUI, den dazu stattgefundenen Überlegungen und den Implementierungsmöglichkeiten gewidmet. Im Folgenden wird allerdings nicht mehr ständig auf diesen Modus eingegangen, sondern vorrangig der direkte Modus behandelt.

Benutzerprogramme, die CMG verwenden, greifen nur auf die Module TEXT und DRAW zu. Diese beiden Schnittstellen bilden die CMG-API.

Um die Aufgaben der einzelnen Module besser zu verstehen, nachfolgend eine Zusammenfassung der Funktionalität:

Abkürzung	Bedeutung	Erklärung
LLIO	**Low Level Input Output**	Direkte Kommunikation mit dem Display über physikalische Anschlüsse, dem Bus. Ergebnis ist eine plattform- und bus-unahängige Kommunikation mit dem Displaycontroller.
CTRL	**Controller**	Unterschiedliche Ansteuerungen der Displaycontroller und Implementierung des standardisierten Zugriffsinterfaces von CMG. Ergebnis ist ein controllerunabhängiger Zugriff auf das Display.

DRV	**Driver**	Die untere Schicht des Grafiktreibers mit grundlegenden Grafikfunktionen und Stift-Unterstützung.
DRAW	**Drawing**	Alle Zeichenoperationen – diese Schicht ist die direkte Benutzer-schnittstelle für Anwendungsprogramme. Hier findet die Trennung zwischen physikalischer Displayauflösung und Orientierung von der eigentlichen Anzeige statt.
TEXT	**Text Rendering**	Alle Funktionen zum Umgang mit Text: Schriftarten laden, Texte vermessen und bearbeiten sowie das eigentliche Ausgeben dieser Texte.
GUI	**User Interface**	Grafisches Benutzerinterface: Aufteilung des Displays in einzelne, voneinander unabhängige, Fenster mit ihren eigenen Nachrichtensystemen.

Tabelle 3 – Beschreibungen der einzelnen Module von CMG

2.5. AUFBAU DER EINZELNEN MODULE

Jedes Modul hat eine bestimmte Grundfunktionalität, die für alle Module gilt sowie das für diese Schicht notwendige, spezifizierte Interface zu implementieren.

Die Grundfunktionalität beläuft sich auf folgende Funktionen und deren interner Ablauf:

- `Init()`: Ruft zuerst die Init-Funktion der nächst niedrigeren Schicht auf und führt anschließend die für die aktuelle Schicht notwendigen Initialisierungen durch. Somit ist sichergestellt, daß alle Schichten in der korrekten Reihenfolge – von der niedrigsten bis hin zur höchsten – initialisiert werden.

- `Exit()`: Führt – im Gegensatz zur Init-Funktion – zuerst die zum Beenden notwendigen Schritte aus und ruft danach die Exit-Funktion der nächst niedrigeren Schicht auf. Somit ist die Reihenfolge genau umgekehrt: Von der höchsten zur niedrigsten Schicht.

- `PowerManagement(…)`: Diese Funktion dient zum Setzen eines bestimmten Displayzustands. Die möglichen Parameter sind in einer globalen, dem Benutzer zugänglichen, Datei definiert. Die Funktion führt je nach gewünschter Aktion die notwendigen Schritte aus und reicht den Aufruf eventuell zur nächst niedrigeren Schicht weiter. Kann die Schicht mit der gewählten Aktion nichts anfangen, so hat sie die unveränderten Parameter ebenfalls an die nächste Schicht weiterzugeben.

2.6. VERZEICHNISAUFBAU

Die Quellcodedateien der einzelnen Module sind in den gleichnamigen Unterverzeichnissen des Moduls abgespeichert.

Alle Dateinamen sind nach folgendem Schema aufgebaut:
`CMG_<Modul>_<Funktion>.<ext>`

Nur die Dateien im Root-Verzeichnis besitzen kein Modul und sehen damit so aus:
`CMG_<Funktion>.<ext>`

2.7. CODING GUIDELINES

Um die Lesbarkeit zu erhöhen und unnötige Fehlerquellen zu vermeiden, verwende ich bestimme Coding-Richtlinien. Im Folgenden möchte ich eine kurze Übersicht darüber geben.

2.7.1. FUNKTIONEN

- **Öffentliche Funktionen**:
  Der Name einer öffentlichen Funktion setzt sich aus dem Präfix „CMG_“, dem jeweiligen Modulnamen und der Funktionsbeschreibung zusammen.

  Das Makro `_PUBLIC` wird in einen leeren String aufgelöst, da bei C standardmäßig alle Funktionen öffentlich sind.

  Syntax:
  ```
  _PUBLIC
  CMG_<Modul>_<Funktion>(…)
  ```

  Beispiel:
  ```
  _PUBLIC
  CMG_LLIO_INIT()
  ```

- **Private Funktionen**:
  Der Name einer privaten Funktion fängt nicht mit dem Präfix „CMG_“ an, sondern nur mit „_“. Anschließend kommen wieder der aktuelle Modulname und zuletzt die Funktionsbeschreibung:

  Das Makro `_PRIVATE` wird in `static` aufgelöst und ist somit nur aus der aktuellen Quellcodedatei referenzierbar.

  Syntax:
  ```
  _PRIVATE
  _<Modul>_<Funktion>(…)
  ```

  Beispiel:
  ```
  _PRIVATE
  _CTRL_SendCommand(…)
  ```

2.7.2. VARIABLEN

- **Lokale Variablenbenennung**:
  Alle Variablen haben einen aussagekräftigen, meist ausgeschriebenen Namen – also statt zum Beispiel `ccx` eher `CurrentCursorX`. Damit wird der Code zwar länger, jedoch wesentlich einfacher zu lesen, da sofort klar wird, was gemeint ist. Wie aus dem Beispiel zu erkennen ist, werden die Teilwörter zusammen geschrieben und jeweils das erste Zeichen davon mit einem Großbuchstaben.

  Alle Variablen besitzen eine Typkennzeichnung in Form eines kleingeschriebenen Präfixes. Dieses Präfix besteht wiederum aus einem optionalen Präfix und der Typbeschreibung.

  Syntax:
  Im Groben ist die Syntax ähnlich der bekannten *Ungarischen Notation*[17], jedoch unterscheiden sich einige Präfixe und die Typbeschreibungen. Im Anschluß eine kurze, nicht vollständige Übersicht:

Typ	Bedeutung	Beispiel
`by`	Byte, 8 Bit, unsigned	`byIndex`, `byLenBits`, `byColor`
`w`	Word, 16 Bit, unsigned	`wCursorOffset`
`dw`	DWord, 32 Bit, unsigned	`dwMaxNeededSize`
`i`	Integer, signed, ohne bestimmte Größe	`iX1`, `iY2`, `iRadius`, `iImageWidth`
`u`	Unsigned, ohne bestimmte Größe	`uNumberOfPoints`, `uSuppressMask`
`b`	Boolean, je nach Plattform	`bFillRange, bCloseList`
<*ohne*>	Für eigene und spezielle Typen	`FillColor`, `Pen`, `Brush`

Tabelle 4 – Variablennotation, Typ

Zusätzlich zum eigentlichen Typ können noch weitere Präfixe – auch Kombinationen untereinander – kommen:

Typ	Bedeutung	Beispiel
`p`	Pointer	`pbyPenStartAddr`, `pBrush`, `pImage`, `piResult`
`a`	Array	`aiLenList`, `paiCoords`, `padwStartAddresses`

Tabelle 5 – Variablennotation, Präfix

Diese Variablenbenennung mag zunächst vielleicht umständlich oder verwirrend aussehen, zahlt sich aber im Code schnell aus und ist danach gut lesbar und verständlich. Sehen Sie sich Codebeispiele an und entscheiden Sie selbst.

- **Globale Variablenbenennung**:
  Globale Variablen sind wie die lokalen aufgebaut, besitzen jedoch noch vorangestellt ein zusätzliches „g_“ mit anschließendem Modulnamen. Dieser Aufwand wird betrieben, damit es keine Namenskonflikte innerhalb CMG oder zwischen CMG und dem Benutzercode gibt.

  Syntax:
  `g_<Modul>_prefix<variableName>`

  Beispiel:
  `g_CTRL_iDisplayWidth`

[17] Ungarische Notation, Wikipedia, 23.05.2007, http://de.wikipedia.org/w/index.php?title=Ungarische_Notation&oldid=32223631

- **Makros und konstante Werte**:
  Makros und konstante Werte werden komplett groß geschrieben und beinhalten ebenfalls den Modulnamen. Bei lokalen Makros und Werten allerdings auch in verkürzter Form, damit diese Werte nicht allzu lang werden. Sie sind ohnehin nur innerhalb der jeweiligen Datei gültig.

  Bei parameterlosen Makros wird im Code auf die Klammern verzichtet, damit deutlich wird, daß es sich um einen Makroaufruf handelt. Das abschließende Semikolon ist jedoch Pflicht. In einem Makro ist stets darauf zu achten, daß es als **ein** logischer Block definiert ist, also von geschweiften oder runden Klammern (je nach Kontext) umgeben ist.

  Beispiel:
  `SET_DATA_PORT_DIRECTION_IN;`

2.8. QUELLCODEAUFBAU

2.8.1. QUELLCODEAUFBAU

Jede CMG-Quellcodedatei besitzt ein gemeinsames Grundgerüst, den Grundaufbau:

```
//!*****************************************************************************
// CMG_$MODULE$_$NAME$.c
//!*****************************************************************************
// CMG - Christian Merkle Graphics
// Copyright Information
// [...]
//!*****************************************************************************
// Author:          Christian Merkle
// Created:         ??.??.200?
//!*****************************************************************************
// History:
//   ??.??.200?     *stable*
//!*****************************************************************************

// main CMG include (will include user configuration)
#include    "../CMG_Main.h"


//-----------------------------------------------------------------------------
// conditional compilation and configuration checks
//-----------------------------------------------------------------------------
#ifndef CMG_$MODULE$_CONFIGURED
    #error CMG_$MODULE$ not configured. Please customize the config file correctly (use the *.config
                        template).
#elif defined( CMG_$MODULE$_$NAME$ )

    #if !defined( CMG_$MODULE$_$SOME_NEEDED_DEFINE$ )
        #error CMG_$MODULE$_* are not defined. Please customize the config file correctly using [...]
    #endif
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
// inludes
//-----------------------------------------------------------------------------
// CMG:
#include    "CMG_$MODULE$.h"
#include    "../1_LLIO/CMG_$NEEDED_MODULE$.h"
[...]
// other:
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
// defines
//-----------------------------------------------------------------------------
[...]
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
// globals
//-----------------------------------------------------------------------------
[...]
//-----------------------------------------------------------------------------
```

```
//---------------------------------------------------------------------------
// functions
//---------------------------------------------------------------------------
[...]
//---------------------------------------------------------------------------

#endif

////////////////////////////////////////////////////////////////////////////
// (c) 2006, 2007 Christian Merkle
```

Listing 1 – Grundaufbau von CMG-Quellcodedateien

Der Quellcode beginnt mit dem Info-Block. Darin enthalten sind der Dateiname, die Copyright-Informationen, der Autor, das Erstellungsdatum, das Freigabedatum sowie relevante Änderungen nach der Freigabe.

Die Platzhalter `$MODULE$` und `$NAME$` sind durch die passenden Werte zu ersetzen.

Anschließend kommt die Include-Anweisung für `CMG_Main.h`. Diese ist zwingend als erstes Include notwendig und stets außerhalb jeglicher `#if`-Blöcke zu halten. Die genaue Beschreibung von `CMG_Main.h` finden Sie in Kapitel 4.1.

Danach findet eine Sicherheitsüberprüfung statt, ob `CMG_$MODULE$_CONFIGURED` definiert ist. Der komplette, restliche Code befindet sich im `#elif defined(…)`-Block.

Darin befinden sich die nötigen Includes: Zuerst die der eigenen Schicht, danach die der referenzierten CMG-Schichten und zuletzt die von extern benötigten Bibliotheken.

Anschließend folgen noch die Defines, die globalen Variablen und die Funktionen.

2.8.2. CONFIG-DATEIEN

Jede Schicht besitzt eine Config-Datei, d.h. eine Vorlage des jeweiligen Config-Abschnitts für die Benutzer-Config-Datei.

Der Benutzer kopiert sich den Inhalt aller Konfigurationsdateien in seine eigene Config-Datei, welche er für sein System und seine Konfiguration anpaßt.

Das Grundgerüst jeder Konfigurationsdatei sieht wie folgt aus:

```
//////////////////////////////////////////////////////////////////////////////
// (2) CMG_$MODULE$ - CONFIG
//////////////////////////////////////////////////////////////////////////////
//
// CMG_$MODULE$_*
// ======================
// $MODULE$ configuration.
//
//
// For internal correct compilation.
// *** DO NOT CHANGE! ***
#define     CMG_$MODULE$_CONFIGURED
//
//
//============================================================================
// User Area:
// ==========
// Read the sections carefully and change the values accordingly to
// your needs.
//
//
// $GROUP_TOPIC$:
// ======================
// $NEXT_LINE_DESCRIPTION$
#define     CMG_$MODULE$_$VARIABLE$                 $VALUE_TO_SET$
//
//
//////////////////////////////////////////////////////////////////////////////
```

Listing 2 – Config-Datei, Grundgerüst

Nach der Beschreibung und dem Hinweis, daß der Benutzer im oberen Bereich nichts ändern sollte, wird der Wert `CMG_$MODULE$_CONFIGURED` definiert. Dieser sollte Ihnen ein Begriff sein; er wurde im letzten Abschnitt vorgestellt und als Bestätigung der Konfiguration eines Moduls verwendet.

Im Benutzerbereich, der sogenannten *User Area*, muß der Benutzer für diese Schicht spezielle Einstellungen vornehmen. Hier reichen die Einstellungen von zu benutzenden Quellcodedateien (Auswählen einzelner Treiber) bis hin zur Definition der Displayauflösung.

2.9. Erstellen des Projekts

Um CMG kompilieren zu können, müssen Sie sich zuerst die gerade besprochene Config-Datei nach Ihren Anforderungen und Wünschen erstellen.

Anschließend kann CMG kompiliert werden. Zur Zeit stehen zwei grundsätzliche Möglichkeiten zur Verfügung: Makefiles und zwei Visual-Studio-Projekte.

Im Visual-Studio-Projekt können Sie nur den Emulator für Windows – und den für GTK+ natürlich auch – kompilieren, dafür das Projekt aber komfortabel bearbeiten und debuggen.

Für alle anderen Systeme und Plattformen stehen Makefiles zur Verfügung. Diese befinden sich im Verzeichnis `Source/make/*`. Zur Zeit existieren hier folgende Unterverzeichnisse:

- `arm/`: Makefile für ARMs
- `avr/`: Makefile für AVRs
- `gtk/`: Makefile für GTK+
- `win/`: Windows Visual Studio 6 Projekt

Als Hilfsfunktion befindet sich im CMG-Rootverzeichnis des Quellcodes eine Make-Include-Datei namens `MakeInclude`.

Sie können sich nun selbst beliebige Makefiles schreiben und eine Liste aller CMG-Quellcodedateien mit folgendem Aufruf erhalten:

```
[...]

# CMG files
CMG_ROOT = ../../CMG/
-include $(CMG_ROOT)MakeInclude


# List C source files here.
SRC = ../../$(TARGET).c $(CMG_SOURCES)

[...]
```

Listing 3 – Einbinden von MakeInclude in eigene Makefiles

Vor dem `-include`-Aufruf muß `CMG_ROOT` mit dem Root-Verzeichnis von CMG definiert sein. Danach befindet sich die Liste in `CMG_SOURCES`.

Die genaue Beschreibung von MakeInclude finden Sie in Kapitel 4.5.

3. HARDWARE

Dieses Kapitel behandelt primär nicht alle mögliche, unterstützte Hardware, sondern nur die während der Arbeit geplante, erstellte, benutzte und getestete.

3.1. GEMEINSAMES INTERFACE

Um jedes Display mit jeder Plattform möglichst einfach testen zu können, muß dafür ein gemeinsames Interface geschaffen werden. Jedes Display und jede Plattform bekommt eine genormte Steckverbindung, mit der man die Komponenten miteinander verbinden kann.

Was genau muß also dieser Verbinder beinhalten?

Zuerst einmal zwei Kontakte für die Stromversorgung mit +5V und Masse. Die meisten Displays benötigen 5V Betriebsspannung, manche arbeiten auch mit 3.3V. Für die Konvertierung und Pegelanpassung der Signale ist die Displayseite verantwortlich. Diese Aufgabe übernimmt das Konverter-Board, welches später in diesem Kapitel noch vorgestellt wird.

Nahezu alle Displays unterstützen ein 8-Bit Protokoll – entweder im 8080- oder im 6800-Format. Es liegt also nahe darauf aufzubauen. Dafür werden acht bidirektionale Datenleitungen und vier auf Ausgang geschaltete Steuerleitungen benötigt, von denen zwei vom verwendeten Protokoll abhängig sind.

Zusätzlich unterstützt das Interface noch zwei weitere, optionale Signale: Das Erste zum Ein- und Ausschalten des Displays, das Zweite zum Aktivieren der Hintergrundbeleuchtung. Diese Funktionen können – müssen jedoch nicht – implementiert sein. Auch ist es möglich, einzelne Signale zu invertieren. Die jeweilige Hardware-Konfiguration wird in CMG in den Config-Dateien eingestellt.

Die Interface-Belegung sieht somit wie folgt aus:

PIN	Richt.	Signal	Beschreibung
1	O	PWR	*Power*, Display An/Aus
2	O	BL	*Backlight*, Hintergrundbeleuchtung
3	O	/CS	*Chip Select*
4	O	A0	*Address Select*
5	O	/RD, R/W	*Read* bzw. *Read Not Write*
6	O	/WR, E	*Write* bzw. *Enable*
7	I/O	D0	Datenleitung 0
8	I/O	D1	Datenleitung 1
…	…	…	…
13	I/O	D6	Datenleitung 6
14	I/O	D7	Datenleitung 7
15		GND	Masse
16		VTG	+5V Versorgung

Tabelle 6 – Pinbelegung Gemeinsames Interface

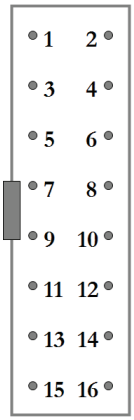


Abbildung 3 – Connector, 16-polig

3.2. PLATTFORMEN

3.2.1. AVR

Die Mikrocontroller der AVR-Reihe von Atmel sind ein komplettes *System-On-A-Chip* im Low-End-Bereich. Sie sind 8-Bit RISC-Prozessoren mit maximalen Taktraten zwischen 16MHz und 20MHz und einer reichhaltigen Peripherie an Bord.

Um CMG ausführen zu können, muß eine gewisse Speichergröße – sowohl für Programmcode, als auch für Daten – vorhanden sein. Der Programmcode belegt zur Zeit – je nach Konfiguration – zwischen 10kB und 16kB. Sollen zusätzliche Schriften unterstützt werden, steigt diese Zahl schnell an und externe Speicher werden nötig. Deshalb sollte zum vernünftigen Arbeiten mit CMG mindestens ein *ATmega32* oder größer verwendet werden.

Zum Programmieren kann jeder beliebige Programmer verwendet werden, hier wurde das STK500 von Atmel[18] [STK500] verwendet:

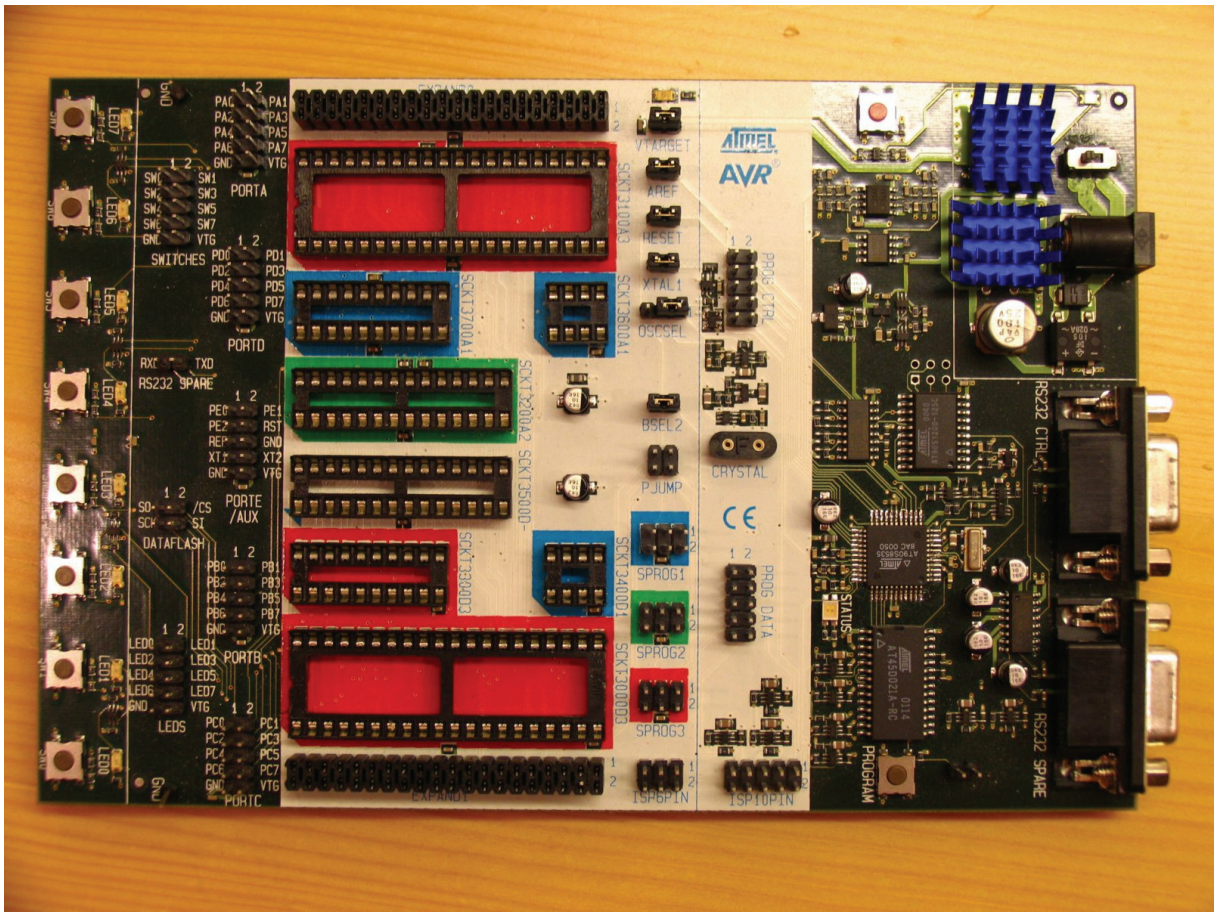


Abbildung 4 – Atmel STK500

Alle Ports des Mikrocontrollers werden auf Verbinder (links im Bild, PORTA - E) herausgeführt. Um nicht direkt auf dem Board zu löten wird ein Adapterkabel, der das Standardinterface bietet, erstellt.

[18] Atmel Corporation, San Jose, Ca, USA, http://www.atmel.com/

Die Pinbelegung der Steuerleitungen kann beliebig gewählt werden, nur die 8 Datenleitungen müssen zusammen an einem Port liegen. Für meine Zwecke habe ich Port A für die Daten- und Port C für die Steuerleitungen gewählt. Somit besitzt das Adapterkabel folgende Zuordnung:

AVR Port/Pin	PIN	Richt.	Signal	Beschreibung
PORTC.2	1	O	PWR	*Power*, Display An/Aus
PORTC.3	2	O	BL	*Backlight*, Hintergrundbeleuchtung
PORTC.4	3	O	/CS	*Chip Select*
PORTC.5	4	O	A0	*Address Select*
PORTC.6	5	O	/RD, R/W	*Read* bzw. *Read Not Write*
PORTC.7	6	O	/WR, E	*Write* bzw. *Enable*
PORTA.0	7	I/O	D0	Datenleitung 0
PORTA.1	8	I/O	D1	Datenleitung 1
…	…	…	…	…
PORTA.6	13	I/O	D6	Datenleitung 6
PORTA.7	14	I/O	D7	Datenleitung 7
PORTA.GND	15		GND	Masse
PORTA.VTG	16		VTG	+5V Versorgung

Tabelle 7 – AVR Adapter für Interface

Die benötigten 5V und Masse werden direkt aus dem Connector von Port A entnommen.

Das fertige Adapterkabel sieht so aus:

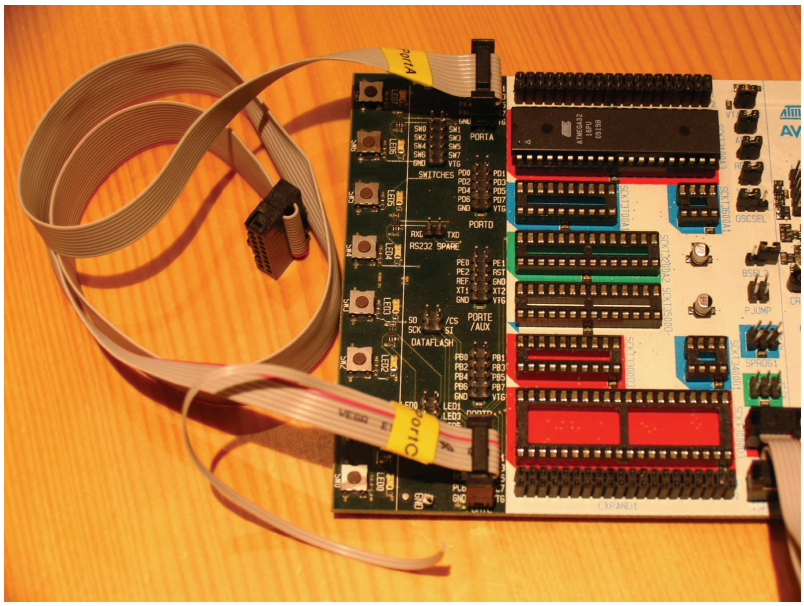


Abbildung 5 – AVR Adapterkabel

3.2.2. ARM

Zum Entwickeln und Testen der ARM-Komponenten wurde der ARM7-Mikrocontroller *LPC2292* von NXP[19], verwendet. Dieser besitzt 256kB Flash-Speicher für Code und 16kB RAM. Er sitzt auf dem Board *LPCEB2000-B* von Embest[20], das zusätzlich ein 2MB großes, externes Flash und ein 512kB großes, externes SRAM enthält. Das Board läuft mit 60Mhz.

[19] NXP Semiconductors, Eindhoven, Niederlande, ehemals Philips, http://www.nxp.com/
[20] Embest, Embest Info&Tech Corporation, Shenzhen, Guangdong, China, http://www.armkits.com/

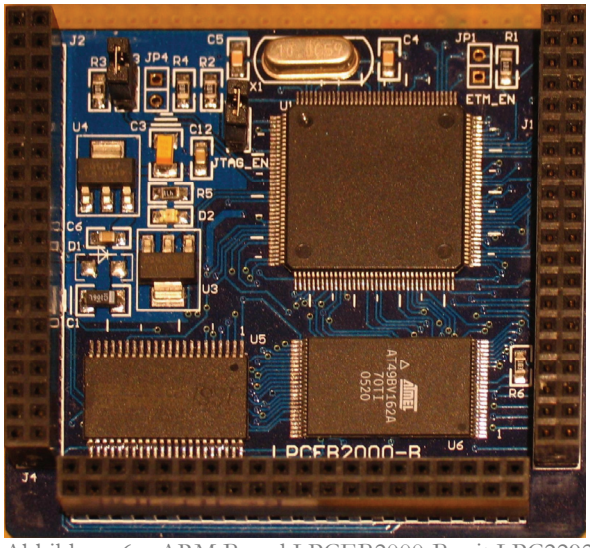


Abbildung 6 – ARM Board LPCEB2000-B mit LPC2292

Programmiert und getestet wird am einfachsten mit *OpenOCD*[21] von Dominic Rath, welcher das JTAG-Interface benutzt. Dazu benötigt man noch eine Komponente zum Ansteuern. Um dies bequem vom PC aus mit USB zu ermöglichen, eignet sich das Board DLP-2232M von FTDI[22] sehr gut:

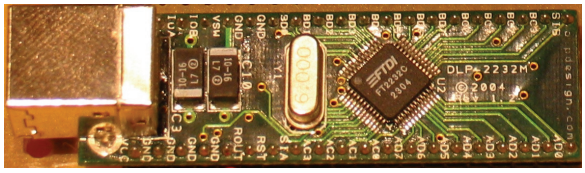


Abbildung 7 – FTDI DLP-2232M Board

Herr Högl stellte mir ein fertig aufgebautes Modul mit beiden Komponenten zur Verfügung, welches nur noch um das Interface erweitert werden mußte. Die einzelnen Ports können direkt von den Pfostensteckern des Boards abgegriffen werden.

Die Daten- und Steuerleitungen können von CMG aus wieder frei definiert werden – nur müssen die Datenleitungen auf acht aufeinander folgende Bits angeschlossen werden. In meinem Beispiel habe ich folgenden Aufbau gewählt:

ARM Port/Pin	PIN	Richt.	Signal	Beschreibung
P0.23	1	O	PWR	*Power*, Display An/Aus
P0.22	2	O	BL	*Backlight*, Hintergrundbeleuchtung
P0.16	3	O	/CS	*Chip Select*
P0.17	4	O	A0	*Address Select*
P0.18	5	O	/RD, R/W	*Read* bzw. *Read Not Write*
P0.19	6	O	/WR, E	*Write* bzw. *Enable*
P0.8	7	I/O	D0	Datenleitung 0
P0.9	8	I/O	D1	Datenleitung 1
…	…	…	…	…
P0.14	13	I/O	D6	Datenleitung 6
P0.15	14	I/O	D7	Datenleitung 7
GND	15		GND	Masse
+5V	16		VTG	+5V Versorgung

Tabelle 8 – ARM Adapter für Interface

[21] OpenOCD, Open On-Chip Debugger, Dominic Rath, http://openocd.berlios.de/web/
[22] FTDI, Future Technology Devices International Ltd., Glasgow, United Kingdom, http://www.ftdichip.com/

Das fertige ARM-Board mit Interface sieht so aus:

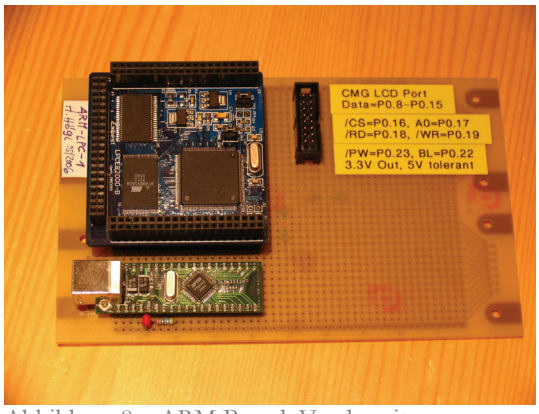


Abbildung 8 – ARM Board, Vorderseite

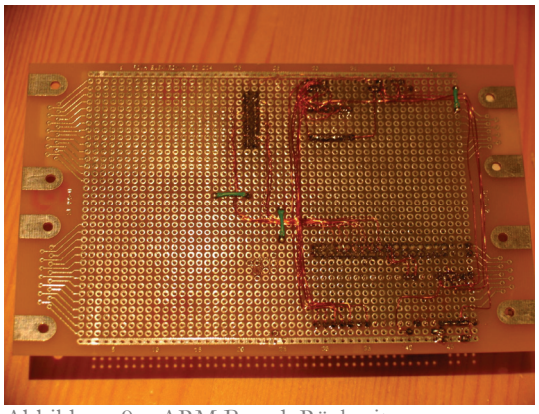


Abbildung 9 – ARM Board, Rückseite

3.2.3. Parallele Schnittstelle

Auch an die parallele Schnittstelle kann das Display direkt angeschlossen werden.

Für die Datenleitungen werden ebenfalls die Datenleitungen der Schnittstelle verwendet. Diese sind im sogenannten *Byte Mode* der Schnittstelle bidirektional ansprechbar. Die aktuelle Richtung wird im Control-Port eingestellt.

Die Steuerleitungen sind nur zur Ausgabe und man kann sie frei unter den Steuerleitungen des Druckerports belegen.

Eine Beispielkonfiguration sieht so aus:

Parallel Port/Pin	PIN	Richt.	Signal	Beschreibung
-na-	1	O	PWR	*Power*, Display An/Aus
-na-	2	O	BL	*Backlight*, Hintergrundbeleuchtung
/STROBE	3	O	/CS	*Chip Select*
/AUTOFEED	4	O	A0	*Address Select*
INIT	5	O	/RD, R/W	*Read* bzw. *Read Not Write*
/SELECT	6	O	/WR, E	*Write* bzw. *Enable*
DATA0	7	I/O	D0	Datenleitung 0
DATA1	8	I/O	D1	Datenleitung 1
…	…	…	…	…
DATA6	13	I/O	D6	Datenleitung 6
DATA7	14	I/O	D7	Datenleitung 7
GND	15		GND	Masse
+5V (extra)	16		VTG	+5V Versorgung

Tabelle 9 – AVR Adapter für Interface

Durch die beschränkten Möglichkeiten der parallelen Schnittstelle gibt es auch Einschränkungen im Betrieb: Die Signale *PWR* und *BL* können nicht herausgeführt werden. Das bedeutet, daß das Display immer eingeschaltet ist und die Hintergrundbeleuchtung fest ein- oder ausgeschaltet werden muß.

Zusätzlich muß eine externe Stromversorgung vorhanden sein, da der parallele Port nicht genügend Strom für die Displays liefern kann.

3.3. LEVEL-CONVERTER-BOARD

Das Level-Converter-Board (*LCB*) hat gleich zwei Aufgaben:

Einerseits benötigen viele Displays zusätzlich zur 5V Logik-Versorgung eine Kontrastspannung. Um dies nicht bei jedem Display neu zu bauen, wird diese Funktion auf ein Modul ausgelagert. Es erzeugt mit Hilfe eines DC-DC-Wandlers [AM1S] aus der 5V Eingangsspannung eine potentialfreie Spannung von 24V. Damit können Kontrastspannungen im Bereich von -24V bis +24V bereitgestellt werden.

Andererseits arbeiten ARM-Prozessoren mit nur 3.3V IO-Spannung. Die Eingänge sind zwar 5V-tolerant, die Ausgänge geben jedoch nur knapp über 3V bei einem Hi-Pegel aus, was für manche Display-Controller nicht immer zur sicheren Eingabe ausreicht. Die sicheren Pegel können in den Datenblättern nachgelesen werden. Das Level-Converter-Board kann die Betriebsspannung des Displays mit Hilfe eines Schalters setzen.

Das Board wird über einen standardisieren 6-poligen Stecker mit dem dazugehörigem Display-Board verbunden.

Die Steckverbindung ist wie folgt definiert:

PIN	Richtung	Signal	Beschreibung
1	→ LCB	+IN	*Power In* (5V)
2	LCB →	+OUT	*Power Out* (5V oder 3.3V)
3	LCB →	+24	*Display Contrast* +24V
4	LCB →	-24	*Display Contrast* -24V
5	→ LCB	GND	*Ground*
6		-nc-	not connected

Tabelle 10 – Level-Converter-Board, Steckbelegung

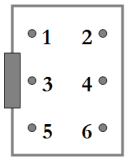


Abbildung 10 – Connector, 6-polig

Das Schaltbild des Boards sieht so aus:

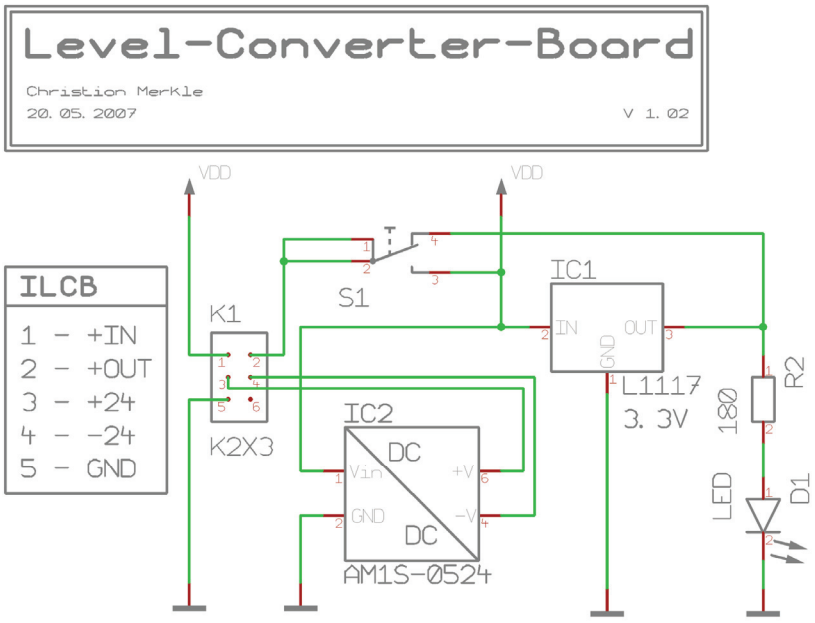


Abbildung 11 – Schaltplan Level-Converter-Board

Hier das fertig aufgebaute Board:

Abbildung 12 – Level-Converter-Board, Vorderseite

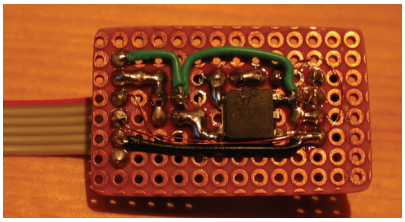


Abbildung 13 – Level-Converter-Board, Rückseite

Ein LCD-Board, welches zusätzlich zu den vorhandenen 5V aus dem Anschluß eine Kontrastspannung oder eine andere Betriebsspannung als 5V benötigt, sollte dieses Level-Converter-Board verwenden. Dazu verbindet es intern nur seinen 5V-Eingang mit dem 5V-Eingang (Pin 1) des LCBs. Seine Versorgungsspannung bezieht es aus dem Ausgang (Pin 2).

Hieraus ergibt sich ein weiterer Vorteil: Es ist sichergestellt, daß das Display nicht ohne Kontrastspannung läuft – wenn z.B. das LCB nicht mit dem LCD-Board verbunden ist.

3.4. Controller und Displays

Die zuvor vorgestellten Plattformen sollen nun mit Hilfe des gemeinsamen Interfaces aus 3.1 an die Displays angeschlossen werden. Diese Aufgabe übernehmen die LCD-Boards.

Sie verbinden den Connector aus 3.1 mit den Anschlüssen des Displays, stellen eventuell vorhandene Logik zum Schalten des Displays und der Hintergrundbeleuchtung bereit und kümmern sich gegebenenfalls um die Bereitstellung der Kontrastspannung.

Im folgenden Abschnitt werden die LCD-Borads – nach Controller sortiert – näher beschrieben:

3.4.1. Powertip PG240128-A, T6963C

Technische Daten zum Display PG240128-A [POWERTIP] von Powertip:

Eigenschaft	Wert
Hersteller	Powertip
Bezeichnung	PG240128-A
Controller	T6963C
Auflösung	240 x 128 [*Pixel*]
Farbtiefe	1 [*bpp*]
Gehäuseabmessungen	144 x 104 [*mm*]
Effektive Anzeigegröße	108 x 57.5 [*mm*]
Pixelgröße	0.40 x 0.40 [*mm*]
Pixelabstand	0.45 x 0.45 [*mm*]
Versorgungsspannung	5V
Kontrastspannung	11V bis 14V, selbst erzeugt

Tabelle 11 – Technische Daten PG240128-A

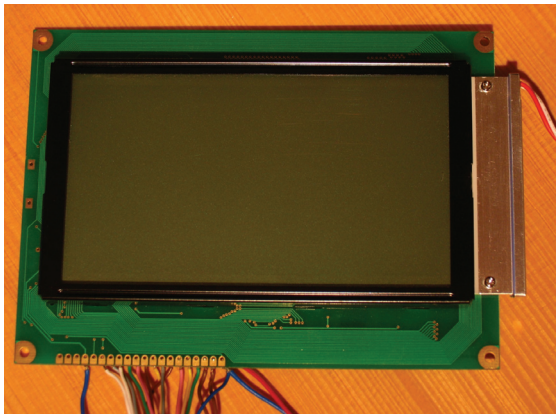


Abbildung 14 – Powertip PG240128-A, Vorderseite

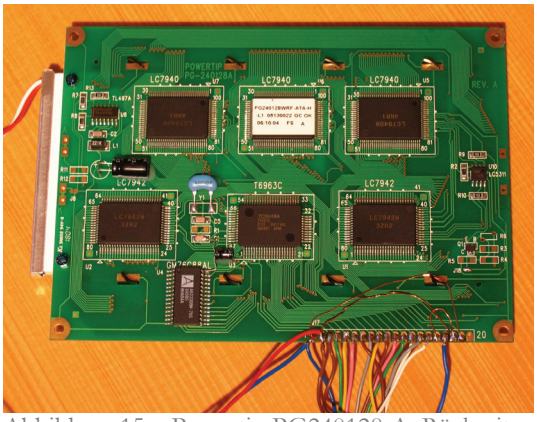


Abbildung 15 – Powertip PG240128-A, Rückseite

Der Controller T6963C [T6963C] ist ein sehr verbreiteter, einfacher Controller von Toshiba[23].

Das Display hat einen 20-poligen Anschluß:

Pin	Symbol	Beschreibung	Zielsymbol
1	Vss	*Power Supply (GND)*	GND
2	Vdd	*Power Supply (+)*	VTG
3	Vo	*Contrast Adjust*	
4	C/D	*Command / Data Select*	A0
5	/RD	*Data Read*	/RD
6	/WR	*Data Write*	/WR
7	DB0	*Data Bus Line 0*	D0
…	…	…	…
14	DB7	*Data Bus Line 7*	D7
15	/CE	*Chip Enable*	/CS
16	/RST	*Reset*	
17	Vee	*Negative Output*	
18	MD2	*Select Number of Columns*	
19	FS1	*Font Selection*	
20	nc	*-not connected-*	-/-

Tabelle 12 – Pinbelegung Powertip PG240128-A aus [POWERTIP]

[23] Toshiba Semiconductor GmbH, Braunschweig, Deutschland, http://www.toshiba-components.com/

Das LCD-Board unterstützt die Signale *PWR* und *BL* und ist wie folgt aufgebaut:

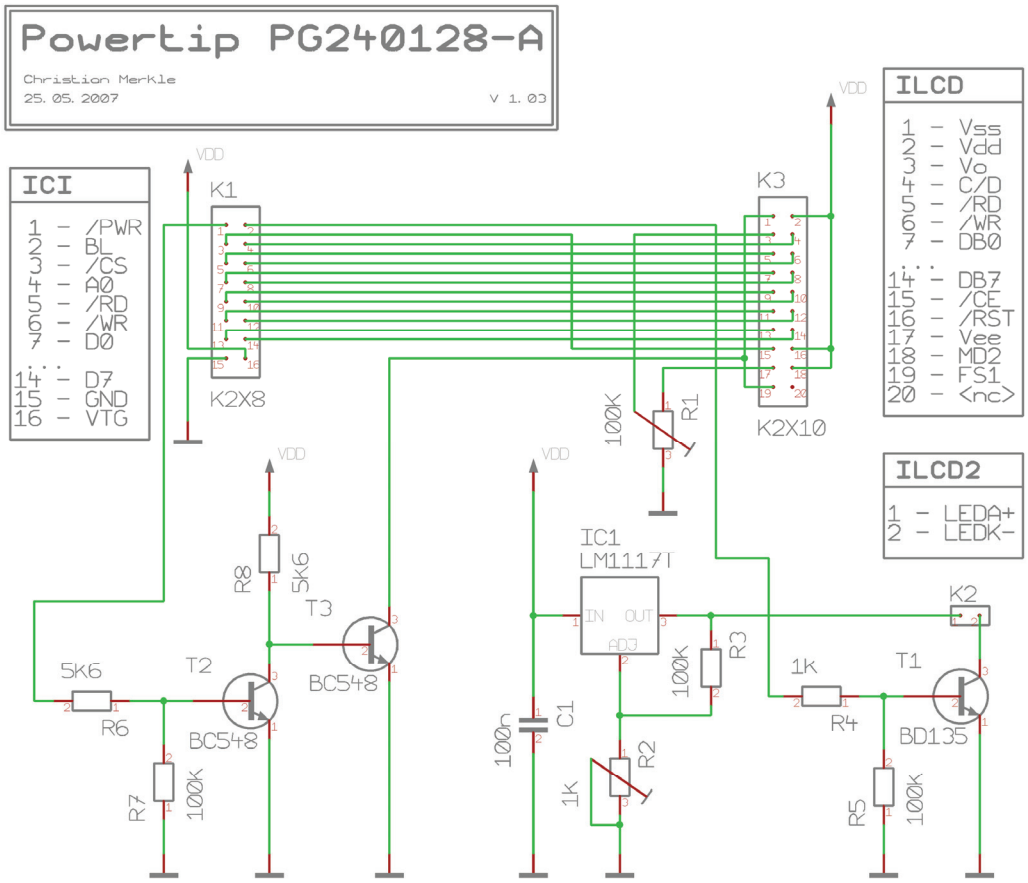


Abbildung 16 – Schaltplan Powertip PG240128-A

Das *PWR*-Signal ist invertiert: Ist *PWR* low, sperrt *T2*. Dadurch leitet *T3* und setzt die Pins 1 und 19 des Displays auf Masse.

Eine weitere Eigenschaft des Displays ist die sehr hohe Stromaufnahme für die Hintergrundbeleuchtung von 900mA bei 5V [POWERTIP]. Um das Interface nicht zu stark zu belasten, muß dieser Wert auf ein ertragbares Niveau gesenkt werden. Dafür verwendet das Board den regelbaren „*Low Dropout Linear Regulator*“ (LDO) des Typs *LM1117T-Adj* [LM1117]. Mit *R2* läßt sich die Ausgangsspannung für die Hintergrundbeleuchtung stufenlos regulieren. *T1* legt bei High des nicht-invertierenden Signals *BL* die Kathoden der LEDs auf Masse.

Der Kontrast kann mit *R1* eingestellt werden. Die dazu notwendige Kontrastspannung erzeugt das Display selbst und stellt diese an Pin 17 zur Verfügung.

Das fertig aufgebaute LCD-Board:

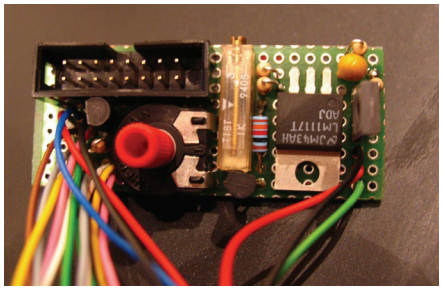


Abbildung 17 – LCD-Board Powertip PG240128-A

3.4.2. DATAVISION DG12864-12, T6963C

Technische Daten zum Display DG12864-12 [DATAVISION] von DataVision:

Eigenschaft	Wert
Hersteller	DataVision
Bezeichnung	DG12864-12
Controller	T6963C
Auflösung	128 x 64 [*Pixel*]
Farbtiefe	1 [*bpp*]
Gehäuseabmessungen	78 x 70 [*mm*]
Effektive Anzeigegröße	56.3 x 38.4 [*mm*]
Pixelgröße	-keine Angaben-
Pixelabstand	-keine Angaben-
Versorgungsspannung	5V
Kontrastspannung	-1V bis -5V

Tabelle 13 – Technische Daten DG12864-12

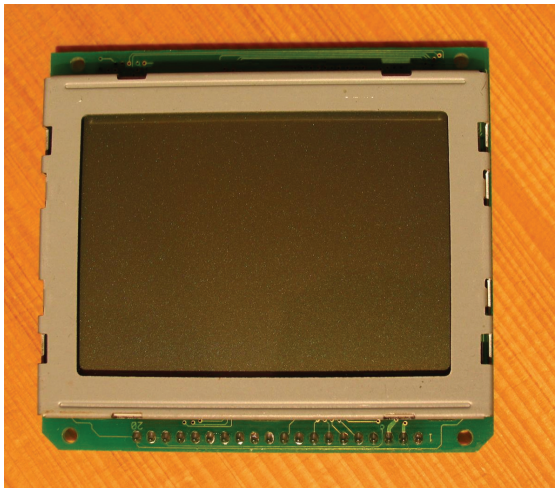


Abbildung 18 – DataVision DG12864-12, Vorderseite

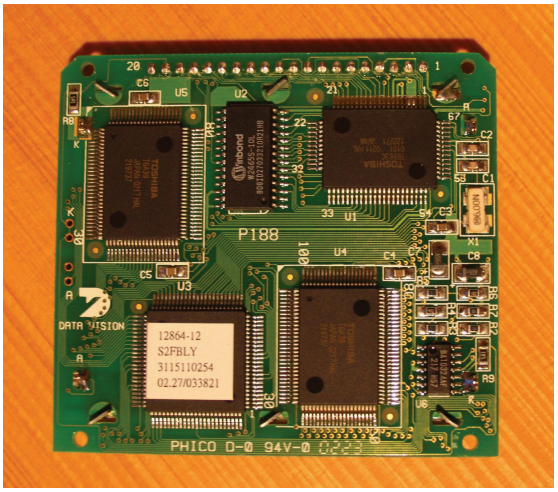


Abbildung 19 – DataVision DG12864-12, Rückseite

Das DG12864-12 verwendet – ebenso wie das oben beschriebene PG240128-A – den T6963C als Controller.

Das Display hat einen 20-poligen Anschluß:

Pin	Symbol	Beschreibung	Zielsymbol
1	FG	*Frame Ground*	GND
2	VSS	*Power Supply (GND)*	GND
3	VDD	*Power Supply (+)*	VTG
4	Vo	*Contrast Adjust*	
5	/WR	*Data Write*	/WR
6	/RD	*Data Read*	/RD
7	/CE	*Chip Enable*	/CS
8	C/D	*Command / Data Select*	A0
9	/RST	*Reset*	
10	DB0	*Data Bus Line 0*	D0
…	…	…	…
17	DB7	*Data Bus Line 7*	D7
18	FS	*Font Select*	
19	LED K	*Backlight Cathode*	
20	LED A	*Backlight Anode*	

Tabelle 14 – Pinbelegung DataVision DG12864-12 aus [DATAVISION]

Das LCD-Board unterstützt die Signale *PWR* und *BL* und ist wie folgt aufgebaut:

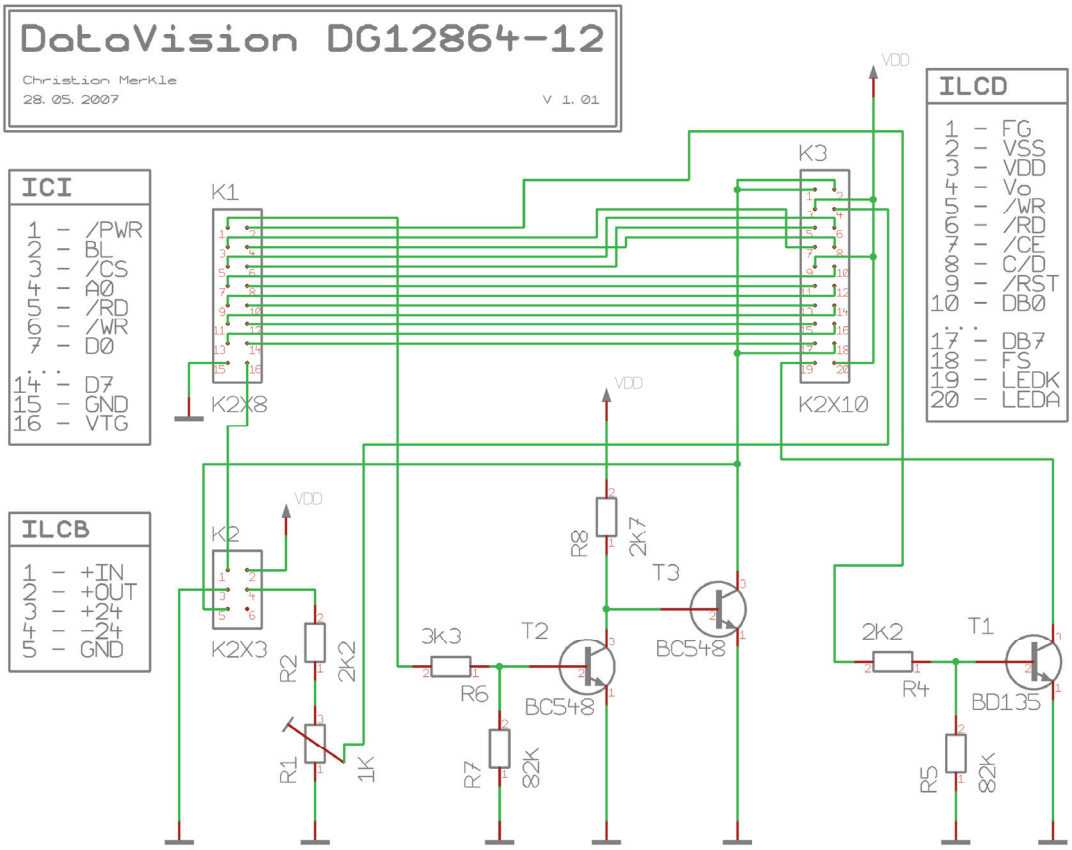


Abbildung 20 – Schaltplan DataVision DG12864-12

Das *PWR*-Signal ist ebenfalls wieder invertiert. Dadurch ist das Display auch ohne Verbindung eingeschaltet, also auch wenn sich der Mikrocontroller am *PWR*-Ausgang im Z-Zustand befindet.

Die Hintergrundbeleuchtung wird wie bei dem vorherigen Display gesteuert.

Die Kontrastspannung wird diesmal aber über das LCB erzeugt, mit *R2* auf gültige Werte beschränkt und kann mit *R1* eingestellt werden.

Das fertig aufgebaute LCD-Board:

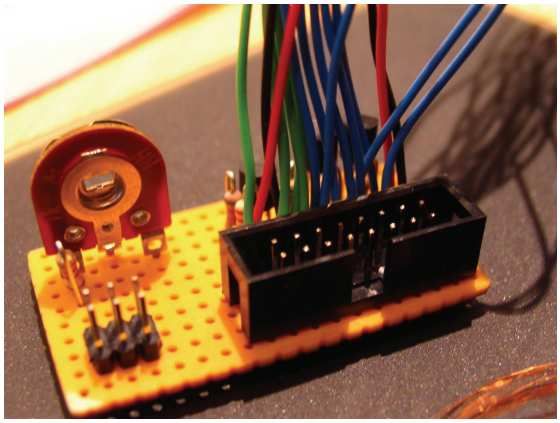


Abbildung 21 – LCD-Board DataVision DG12864-12

3.4.3. EDT EW32FY0FLW, S1D13700

Technische Daten zum Display EW32FY0FLW [EW32] von EDT:

Eigenschaft	Wert
Hersteller	EDT
Bezeichnung	EW32FY0FLW
Controller	S1D13700
Auflösung	320 x 240 [*Pixel*]
Farbtiefe	1, 2 oder 4 [*bpp*]
Gehäuseabmessungen	94 x 67 [*mm*]
Effektive Anzeigegröße	67.8 x 57.6 [*mm*]
Pixelgröße	0.23 [*mm*]
Pixelabstand	0.24 [*mm*]
Versorgungsspannung	3.3V bis 5.5V
Kontrastspannung	19.9V bis 23.5V

Tabelle 15 – Technische Daten EW32FY0FLW

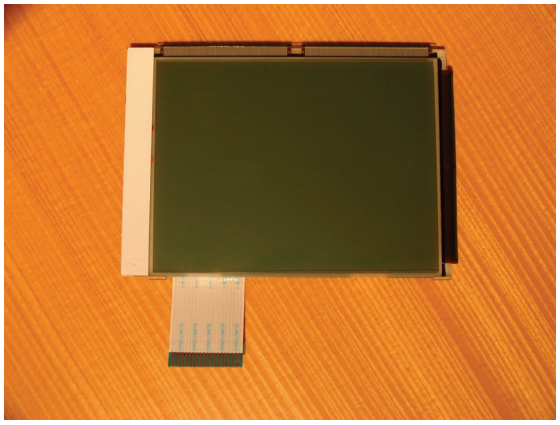


Abbildung 22 – EW32FY0FLW, Vorderseite

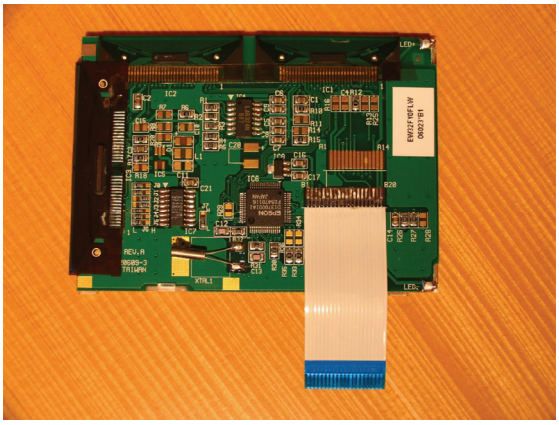


Abbildung 23 – EW32FY0FLW, Rückseite

Das EW32FY0FLW verwendet den wesentlich mächtigeren Controller S1D13700 [S1D13700] von Epson.

Auch dieses Display hat einen 20-poligen Anschluß, allerdings als Folienleiter:

Pin	Symbol	Beschreibung	Zielsymbol
1	VSS	*Power Supply (GND)*	GND
2	VDD	*Power Supply (+)*	VTG
3	-nc-	*Not Connected*	-/-
4	A0	*Command / Data Select*	A0
5	/WR	*Data Write*	/WR
6	/RD	*Data Read*	/RD
7	D0	*Data 0*	D0
…	…	…	…
14	D7	*Data 7*	D7
15	/CS	*Chip Select*	/CS
16	/RST	*Reset*	
17	VEE	*Power Supply for LCD Driving*	
18	SEL1	*8080 or 6800 Select L:8080*	
19	VLED	*Backlight Anode*	
20	VLSS	*Backlight Cathode*	

Tabelle 16 – Pinbelegung EW32FY0FLW aus [EW32]

Das LCD-Board unterstützt nur das Signal *BL* und ist wie folgt aufgebaut:

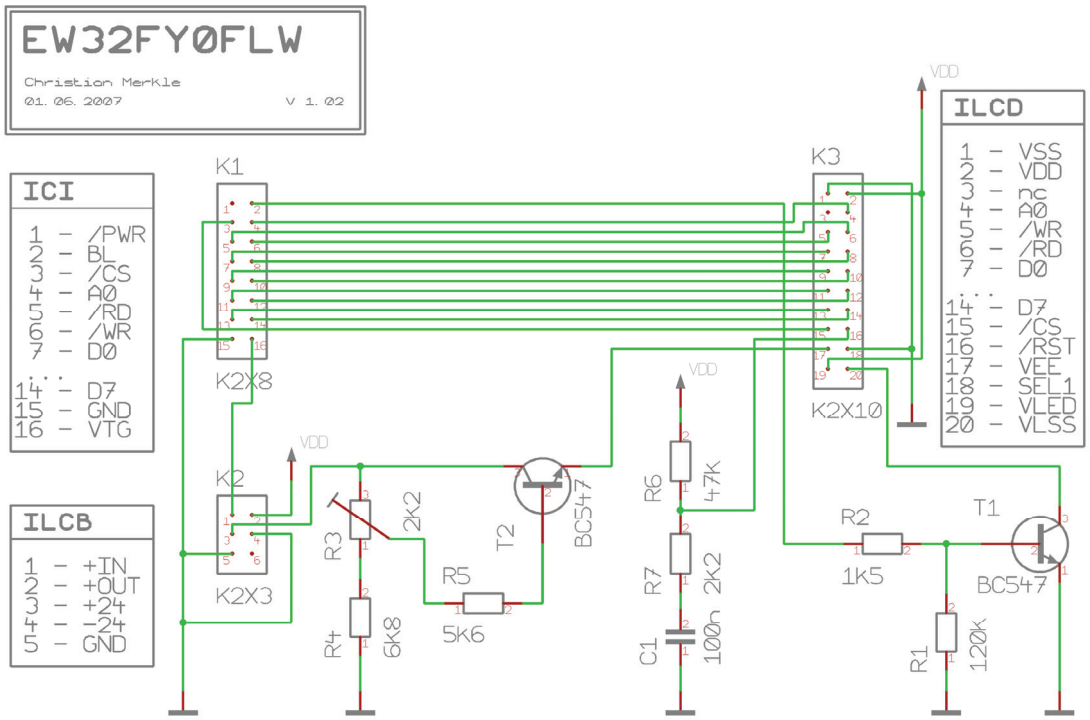


Abbildung 24 – Schaltplan EW32FY0FLW

Das Board besitzt kein *PWR*-Signal. Dadurch ist das Display immer eingeschaltet. Die Hintergrundbeleuchtung wird wie bei den vorherigen Displays gesteuert.

Bei der Kontrastspannung taucht eine Besonderheit auf. Das EW32FY0FLW braucht einen sehr hohen Kontraststrom von bis zu 9mA [EW32]. Dieser Strom ist jedoch zu hoch für den Potentiometer *R3*. *T2* übernimmt diese Aufgabe und verstärkt den Strom der mit *R3* ausgewählten Spannung.

Der Display-Reset wird im Gegensatz zu den anderen Controllern nicht automatisch beim Anlegen der Versorgungsspannung ausgelöst. Zu Beginn leitet *C1* und */RST* liegt über *R7* auf Masse. *C1* lädt sich über *R7* und *R6* auf bis */RST* auf High liegt. Somit ist ein Reset des Controllers sichergestellt.

Das fertig aufgebaute LCD-Board:

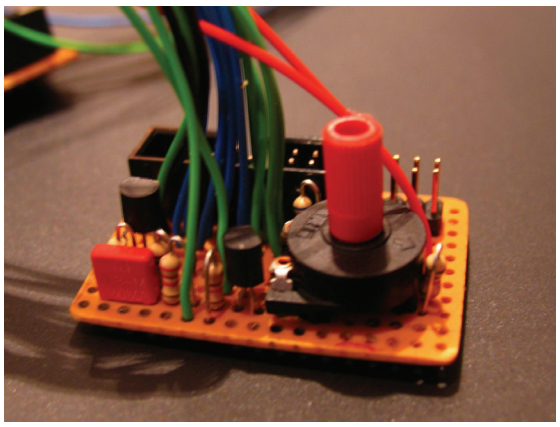


Abbildung 25 – LCD-Board EW32FY0FLW, 1

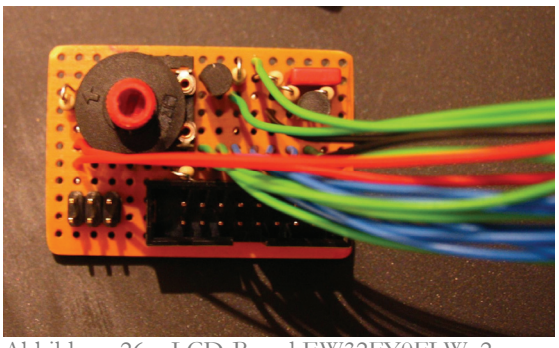


Abbildung 26 – LCD-Board EW32FY0FLW, 2

3.4.4. SHARP M078CKA, LH155

Technische Daten zum Display M078CKA [M078CKA] von Sharp:

Eigenschaft	Wert
Hersteller	Sharp
Bezeichnung	M078CKA
Controller	LH155
Auflösung	240 x 64 [*Pixel*]
Farbtiefe	1 [*bpp*]
Gehäuseabmessungen	89 x 77 [*mm*]
Effektive Anzeigegröße	72 x 32 [*mm*]
Pixelgröße	0.23 [*mm*]
Pixelabstand	0.24 [*mm*]
Versorgungsspannung	5V
Kontrastspannung	14V bis 17V

Tabelle 17 – Technische Daten M078CKA

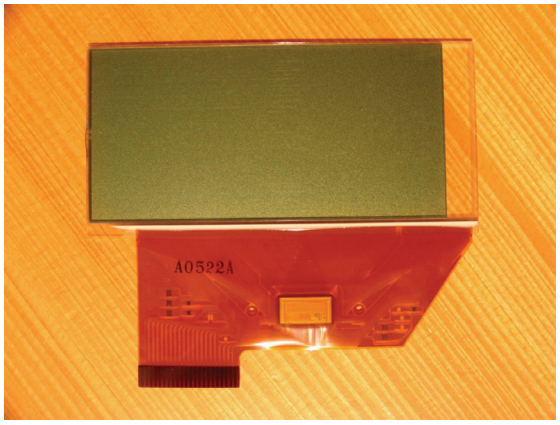


Abbildung 27 – M078CKA, Vorderseite

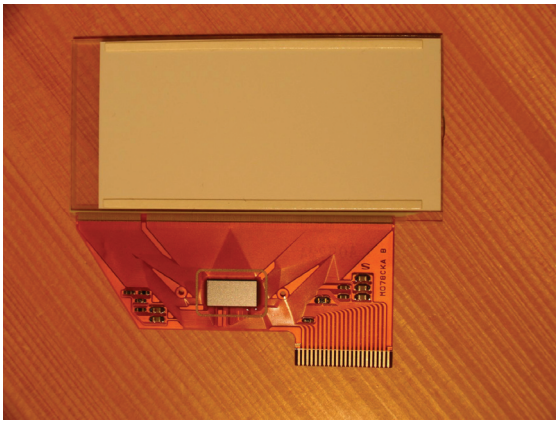


Abbildung 28 – M078CKA, Rückseite

Das M078CKA verwendet den sehr einfachen Controller LH155 von Sharp.

Dieses Display hat einen 22-poligen Anschluß – ebenfalls als Folienleiter:

Pin	Symbol	Beschreibung	Zielsymbol
1	-nc-	*Not Connected*	-/-
2	GND	*Power Supply (GND)*	GND
3	/RESB	*Display Reset*	
4	/CSB	*Chip Select*	/CS
5	RS	*Data / Command Select*	A0
6	M86	*8080 or 6800 Select L:8080*	
7	VDD	*Power Supply (+)*	VTG
8	/WRB	*Data Write*	/WR
9	/RDB	*Data Read*	/RD
10	D0	*Data 0*	D0
…	…	…	…
17	D7	*Data 7*	D7
18	GND	*Power Supply (GND)*	GND
19	VDD	*Power Supply (+)*	VTG
20	Vo	*Power Supply for LCD Driving*	
21	GND	*Power Supply (GND)*	GND
22	-nc-	*Not Connected*	-/-

Tabelle 18 – Pinbelegung M078CKA aus [M078CKA]

Das LCD-Board unterstützt weder das Signal *BL* noch *PWR* und ist wie folgt aufgebaut:

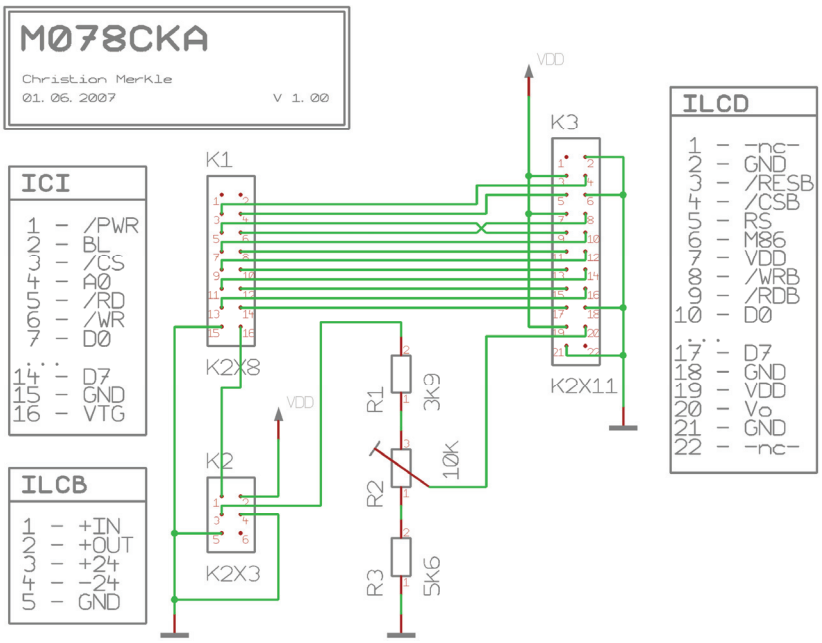


Abbildung 29 – Schaltplan M078CKA

Das Board soll möglichst einfach sein und implementiert daher kein *PWR*-Signal. Dadurch ist das Display immer eingeschaltet. Da es keine Hintergrundbeleuchtung besitzt, ist auch kein *BL*-Signal notwendig.

Die Kontrastspannung wird über den Widerstand *R2* eingestellt.

Das fertig aufgebaute LCD-Board:

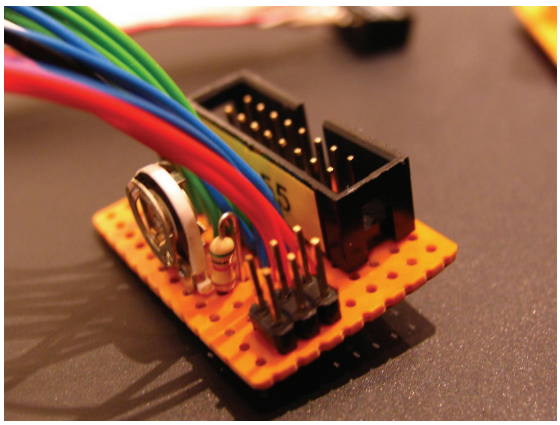


Abbildung 30 – LCD-Board M078CKA

3.4.5. EMBEDDED ARTISTS BOARD, LDS176

Technische Daten zum Display auf dem Board „*LPC2104 Color LCD Game with Bluetooth*“ [EMBART] von Embedded Artists[24]:

Eigenschaft	Wert
Hersteller	Embedded Artists
Bezeichnung	*LPC2104 Color LCD Game with Bluetooth*
Controller	LDS176
Auflösung	130 x 130 [*Pixel*]
Farbtiefe	12 [*bpp*] (native, 8 [*bpp*] and 16 [*bpp*] support)
Gehäuseabmessungen	35 x 46 [*mm*]
Effektive Anzeigegröße	27 x 27 [*mm*]
Pixelgröße	-keine Angaben-
Pixelabstand	-keine Angaben-
Versorgungsspannung	1.65V bis 1.95V, 2.6V bis 2.9V
Kontrastspannung	Max. 18V

Tabelle 19 – Technische Daten Embedded Artists Board

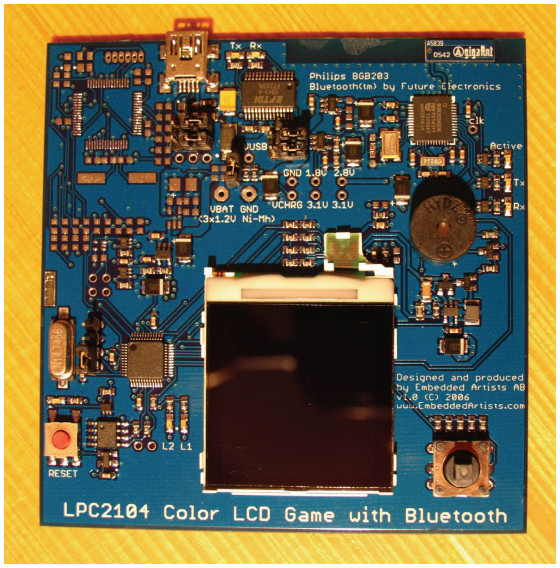


Abbildung 31 – Embedded Artists Board, 1

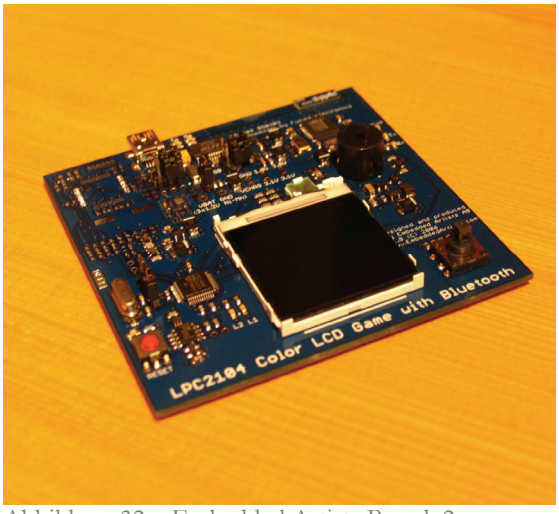


Abbildung 32 – Embedded Artists Board, 2

Dieses Board ist komplett anders als die zuvor beschriebenen, da es schon fertig aufgebaut ist und alle Bauteile auf einer Platine sind. Aber gerade dadurch läßt sich auch der Einsatz in anderen Umgebungen testen.

Wie der Name schon vermuten läßt, arbeitet auf der Platine ein LPC2104-Mikrocontroller der ARM7-Serie von NXP (ehemals Phillips) mit 128kB Flash und 16kB RAM bei knapp 60MHz. Neben dem Display sind einige weitere interessante Baugruppen vorhanden, unter anderem ein Bluetooth-Modul, ein USB-Anschluß mit FT232R-Chip von FTDI, ein Fünf-Tasten-Joystick, ein Piezo-Lautsprecher, ein 8kBit I2C EEPROM und weitere.

Das Display verwendet den komplexen Controller LDS176 von Leadis. Dieser kann in allen drei Bus-Modi verwendet werden (8080, 6800 und SPI), wobei er hier als SPI angeschlossen wurde.

[24] Embedded Artists AB, Friisgatan 33, 214 21 Malmö, Sweden, http://embeddedartists.com

Dieses Display hat damit einen nur 10-poligen Anschluß:

Pin	Symbol	Beschreibung	Zielsymbol
1	VDD18	*Power Supply (+1.8V)*	
2	/RST	*Reset*	
3	DATA	*Data In/Out*	
4	CLK	*Clock*	
5	/CS	*Chip Select*	
6	VDD28	*Power Supply (+2.8V)*	
7	-nc-	*Not Connected*	
8	GND	*Power Supply (GND)*	
9	LED-	*Backlight Cathode*	
10	LED+	*Backlight Anode*	

Tabelle 20 – Pinbelegung Embedded Artists Board

Das Board unterstützt das Signal *BL* (P0.12-BACKLIGHT). Im Folgenden werden nur die LCD-relevanten Teile des Schaltplans betrachtet:

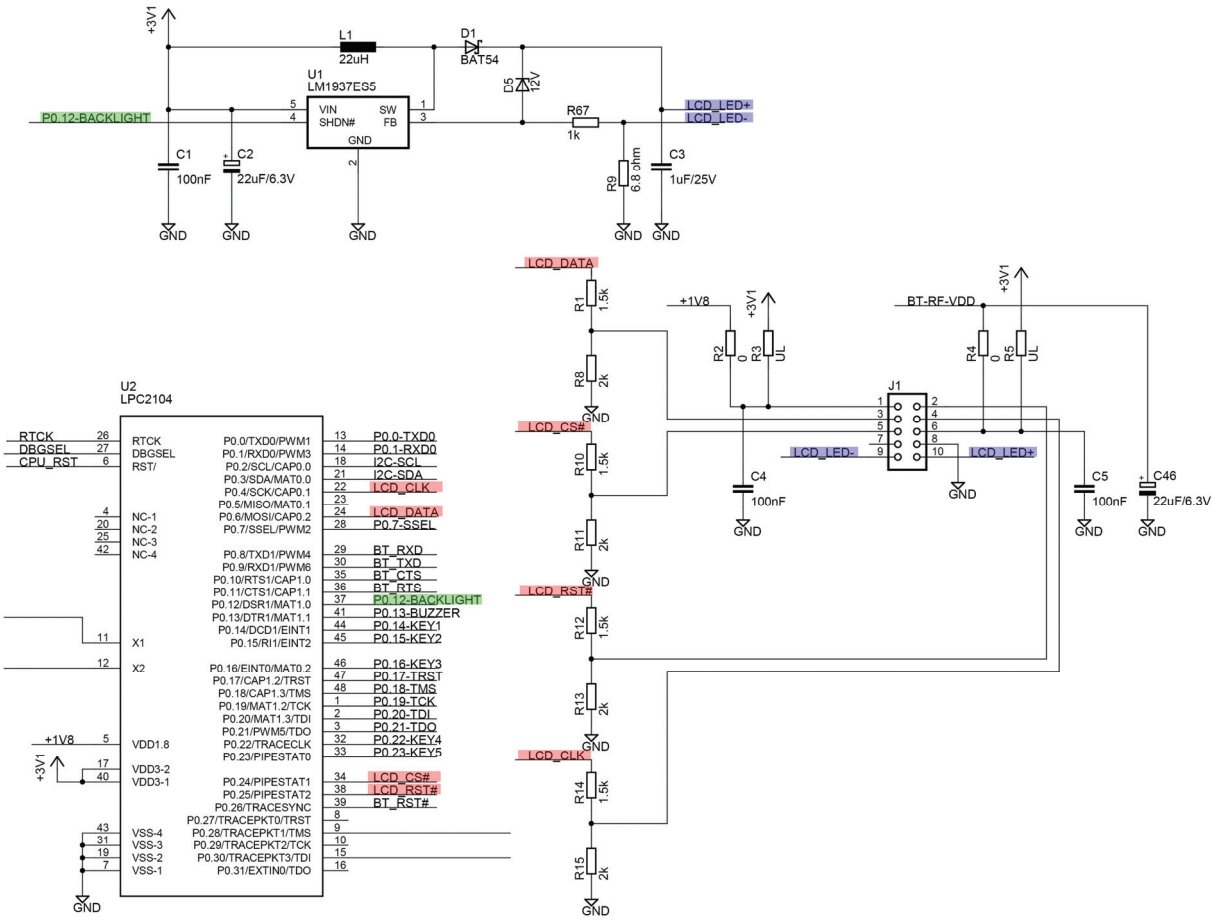


Abbildung 33 – Schaltplan (teilweise, LCD-relevant) Embedded Artists Board, aus [EMBART]

4. SOFTWAREBASIS

Um die einfache Erweiterbarkeit und die reibungslose Integration der Module in das Gesamtprojekt sicherzustellen, ist bei umfangreichen Projekten eine solide Basis notwendig. In diesem Kapitel werden die Haupt-Header-Dateien, die Typdefinitionen, die globalen Hilfsmakros und die plattformabhängigen Funktionen genauer beschrieben. Bis auf letztere befinden sich alle Dateien direkt im Hauptverzeichnis der CMG-Sourcen.

4.1. CMG-HEADER

Es gibt zwei Haupt-Header-Dateien, die sogenannten *Project-Headers*:

- `CMG.h` für Anwenderprogramme:
  Jedes externe Programm, welches CMG-Funktionalität benutzen möchte, muß nur diese Include-Datei einbinden.

  Sie bindet `CMG_Main.h` und `CMG_DRAW.h` für die DRAW-Funktionalität ein. Alle darunterliegenden Schichten werden somit vor dem Benutzer verborgen.

```
#ifndef CMG_H
#define CMG_H

/*
// User cmg include file
// =====================
// This file should be included by the user application to enable cmg support
*/

// main CMG include (will include user configuration)
#include    "CMG_Main.h"

// include DRAW support
#include    "4_DRAW/CMG_DRAW.h"

#endif
```

Listing 4 – CMG.h

- `CMG_Main.h` für **alle** CMG-Quelldateien:
  Jede CMG-Quelldatei muß als allererste Codezeile diese Datei einbinden.

  Diese lädt zuerst die User-Config und anschließend die Typdefinitionen und die globalen Makros. Beide werden in den nächsten Abschnitten näher betrachtet.

```
#ifndef CMG_MAIN_H
#define CMG_MAIN_H

/*
// Main cmg include file
// =====================
// This file must be included _FIRST_ and in _EVERY_ CMG source file
*/

// include user config
#include "../CMG_Config.h"

// include cmg types
#include "CMG_Types.h"

// include common defines
#include "CMG_Commons.h"

// include globals
```

```
#include "CMG_Globals.h"

#endif
```

Listing 5 – CMG_MAIN.h

Jede CMG-Header-Datei umschließt alle Quellcodezeilen mit einem `#ifndef`. Damit wird sichergestellt, daß auch bei mehrmaligem Einbinden dieser Datei keine Fehler auftreten und sie insgesamt nur ein einziges Mal eingebunden wird.

Diese Zeilen werden in zukünftigen Codeauszügen nicht mehr explizit angegeben, sondern werden immer als gegeben angenommen.

Der Bezeichner ergibt sich aus dem Header-Dateinamen. Aus `CMG_MAIN.h` wird also `CMG_MAIN_H`:

```
#ifndef CMG_MAIN_H
#define CMG_MAIN_H

…

#endif
```

Listing 6 – CMG-Header Aufbau

4.2. Typen

Ein großer Nachteil an C ist, daß die eingebauten Typen keine festgelegte Größe besitzen. So kann – je nach Plattform – ein unterschiedlicher Speicheraufbau und damit Programmablauf stattfinden. Auch ist die Kompatibilität untereinander nicht gegeben.

Um plattformunabhängigen Code zu schreiben müssen also alle verwendeten Typen exakt mit Speichergröße und Wertebereich definiert werden. Dieser Teil ist plattformabhängig und über gewisse Schalter der jeweiligen Plattform anzupassen. Für X86-, ARM- und AVR-Systeme sieht die Definition so aus:

```
//------------------------------------------------------------------------
// exact size types (must be exactly the size)
//------------------------------------------------------------------------

// unsigned types
typedef     unsigned char           cmg_u8;
typedef     unsigned short          cmg_u16;
typedef     unsigned long           cmg_u32;

// signed types
typedef     signed char             cmg_s8;
typedef     signed short            cmg_s16;
typedef     signed long             cmg_s32;
//------------------------------------------------------------------------
```

Listing 7 – CMG_Types.h, Teil 1

Hiermit sind nun alle zu verwendenden Typen von 8 bis 32 Bit als vorzeichenlose (u für *unsigned*) und als vorzeichenbehaftete (s für *signed*) definiert.

Die ständige Verwendung des kleinstmöglichen Datentyps ist aber aus Performancegründen nicht immer die beste Wahl. Es sollte deshalb eher eine Abwägung zwischen Performance und Speicherverbrauch getroffen werden. Eine einfache Zählervariable, die nur von Null bis 100 zählt, sollte in X86-Systemen am besten als 32-Bit-Wert definiert werden, da als einzelnes Byte ein großer Performanceverlust die Folge wäre. Der kleine AVR müßte nun aber wesentlich mehr berechnen. Für ihn ist 8-Bit die richtige Entscheidung.

Zur Lösung des Problems existieren in CMG die m-Typen. Das m steht für das Minimum an Bit-Breite. Sie müssen mindesten die angegebene Größe besitzen, können jedoch auch mehr haben.

Bei jeder neuen Variablen sollte also genau darauf geachtet werden, welcher Typ für sie gewählt wird.

Für speicherplatzkritische 8-Bit-Systeme sollte die Definition so aussehen:

```
//-------------------------------------------------------------------------
// minimum size types (must be at least the size but can be larger to
// improove performance)
//-------------------------------------------------------------------------

// unsigned types
typedef     cmg_u8                  cmg_mu8;
typedef     cmg_u16                 cmg_mu16;
typedef     cmg_u32                 cmg_mu32;

// signed types
typedef     cmg_s8                  cmg_ms8;
typedef     cmg_s16                 cmg_ms16;
typedef     cmg_s32                 cmg_ms32;

//-------------------------------------------------------------------------
```

Listing 8 – CMG_Types.h, Teil 2 / 1

Für speicherplatzunkritische 32-Bit-Systeme kann die Definition so aussehen:

```
//-------------------------------------------------------------------------
// minimum size types (must be at least the size but can be larger to
// improove performance)
//-------------------------------------------------------------------------

// unsigned types
typedef     cmg_u32                 cmg_mu8;
typedef     cmg_u32                 cmg_mu16;
typedef     cmg_u32                 cmg_mu32;

// signed types
typedef     cmg_s32                 cmg_ms8;
typedef     cmg_s32                 cmg_ms16;
typedef     cmg_s32                 cmg_ms32;

//-------------------------------------------------------------------------
```

Listing 9 – CMG_Types.h, Teil 2 / 2

Neben diesen Typen gibt es noch eine Reihe von weiteren allgemeinen und CMG-spezifischen:

```
//-------------------------------------------------------------------------
// other types
//-------------------------------------------------------------------------

typedef     void                    cmg_void;

typedef     cmg_u8                  cmg_bool;
#define     true                    1
#define     false                   0

typedef     char                    cmg_char;

typedef     cmg_ms16                cmg_Coord;      // all coordinates
typedef     cmg_ms32                cmg_CoordLong;  // double sized coordinate (i.e. for mul calculations)
typedef     cmg_u16                 cmg_ScrAddr;    // all controller screen addresses (must be u16)
typedef     cmg_u8                  cmg_Color;      // color (must be u8)
//-------------------------------------------------------------------------
```

Listing 10 – CMG_Types.h, Teil 3

Koordinaten sind in CMG generell 16-Bit und vorzeichenbehaftet. Somit lassen sich auch Displays größer als 256 Pixel ansprechen und auch negative Koordinaten sind erlaubt.

Controller-Offset-Werte sind 16-Bit unsigned. Damit lassen sich nahezu alle Displays ansteuern. Sollte es – wider Erwarten – doch nötig sein auf einen größeren Bereich zuzugreifen, so lassen sich auch neue Typen einfach einfügen.

Farben sind in CMG generell 8-Bit breit. Ob sie nun nur 2 oder aber 256 Farben beinhalten ist nebensächlich.

Eine weitere Besonderheit sind Fixed-Point-Zahlen. Diese sind 32-Bit Werte, welche fest aus 16-Bit Ganzzahlen und 16-Bit Nachkommaanteil bestehen. Mit diesen Zahlen kann der Prozessor, auch ohne Gleitkomma-Einheit, sehr schnell rechnen. Auch die Umwandlung zwischen den Formaten geht sehr schnell, indem man nur die ersten beiden Bytes betrachtet, bzw. noch zwei Null-Bytes anhängt. Lediglich bei der Rundung muß zuvor noch ein Wert addiert werden:

```
//------------------------------------------------------------------------
// fixed point types
//------------------------------------------------------------------------
typedef     cmg_s32                 cmg_sfp1616;    // signed fixed point 16.16

#define     SFP_TO_INT(sfp)         ( (sfp) >> 16 ) // return int part of sfp
#define     SFP_ROUNDTO_INT(sfp)    ( ( (sfp) + 0x00007ff8l ) >> 16 ) // return rounded int part of sfp
#define     INT_TO_SFP(val)         ( ( (cmg_sfp1616)(val) ) << 16 )  // return sfp from int
#define     FLOAT_TO_SFP(val)       ( (cmg_sfp1616) ( ( (float)(val) ) * ( (float)(1<<16) ) ) )
//------------------------------------------------------------------------
```

Listing 11 – CMG_Types.h, Teil 4

Weiter besitzt CMG für bestimmte Funktionen Rückgabewerte, die Aufschluß über den Erfolg der Funktion geben. `_OK` zeigt einen fehlerfreien Ablauf an, wobei die restlichen Werte die Ursache beschreiben. Zu beachten ist, daß `_OS`, `_GTK` und `_OS_OR_GTK` den gleichen Wert besitzen.

```
//------------------------------------------------------------------------
// RESULT type
//------------------------------------------------------------------------

typedef     cmg_mu8                 cmg_Result;

#define     CMG_OK                  0x00        // all ok
#define     CMG_ERROR_PARAM         0x01        // parameter error
#define     CMG_ERROR_NOMEM         0x02        // out of memory
#define     CMG_ERROR_OS            0x03        // operating system error (win32)
#define     CMG_ERROR_GTK           0x03        // GTK+ system error (GTK+)
#define     CMG_ERROR_OS_OR_GTK     0x03        // underlaying system error
#define     CMG_ERROR_NOTSUPPORTED  0x04        // ctrl specific fnc not supported
//------------------------------------------------------------------------
```

Listing 12 – CMG_Types.h, Teil 5

Anschließend werden noch ein paar C++-Bezeichner definiert, um den Code etwas lesbarer und verständlicher zu machen. Die Funktionalität ist jedoch eher eingeschränkt, wobei mit `static` die Funktion tatsächlich nur im aktuellen Code sichtbar ist.

```
//------------------------------------------------------------------------
// defines
//------------------------------------------------------------------------

// function modifiers
#define     _PUBLIC
#define     _PROTECTED
#define     _PRIVATE                static

#define     NULL                    0
```

Listing 13 – CMG_Types.h, Teil 6

Zuletzt werden noch ein paar compilerspezifische Definitionen benötigt. Leider sind diese nicht einheitlich und jeder Compiler-Hersteller kocht sein eigenes Süppchen. Noch dazu sind sie – vermutlich aus gutem Grund – schlecht dokumentiert.

Die Rede ist vom Code-Inlining, also dem direkten Ersetzen eines Funktionsaufrufes (Call) in den Zielcode der Funktion. Damit steigt zwar der Speicherverbrauch an, der Performancegewinn jedoch auch überproportional. Kritiker bringen gerne die Komplexität des Ersetzens an und daß der Compiler das sowieso versuchen würde, wenn es etwas bringt. Davon bin ich nicht überzeugt und denke, daß ein fähiger Programmierer sehr wohl besser damit umgehen kann, als ein Compiler.

Die Compiler bringen mit dem Schlüsselwort `inline` eine Möglichkeit hierfür mit. Allerdings ist das nur ein Vorschlag für den Compiler und er kann selbst entscheiden, ob er es lieber sein läßt,

was er meistens auch tut. Es galt also einen Befehl zu finden, der das `inline` erzwingt, auch im Debug-Code. Microsoft bietet hierfür das Schlüsselwort `_forceinline` und der GNU-Compiler das Attribut `__attribute__((always_inline))`:

```
#ifdef WIN32
    // Mircosoft C Compiler
    #define     _INLINE_FORCE           _forceinline
    #define     _INLINE                 _inline
#else
    // GCC Compiler
    #define     _INLINE_FORCE           __attribute__((always_inline))
    #define     _INLINE                 inline
#endif
```

Listing 14 – CMG_Types.h, Teil 7

4.3. Globale Defines und Hilfsmakros

In der Datei `CMG_Globals.h` werden alle globalen Definitionen für CMG vorgenommen, die sich nicht einer bestimmten Ausführungsschicht zuordnen lassen, sondern von mehreren oder sogar allen gemeinsam verwendet werden.

Bis jetzt sind das nur die Power-Statūs:

```
//*****************************************************************************
// power state defines
#define     CMG_PWR_DISPLAY_OFF             0
#define     CMG_PWR_DISPLAY_ON              1
#define     CMG_PWR_BACKLIGHT_OFF           2
#define     CMG_PWR_BACKLIGHT_ON            3
//*****************************************************************************
```

Listing 15 – CMG_Globals.h

Die Datei `CMG_Commons.h` beinhaltet eine Vielzahl universeller Makros, die den Code einfacher und übersichtlicher machen:

Zuerst die Funktionen zum Setzen und Löschen bestimmter Bits in Variablen:

```
// set or clear a bit in val and save the result back to val
#define     SET_BIT(val,bit)            val |= ( 1 << (bit) )
#define     CLR_BIT(val,bit)            val &= ~( 1 << (bit) )

// set or clear a bit in val and return the result
#define     SETBIT(val,bit)             ( val | ( 1 << (bit) ) )
#define     CLRBIT(val,bit)             ( val & ( ~( 1 << (bit) ) ) )
```

Listing 16 – CMG_Commons.h, Teil 1

Mit `SET-` bzw. `GET_BIT(…)` wird das Ergebnis wieder direkt in die Quellvariable zurückgeschrieben – mit dem Paar ohne dem Unterstrich im Namen nur zurückgeliefert und kann in beliebigen Variablen gespeichert werden.

Um einen Wert zu erhalten, in dem nur ein bestimmtes Bit gesetzt ist, eignet sich dieses Makro:

```
// returns a value with the specified bit set
#define     BIT(bit)                    ( 1 << (bit) )
```

Listing 17 – CMG_Commons.h, Teil 2

Um die Anzahl der benötigten ganzen Bytes, die *n* Bits darstellen können, zu berechnen, dient nachfolgende Funktion. Dieses wird zum Beispiel benötigt, um zu berechnen, wie viele Bytes zur Darstellung von *n* Pixel benötigt werden:

```
// necessary amount of bytes to store n bits
#define     GET_BYTES_FROM_BITS(bits)   ( ( ( (bits) - 1 ) >> 3 ) + 1 )
```

Listing 18 – CMG_Commons.h, Teil 3

Das Aufteilen von 16-Bit Werten in zwei 8-Bit Werte übernehmen diese Makros:

```
// get the upper or lower 8 bits form a word
#define     LO_BYTE(x)                  (cmg_u8)( (x) & ( (cmg_u8)0xff ) )
#define     HI_BYTE(x)                  (cmg_u8)( ( (x) >> 8 ) & ( (cmg_u8)0xff ) )

// build a word from two bytes
#define     BUILD_WORD(hi,lo)           ( ( ((cmg_u16)(hi)) << 8 ) | ((cmg_u8)(lo)) )
```

Listing 19 – CMG_Commons.h, Teil 4

Ein paar Größendefinitionen:

```
// varios sizes
#define     KB                          ( 1024 )
#define     MB                          ( 1024 * 1024 )
```

Listing 20 – CMG_Commons.h, Teil 5

Oft benötigte Funktionen sind das Berechnen des Minimums bzw. des Maximums:

```
// min / max / bound
#define     MIN(a,b)                    ( (a)<(b) ? (a) : (b) )
#define     MAX(a,b)                    ( (a)>(b) ? (a) : (b) )
#define     BOUND_MIN_MAX(var,lower,upper)  MIN( MAX( var, lower ), upper )
```

Listing 21 – CMG_Commons.h, Teil 6

`BOUND_MIN_MAX(…)` liefert den Wert der Variable `var` zurück, stellt jedoch sicher, daß dieser im Bereich zwischen `lower` und `upper` ist:

Eine weitere, wichtige Aufgabe ist das Austauschen zweier Variablen. `SWAP(…)` vertauscht `a` und `b`, braucht dazu allerdings aufgabenbedingt eine temporäre Variable. `SWAP_TYPE(…)` dagegen braucht keine konkrete Variable, sondern nur den Typ. Sie legt intern diese Variable selbst an:

```
// swap
#define     SWAP(a,b,temp)              { (temp) = (a);   (a) = (b);   (b) = (temp); }
#define     SWAP_TYPE(a,b,type)         { type typeTemp;    SWAP( (a), (b), typeTemp ); }
```

Listing 22 – CMG_Commons.h, Teil 7

Zu guter Letzt soll die Fehlerbehandlung und Rückgabe in CMG-Funktionen, die einen Statuswert zurückgeben (Typ `cmg_Result`), vereinheitlicht werden:

```
// checks and error handling
#define     CHECK(expr,error)           if ( expr ) { iResult = error; goto _Error; }
#define     CHECKRESULT()               if ( iResult != CMG_OK ) { goto _Error; }
```

Listing 23 – CMG_Commons.h, Teil 8

Ein Grundgerüst einer solchen Funktion sieht wie folgt aus:

```
/*****************************************************************************
| CMG_***
|-----------------------------------
| <function description>
+----------------------------------------------------------------------------
| @Params:          ...
| @Return Value:    result
+*****************************************************************************/
_PUBLIC
cmg_Result CMG_***( void )
{
    cmg_Result  iResult;

    // init CMG_*
    iResult = CMG_***_Init();
    CHECKRESULT();

    // init globals
    g_pSomePtr = malloc( <size> );
    CHECK( g_pSomePtr == NULL, CMG_ERROR_NOMEM );
```

```
    // return success
    return CMG_OK;

    //.............................
    // on error free everything
_Error:
    CMG_DRV_Exit();
    return iResult;
}
```

Listing 24 – Funktion mit Rückgabewert, Grundgerüst

Innerhalb der Funktion muß `iResult` als `cmg_Result` definiert werden. Mit `CHECKRESULT()` wird der Inhalt von `iResult` auf `CMG_OK` geprüft und gegebenenfalls zu `_Error` gesprungen. Mit `CHECK(…)` können beliebige Abfragen stattfinden und die betreffende Fehlermeldung festgelegt werden, die im Fehlerfall zurückgeliefert wird.

4.4. Plattformabhängige Funktionen

Bis jetzt benötigt CMG nur eine plattformabhängige Funktion: Das aktive Warten:

```
    void    CMG_Sleep_ms( cmg_u16 wDelay_ms );
```

Listing 25 – CMG_PlatformDependent.h

Diese Aufgabe wird je nach gewählter Zielplattform anders gelöst:

```
#if defined( CMG_PLATFORM_AVR )

    #include <util/delay.h>

    void CMG_Sleep_ms( cmg_u16 wDelay_ms )
    {
        for ( cmg_u16 wTime_ms = 0; wTime_ms < wDelay_ms; wTime_ms++ )
            _delay_ms( 1.f );
    }


#elif defined( CMG_PLATFORM_ARM )

    void CMG_Sleep_ms( cmg_u16 wDelay_ms )
    {
        volatile long i = 0;
        for ( long x = 0; x < (wDelay_ms * 5450); x++ )
            i++;
    }


#elif defined( CMG_PLATFORM_X86_WINDOWS )

    #include <windows.h>

    void CMG_Sleep_ms( cmg_u16 wDelay_ms )
    {
        Sleep( wDelay_ms );
    }


#elif defined( CMG_PLATFORM_X86_GTK )

    #include <glib.h>

    void CMG_Sleep_ms( cmg_u16 wDelay_ms )
    {
        g_usleep( ( (glong) wDelay_ms ) * 1000 );
    }


#endif
```

Listing 26 – CMG_PlatformDependent.c

Die AVR-Bibliothek bietet direkt eine Funktion zum Warten in Millisekunden an. Diese ist – je nach Taktrate – aber nur bis zu wenigen Millisekunden definiert und erwartet den Übergabeparameter noch dazu im Float-Format. Deshalb wird hier in einer Schleife, auf die Anzahl der Millisekunden – jeweils genau eine – gewartet.

Für die AMR-Plattform wird nur eine feste Anzahl von Wiederholungen ausgeführt. Dieser Punkt ist sicher noch verbesserungswürdig und muß der Taktrate angepaßt werden.

Windows bietet dafür eine WinApi-Funktion namens `Sleep(…)` an und GTK eine mit dem Namen `g_usleep(…)`.

4.5. Make

Um nicht in jeder Make-Datei alle Quellcode-Dateien einzeln aufzuzählen, dient die Datei `MakeInclude`. In ihr sind alle verwendeten Sourcen nach dem folgenden Schema definiert:

```
###############################################################################
# CMG files
###############################################################################

# IMPORTANT:
# ----------
# $(CMG_ROOT) must be defined before!

# LLIO_*:
CMG_LLIO_DIR = $(CMG_ROOT)1_LLIO/
CMG_LLIO  = $(CMG_LLIO_DIR)CMG_LLIO_ARM_8D4CTL_8080.c
CMG_LLIO += $(CMG_LLIO_DIR)CMG_LLIO_AVR_8D4CTL_8080.c
CMG_LLIO += $(CMG_LLIO_DIR)CMG_LLIO_AVR_8D4CTL_6800.c
CMG_LLIO += $(CMG_LLIO_DIR)CMG_LLIO_X86_PARALLEL_8D4CTL_8080.c
CMG_LLIO += $(CMG_LLIO_DIR)EMU/CMG_LLIO_EMU_DISPLAY_GDI.c
CMG_LLIO += $(CMG_LLIO_DIR)EMU/CMG_LLIO_EMU_DISPLAY_GTK.c
CMG_LLIO += $(CMG_LLIO_DIR)EMU/CMG_LLIO_EMU_EMULATOR_T6963.c
CMG_LLIO += $(CMG_LLIO_DIR)EMU/CMG_LLIO_EMU_DISPLAY_COMMON.c

# CTRL:
CMG_CTRL_DIR = $(CMG_ROOT)2_CTRL/
CMG_CTRL  = $(CMG_CTRL_DIR)CMG_CTRL_T6963.c
CMG_CTRL += $(CMG_CTRL_DIR)CMG_CTRL_LH155.c
CMG_CTRL += $(CMG_CTRL_DIR)CMG_CTRL_S1D13700.c

# DRV:
CMG_DRV_DIR = $(CMG_ROOT)3_DRV/
CMG_DRV = $(CMG_DRV_DIR)CMG_DRV_UNI_1BPP.c

# DRAW:
CMG_DRAW_DIR = $(CMG_ROOT)4_DRAW/
CMG_DRAW  = $(CMG_DRAW_DIR)CMG_DRAW_Ellipse.c
CMG_DRAW += $(CMG_DRAW_DIR)CMG_DRAW_GetScreen.c
CMG_DRAW += $(CMG_DRAW_DIR)CMG_DRAW_Image.c
CMG_DRAW += $(CMG_DRAW_DIR)CMG_DRAW_Line.c
CMG_DRAW += $(CMG_DRAW_DIR)CMG_DRAW_LineList.c
CMG_DRAW += $(CMG_DRAW_DIR)CMG_DRAW_LineWidth.c
CMG_DRAW += $(CMG_DRAW_DIR)CMG_DRAW_Main.c
CMG_DRAW += $(CMG_DRAW_DIR)CMG_DRAW_MainDrawHelpers.c
CMG_DRAW += $(CMG_DRAW_DIR)CMG_DRAW_MainDrawstyleColorPenBrush.c
CMG_DRAW += $(CMG_DRAW_DIR)CMG_DRAW_Pixel.c
CMG_DRAW += $(CMG_DRAW_DIR)CMG_DRAW_Rectangle.c
CMG_DRAW += $(CMG_DRAW_DIR)CMG_DRAW_RoundedRectangle.c
CMG_DRAW += $(CMG_DRAW_DIR)CMG_DRAW_Scroll.c
CMG_DRAW += $(CMG_DRAW_DIR)CMG_DRAW_Triangle.c

[…]

# SOURCES:
CMG_SOURCES = $(CMG_LLIO) $(CMG_CTRL) $(CMG_DRV) $(CMG_DRAW) $(CMG_TEXT) $(CMG_MISC)
```

Listing 27 – MakeInclude, Ausschnitt

In der eigentlichen Make-Datei muß zuvor `CMG_ROOT` definiert werden. Eine Beispieleinbindung sieht so aus:

```
[…]
# CMG files
CMG_ROOT = ../../CMG/
-include $(CMG_ROOT)MakeInclude

# List C source files here. (C dependencies are automatically generated.)
SRC = ../../$(TARGET).c $(CMG_SOURCES)
[…]
```

Listing 28 – Beispiel-Make für CMG-Sourcen-Einbindung

4.6. AUFBAU DER MODUL-SCHICHTEN

Zur Erinnerung noch einmal die Module von CMG:

Direkter Modus:

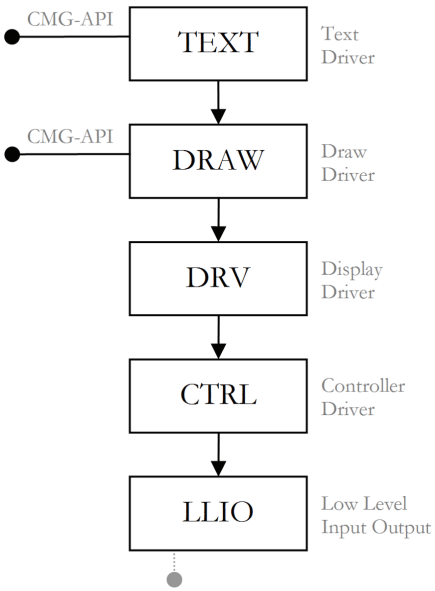


Abbildung 34 – CMG Aufbau 1

GUI-Modus:

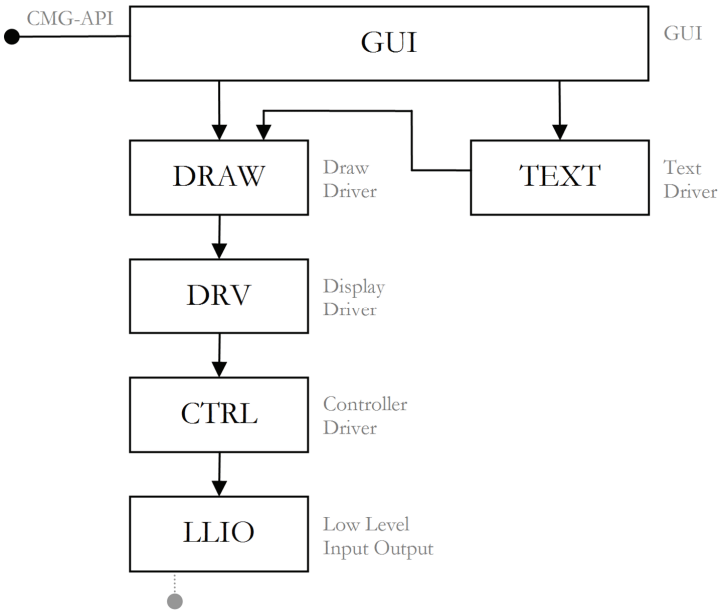


Abbildung 35 – ARM Board, Rückseite

Die dazugehörenden Aufgaben:

Abkürzung	Bedeutung	Erklärung
LLIO	**Low Level Input Output**	Direkte Kommunikation mit dem Display über physikalische Anschlüsse, dem Bus. Ergebnis ist eine plattform- und busunahängige Kommunikation mit dem Displaycontroller.
CTRL	**Controller**	Unterschiedliche Ansteuerungen der Displaycontroller und Implementierung des standardisierten Zugriffsinterfaces von CMG. Ergebnis ist ein controllerunabhängiger Zugriff auf das Display.
DRV	**Driver**	Die untere Schicht des Grafiktreibers mit grundlegenden Grafikfunktionen und Stift-Unterstützung.
DRAW	**Drawing**	Alle Zeichenoperationen – diese Schicht ist die direkte Benutzerschnittstelle für Anwendungsprogramme. Hier findet die Trennung zwischen physikalischer Displayauflösung und Orientierung von der eigentlichen Anzeige statt.
TEXT	**Text Rendering**	Alle Funktionen zum Umgang mit Text: Schriftarten laden, Texte vermessen und bearbeiten sowie die eigentliche Ausgabe dieser Texte.
GUI	**User Interface**	Grafisches Benutzerinterface: Aufteilung des Displays in einzelne, voneinander unabhängige Fenster mit ihren eigenen Nachrichtensystemen.

Tabelle 21 – Beschreibungen der einzelnen Module von CMG

5. LLIO – Low Level IO

Die unterste aller Schichten ist für das Ansprechen des Displays zuständig. Dabei werden gleich zwei der Anforderungen berücksichtigt:

- Plattform
- Bus-Protokoll

Ein LLIO-Treiber stellt somit eine Bus-Protokoll- und plattformabhängige Schnittstelle zum Displaycontroller her. Mit dieser Schicht kann man byteweise auf den Displaycontroller zugreifen.

Ganz läßt es sich nicht vermeiden, aber um möglichst wenig plattformabhängigen Code zu erzeugen, muß der LLIO-Code so kompakt wie möglich sein.

Auf der LLIO-Ebene (die Schnittstelle der LLIO-Schicht) kann man anschließend plattformunabhängig auf beliebige Displays zugreifen, ohne sich weitere Gedanken über plattform-, zugriffs- oder systemspezifische Teile zu machen. Auch das korrekte Zugriffstiming übernimmt diese Schicht, allerdings nur auf Busebene. Controllerspezifische Timings müssen von höheren Schichten beachtet werden.

Die LLIO-Ebene besitzt allerdings stets einen physikalischen Ausgang, an welchem ein Display angeschlossen wird. Die im nächsten Kapitel vorgestellten Emulatoren haben zwar die gleiche Schnittstelle wie die LLIO-Ebene, intern aber einen komplett anderen Aufbau.

5.1. Aufbau

Die Quell- und Config-Dateien befinden sich im Ordner `CMG/1_LLIO`.

Für jede zu unterstützende Plattform- und Bus-Protokoll-Kombination muß eine Quelldatei, inklusive einer dazugehörigen Config-Datei, existieren.

Unterstützte Plattformen sind bis jetzt:

- AVR
- ARM
- X86 Parallelport (teilweise)

Diese Liste ist beliebig erweiterbar unter der Voraussetzung, daß ANSI-C unterstützt wird und Zugriff auf externe Schnittstellen besteht – in welcher Form auch immer. Ob nun, wie bei AVR und ARM, direkt über IO-Pins oder über parallele Schnittstellen, Erweiterungskarten oder USB-Module, ist nebensächlich.

Die unterstützten Bus-Protokelle sind:

- 8080 (Intel), (/CS, /A0, /WR, /RD)
- 6800 (Motorola) , (/CS, /A0, R/W, /E)
- SPI (Serial Peripheral Interface), (MOSI, MISO, SCK)

Dieses sind die gängigsten Protokolle für Displays. Die 8080- und 6800-Protokolle werden in Software emuliert, könnten aber mit der passenden Hardware auch direkt in den Prozessorbus eingebunden werden. Dazu wären dann allerdings angepaßte LLIO-Treiber nötig.

Prinzipiell sind für einen LLIO-Treiber alle Kombinationen möglich, wobei manche Kombinationen weniger sinnvoll erscheinen.

Bis jetzt sind folgende Kombinationen verfügbar:

- `CMG_LLIO_ARM_8D4CTL_8080.c`
- `CMG_LLIO_AVR_8D4CTL_6800.c`
- `CMG_LLIO_AVR_8D4CTL_8080.c`
- `CMG_LLIO_X86_PARALLEL_8D4CTL_8080.c`

Der Aufbau des Dateinamens ist einfach: Nach der Schicht wird die Plattform angegeben (ARM, AVR oder X86), anschließend das Ausgabemedium (fehlt es, sind die IO-Pins gemeint) und zuletzt das Busprotokoll. 8D4CTL steht hierbei für acht Datenleitungen und vier Steuerleitungen. Das zusätzliche PARALLEL bei X86 bedeutet, daß die parallele Schnittstelle als Ausgabemedium verwendet wird.

Ein Treiber für z.B. einen ARM-Prozessor mit SPI-Ausgabe würde `CMG_LLIO_ARM_SPI.c` heißen, für einen FTDI-Chip – an einem USB-Port des PCs angeschlossen – `CMG_LLIO_X86_USB_FTDI.c`.

Hier noch einmal eine grafische Übersicht:

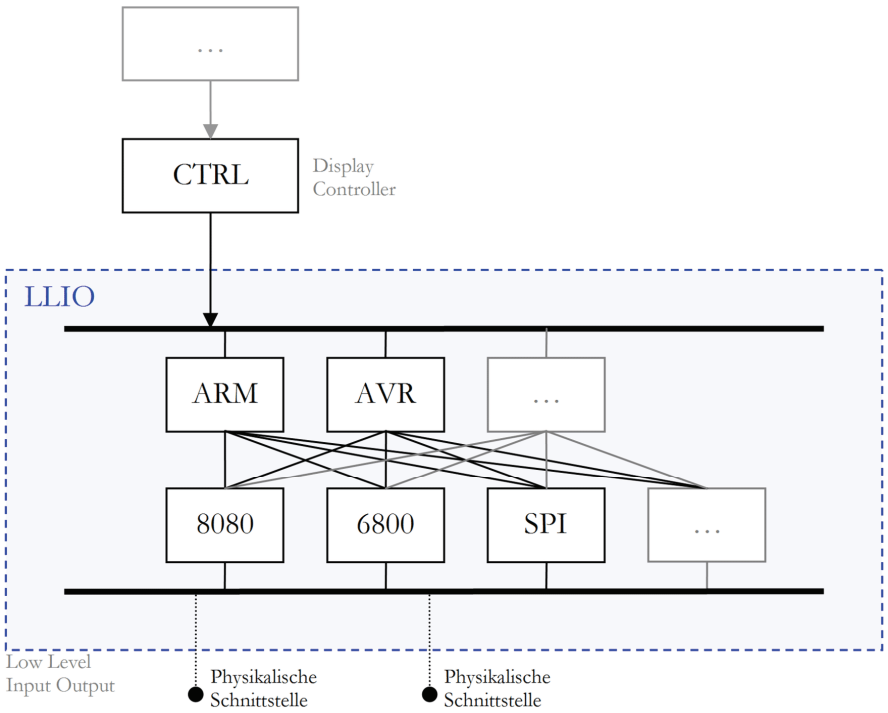


Abbildung 36 – CTRL-Aufbau

5.2. SCHNITTSTELLEN

Zu Beginn stellt sich die Frage wie man möglichst viel von der Ansteuerung des Displays abstrahieren kann, um somit einen kleinen, plattformabhängigen Code zu bekommen. Dabei kommt es auch darauf an, was alle Controller gemeinsam haben.

Alle mir bekannten, kleinen Controller arbeiten auf der untersten Eben mit einem bidirektionalen, 8-Bit breiten Datenregister und den dazugehörenden Steuerleitungen. Zusätzlich bieten sie die Auswahl zwischen zwei Adressen.

Die Adreßaufteilung ist meist wie folgt: Beim Schreiben gibt es ein Befehls- und Datenregister; beim Lesen ein Daten- und Statusregister.

Die eigentliche Aufgabe der LLIO-Schnittstelle ist der Zugriff auf eben genau diese zwei Adressen, bzw. vier Register. Da die Reihenfolge der Register nicht einheitlich ist und teilweise sogar das Statusregister gar nicht verwendet wird, bezeichnet CMG in LLIO die Register nur mit der Adresse 0 oder 1.

Wie schon in den Grundlangen angesprochen, besitzt jede Schicht – zusätzlich zu ihren speziellen – die gemeinsamen Schnittstellen mit `Init()`, `Exit()` und `PowerManagement(…)`.

Die Schnittstellen der LLIO-Schicht sehen nun so aus:

```
    //-- Init/Exit ---------------------------------------------------------
        cmg_Result  CMG_LLIO_Init( void );
        void        CMG_LLIO_Exit( void );
        void        CMG_LLIO_PowerManagement( cmg_mu8 uState );
    //----------------------------------------------------------------------

    //-- Write -------------------------------------------------------------
        void        CMG_LLIO_WriteA0( cmg_u8 byData );
        void        CMG_LLIO_WriteA1( cmg_u8 byData );
    //----------------------------------------------------------------------

    //-- Read --------------------------------------------------------------
        cmg_u8      CMG_LLIO_ReadA0( void );
        cmg_u8      CMG_LLIO_ReadA1( void );
    //----------------------------------------------------------------------
```

Listing 29 – LLIO.h

`WriteA0(…)`, `WriteA1(…)`, `ReadA0()` und `ReadA1()` sind die zuvor besprochenen Funktionen zum byteweisen Zugriff auf den Controller.

5.3. KONFIGURATION

Die LLIO-Schicht besitzt noch eine weiter Besonderheit: Da jede Plattform und jedes Protokoll unterschiedlichste Einstellungen nötig macht, gibt es keine gemeinsame Config-Datei für die gesamte Schicht, sondern jeder Typ erhält seine eigene.

In ihr werden, z.B. beim AVR, die Ports – mit den dazugehörigen Pins (Bits) – den einzelnen Signalen zugeordnet. Ein Beispielausschnitt aus `CMG_LLIO_AVR_8D4CTL_8080.config` sieht so aus:

```
// Data-Port:
// ----------
// All 8 data pins of the display must be connected to an 8 bit port.
#define     LLIO_DATA_PORT                  PORTA
#define     LLIO_DATA_PIN                   PINA
#define     LLIO_DATA_PORT_DDR              DDRA
//
//
//
// Control-Pins:
// -------------
// Each control signal can be freely defined.
//
// * /CS:
#define     LLIO_CTRL_iCS_PORT              PORTC
#define     LLIO_CTRL_iCS_PORT_DDR          DDRC
#define     LLIO_CTRL_iCS_BIT               4
//
// * A0:
#define     LLIO_CTRL_A0_PORT               PORTC
#define     LLIO_CTRL_A0_PORT_DDR           DDRC
#define     LLIO_CTRL_A0_BIT                5
//
// * /RD:
#define     LLIO_CTRL_iRD_PORT              PORTC
#define     LLIO_CTRL_iRD_PORT_DDR          DDRC
#define     LLIO_CTRL_iRD_BIT               6
//
// * /WR:
#define     LLIO_CTRL_iWR_PORT              PORTC
#define     LLIO_CTRL_iWR_PORT_DDR          DDRC
#define     LLIO_CTRL_iWR_BIT               7
```

Listing 30 – CMG_LLIO_AVR_8D4CTL_8080.config, Teil 1

Hier wurden der Datenport und die vier Steuerleitungen definiert.

Ein AVR legt im `DDR*`-Register die Richtung fest, die Ausgabe findet mit `PORT*` und das Einlesen mit `PIN*` statt. Im Gegensatz zu ARM: Hier können Leitungen nur mit einem `SET`- und einem `CLR`-Register gesetzt bzw. gelöscht werden. Auch gibt es keine abgeschlossenen 8-Bit Ports, sondern 32-Bit Ports, in welchen man das Startbit markiert:

```
// Data-Port:
// ----------
// All 8 data pins of the display must be connected to an 8 bit port.
#define     LLIO_DATA_PORT_SET              IO0SET
#define     LLIO_DATA_PORT_CLR              IO0CLR
#define     LLIO_DATA_PORT_DIR              IO0DIR
#define     LLIO_DATA_PORT_PIN              IO0PIN
#define     LLIO_DATA_STARTBIT              8
//
//
//
// Control-Pins:
// -------------
// Each control signal can be freely defined.
//
// * /CS:
#define     LLIO_CTRL_iCS_PORT_SET          IO0SET
#define     LLIO_CTRL_iCS_PORT_CLR          IO0CLR
#define     LLIO_CTRL_iCS_PORT_DIR          IO0DIR
#define     LLIO_CTRL_iCS_BIT               16
//
// * A0:
#define     LLIO_CTRL_A0_PORT_SET           IO0SET
#define     LLIO_CTRL_A0_PORT_CLR           IO0CLR
#define     LLIO_CTRL_A0_PORT_DIR           IO0DIR
#define     LLIO_CTRL_A0_BIT                17

[…]
```

Listing 31 – CMG_LLIO_ARM_8D4CTL_8080.config, Teil 1

Weiter unten in den Konfigurationsdateien werden die PWR- und die BL-Leitung zum Schalten des Displays und der Hintergrundbeleuchtung konfiguriert:

```
// Display Power Supply Control:
// -----------------------------
// If the display is not always connected to power and can be switched
// from an IO-pin define this:
#define     LLIO_DISPLAYPOWER_ENABLE
//
// If display power is enabled configure the pin:
#define     LLIO_DISPLAYPOWER_PORT          PORTC
#define     LLIO_DISPLAYPOWER_PORT_DDR      DDRC
#define     LLIO_DISPLAYPOWER_BIT           2
//
// If the switching logic is negative on (inverted) define this:
#define     LLIO_DISPLAYPOWER_INVERTED
//
//
//
// Backlight Control:
// ------------------
// If a backlight is available and this backlight can be switched from an
// IO-pin define this:
#define     LLIO_BACKLIGHT_ENABLE
//
// If backlight is enabled configure the pin:
#define     LLIO_BACKLIGHT_PORT             PORTC
#define     LLIO_BACKLIGHT_PORT_DDR         DDRC
#define     LLIO_BACKLIGHT_BIT              3
//
// If the switching logic is negative on (inverted) define this:
//#define     LLIO_BACKLIGHT_INVERTED
//
// If the backlight should be initially on define this:
#define     LLIO_BACKLIGHT_INITIALLY_ON
```

Listing 32 – CMG_LLIO_AVR_8D4CTL_8080.config, Teil 2

Für beide Signale muß festgelegt werden, ob CMG diese verwendet (`LLIO_***_ENABLE`), an welchem Pin des Controllers diese anliegen (controllerspezifisch) und ob das Signal invertiert ausgegeben werden soll oder nicht (`LLIO_***_INVERTED`). Zusätzlich wird der Anfangszustand der Hintergrundbeleuchtung mit (`LLIO_BACKLIGHT_INITIALLY_ON`) festgelegt.

Zuletzt müssen noch zwei timing-relevante Einstellungen festgelegt werden:

```
// Timing: Wait Cycles:
// ----------------------------
// Define here the amount of additional waitcycles (nops) when /WR or /RD
// getting low (active) until they get high again. Look for informations
// in the controller's datasheet:
#define     LLIO_WR_WAITCYCLES              0
#define     LLIO_RD_WAITCYCLES              5
```

Listing 33 – CMG_LLIO_AVR_8D4CTL_8080.config, Teil 2

`LLIO_WR_WAITCYCLES` und `LLIO_RD_WAITCYCLES` definieren die Anzahl der Taktzyklen, die der Prozessor nach dem Aktivsetzen von /WR bzw. /RD warten muß, bis er das Signal wieder deaktiviert. Die genaue, minimale Zeit ist in den Datenblättern des jeweiligen Display-Controllers definiert.

Je nach Prozessor, Taktrate und Displaycontroller muß nun ein passender Wert berechnet werden. Ist dieser zu hoch, wird die Ausgabe nur verlangsamt – es treten keine weiteren negativen Eigenschaften auf. Ist er jedoch zu niedrig, können Datenübertragungsfehler auftreten und die Ansteuerung des Displays fällt teilweise aus oder ist sogar überhaupt nicht mehr möglich. Es sollte demnach eher ein etwas zu großer Wert gewählt werden.

5.4. GEMEINSAME KOMPONENTEN

Die eben definierte Anzahl der Waitcycles muß nun noch in einem Programmcode umgesetzt werden. Dafür muß ein Weg gefunden werden, um den Prozessor an einer beliebigen Stelle für *n* Taktzyklen anzuhalten bzw. anderweitig zu beschäftigen. Genau dies ist die Aufgabe der Include-Datei `CMG_LLIO_WaitCycles.h`:

```
#if !defined( LLIO_WR_WAITCYCLES ) || !defined( LLIO_RD_WAITCYCLES )
    #error LLIO_*_WAITCYCLES are not defined. Please customize the config file correctly using the *.config
                            template.
#endif


_PRIVATE
_INLINE_FORCE                           // MUST be inline
void LLIO_Write_WaitCycles( void )
{
    #if LLIO_WR_WAITCYCLES >= 1
        asm volatile ("nop");
    #endif
    #if LLIO_WR_WAITCYCLES >= 2
        asm volatile ("nop");
    #endif
    #if LLIO_WR_WAITCYCLES >= 3
        asm volatile ("nop");
    #endif
    #if LLIO_WR_WAITCYCLES >= 4
        asm volatile ("nop");
    #endif

[...]

    #if LLIO_WR_WAITCYCLES >= 31
        asm volatile ("nop");
    #endif
    #if LLIO_WR_WAITCYCLES >= 32
        asm volatile ("nop");
    #endif
    #if LLIO_WR_WAITCYCLES >= 33
        #error Too many read waitcycles. Reduce number or adept this file.
    #endif
}


_PRIVATE
_INLINE_FORCE                           // MUST be inline
void LLIO_Read_WaitCycles( void )
{
    #if LLIO_RD_WAITCYCLES >= 1
        asm volatile ("nop");
    #endif
    #if LLIO_RD_WAITCYCLES >= 2
        asm volatile ("nop");
    #endif

[...]

    #if LLIO_RD_WAITCYCLES >= 32
        asm volatile ("nop");
    #endif
    #if LLIO_RD_WAITCYCLES >= 33
        #error Too many read waitcycles. Reduce number or adept this file.
    #endif
}
```

Listing 34 – CMG_LLIO_WaitCycles.h (gekürzt)

Zunächst wird geprüft, ob die benötigten Symbole `LLIO_WR_WAITCYCLES` und `LLIO_RD_WAITCYCLES` definiert sind und andernfalls ein Fehler ausgegeben.

Die beiden Funktionen `LLIO_Write_WaitCycles()` und `LLIO_Read_WaitCycles()` sind genau gleich aufgebaut, nur greifen sie jeweils auf ihre passende Anzahl zu.

Beide Funktionen müssen zwingend Inline sein (mit `_INLINE_FORCE`), damit nicht noch zusätzlich der Overhead des Funktionsaufrufes dazukommt.

Die eigentliche Warteaufgabe für den Prozessor erledigt der Assembler-Befehl `nop`, welcher einfach nur für die Anzahl der Wartetakte hintereinander gereiht wird.

5.5. AVR

Zuerst soll der controller- und hardwareabhängige Teil des AVRs besprochen werden. Dieser ist für die beiden Protokolle 8080 und 6800 implementiert.

5.5.1. 8080-Bus

Zu Beginn werden Hilfsmakros für Steuerung definiert, um später den Code leichter lesbar zu machen und Fehler zu vermeiden:

```
// macros for ctrl interaction
#define     SET_CTRL_CS_SELECT          CLR_BIT( LLIO_CTRL_iCS_PORT, LLIO_CTRL_iCS_BIT )
#define     SET_CTRL_CS_DESELECT        SET_BIT( LLIO_CTRL_iCS_PORT, LLIO_CTRL_iCS_BIT )

#define     SET_CTRL_A0_0               CLR_BIT( LLIO_CTRL_A0_PORT, LLIO_CTRL_A0_BIT )
#define     SET_CTRL_A0_1               SET_BIT( LLIO_CTRL_A0_PORT, LLIO_CTRL_A0_BIT )

#define     SET_CTRL_RD_READ            CLR_BIT( LLIO_CTRL_iRD_PORT, LLIO_CTRL_iRD_BIT )
#define     SET_CTRL_RD_CLEAR           SET_BIT( LLIO_CTRL_iRD_PORT, LLIO_CTRL_iRD_BIT )

#define     SET_CTRL_WR_WRITE           CLR_BIT( LLIO_CTRL_iWR_PORT, LLIO_CTRL_iWR_BIT )
#define     SET_CTRL_WR_CLEAR           SET_BIT( LLIO_CTRL_iWR_PORT, LLIO_CTRL_iWR_BIT )


// macros for data interaction
#define     SET_DATA_PORT_DIRECTION_IN  LLIO_DATA_PORT_DDR = 0x00
#define     SET_DATA_PORT_DIRECTION_OUT LLIO_DATA_PORT_DDR = 0xff

#define     SET_DATA_PORT(data)         LLIO_DATA_PORT = ( data )
#define     GET_DATA_PORT()             LLIO_DATA_PIN


// macros for power interaction
// display power
#ifdef LLIO_DISPLAYPOWER_ENABLE
    #ifndef LLIO_DISPLAYPOWER_INVERTED
        // standard power logic
        #define     SET_DISPLAYPOWER_ON         SET_BIT( LLIO_DISPLAYPOWER_PORT, LLIO_DISPLAYPOWER_BIT )
        #define     SET_DISPLAYPOWER_OFF        CLR_BIT( LLIO_DISPLAYPOWER_PORT, LLIO_DISPLAYPOWER_BIT )
    #else
        // inverted power logic
        #define     SET_DISPLAYPOWER_ON         CLR_BIT( LLIO_DISPLAYPOWER_PORT, LLIO_DISPLAYPOWER_BIT )
        #define     SET_DISPLAYPOWER_OFF        SET_BIT( LLIO_DISPLAYPOWER_PORT, LLIO_DISPLAYPOWER_BIT )
    #endif
#endif

// backlight
#ifdef LLIO_BACKLIGHT_ENABLE
    #ifndef LLIO_BACKLIGHT_INVERTED
        // standard backlight logic
        #define     SET_BACKLIGHT_ON            SET_BIT( LLIO_BACKLIGHT_PORT, LLIO_BACKLIGHT_BIT )
        #define     SET_BACKLIGHT_OFF           CLR_BIT( LLIO_BACKLIGHT_PORT, LLIO_BACKLIGHT_BIT )
    #else
        // inverted backlight logic
        #define     SET_BACKLIGHT_ON            CLR_BIT( LLIO_BACKLIGHT_PORT, LLIO_BACKLIGHT_BIT )
        #define     SET_BACKLIGHT_OFF           SET_BIT( LLIO_BACKLIGHT_PORT, LLIO_BACKLIGHT_BIT )
    #endif
#endif
```

Listing 35 – CMG_LLIO_AVR_8D4CTL_8080.c, Teil 1

Es existieren zwei Ziele: Das Control- (CTRL) und das Data-Register (DATA). Beim Control-Register kann jedes der vier Signale einzeln gesetzt werden; das Data-Register jedoch nur gesamt gesetzt oder eingelesen werden. Zuvor muß noch die entsprechende Richtung festgelegt werden.

Der restliche Teil schaltet das gesamte Display bzw. die Hintergrundbeleuchtung ein und aus – jedoch nur, falls dieses Feature verwendet werden soll.

Beim Initialisieren der LLIO-Schicht geschieht folgendes:

```
/*****************************************************************************
| CMG_LLIO_Init
|-----------------------------------
| Init  (init downwards: nothing)
+-----------------------------------------------------------------------------
| @Params:          none
| @Return Value:    result
+*****************************************************************************/
_PUBLIC
cmg_Result CMG_LLIO_Init( void )
{
    // switch on display power if available
    #ifdef LLIO_DISPLAYPOWER_ENABLE
        // switch on
        SET_DISPLAYPOWER_ON;

        // set ctrl pins direction out
        SET_BIT( LLIO_DISPLAYPOWER_PORT_DDR, LLIO_DISPLAYPOWER_BIT );
    #endif

    // set data port std direction (all in) and std value
    SET_DATA_PORT( 0x00 );
    SET_DATA_PORT_DIRECTION_IN;

    // set ctrl pins std value
    SET_CTRL_CS_DESELECT;
    SET_CTRL_A0_0;
    SET_CTRL_RD_CLEAR;
    SET_CTRL_WR_CLEAR;

    // set ctrl pins direction (all out)
    SET_BIT( LLIO_CTRL_iCS_PORT_DDR,    LLIO_CTRL_iCS_BIT );
    SET_BIT( LLIO_CTRL_A0_PORT_DDR,     LLIO_CTRL_A0_BIT );
    SET_BIT( LLIO_CTRL_iRD_PORT_DDR,    LLIO_CTRL_iRD_BIT );
    SET_BIT( LLIO_CTRL_iWR_PORT_DDR,    LLIO_CTRL_iWR_BIT );

    // wait until display is ready
    CMG_Sleep_ms( LLIO_DISPLAY_INIT_WAITTIME_MS );

    // select chip
    SET_CTRL_CS_SELECT;

    // enable backlight if available
    #ifdef LLIO_BACKLIGHT_ENABLE
        // set initial power state
        #ifdef LLIO_BACKLIGHT_INITIALLY_ON
            SET_BACKLIGHT_ON;
        #else
            SET_BACKLIGHT_OFF;
        #endif

        // set ctrl pins direction out
        SET_BIT( LLIO_BACKLIGHT_PORT_DDR, LLIO_BACKLIGHT_BIT );
    #endif

    return  CMG_OK;
}
```

Listing 36 – CMG_LLIO_AVR_8D4CTL_8080.c, Teil 2

Zuerst wird das Display eingeschaltet, danach der Data-Port auf Null gesetzt, anschließend der Control-Port initialisiert, nach kurzer Wartezeit das Display mit `/CS` selektiert und zuletzt die Hintergrundbeleuchtung auf den Anfangswert gesetzt.

Zu beachten gilt, daß zuerst der Zielzustand des Ports ausgewählt und erst danach die Richtung auf Ausgang geschaltet wird. Dadurch werden kurze Signale in der falschen Richtung vermieden.

Beim Beenden der LLIO-Schicht werden die Schritte teilweise wieder rückgängig gemacht:

```
/*****************************************************************************
| CMG_LLIO_Exit
|-----------------------------------
| Exit (exit downwards: nothing)
+-----------------------------------------------------------------------------
| @Params:          none
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_LLIO_Exit( void )
{
    // switch off backlight if available
    #ifdef LLIO_BACKLIGHT_ENABLE
        SET_BACKLIGHT_OFF;
    #endif

    // deselect chip
    SET_CTRL_CS_DESELECT;
```

```
    // set data port std direction (all in) and std value
    SET_DATA_PORT_DIRECTION_IN;

    // reset ctrl pins direction (all in)
    CLR_BIT( LLIO_CTRL_iCS_PORT_DDR,    LLIO_CTRL_iCS_BIT );
    CLR_BIT( LLIO_CTRL_A0_PORT_DDR,     LLIO_CTRL_A0_BIT );
    CLR_BIT( LLIO_CTRL_iRD_PORT_DDR,    LLIO_CTRL_iRD_BIT );
    CLR_BIT( LLIO_CTRL_iWR_PORT_DDR,    LLIO_CTRL_iWR_BIT );

    // switch off display power if available
    #ifdef LLIO_DISPLAYPOWER_ENABLE
        SET_DISPLAYPOWER_OFF;
    #endif

    // wait until display is off
    CMG_Sleep_ms( LLIO_DISPLAY_INIT_WAITTIME_MS );
}
```

Listing 37 – CMG_LLIO_AVR_8D4CTL_8080.c, Teil 3

Zuerst schaltet die Funktion die Hintergrundbeleuchtung aus, selektiert das Display ab und setzt alle Ein-/Ausgänge auf den hochohmigen Zustand. Zuletzt wird die Stromzufuhr zum Display abgeschaltet.

Die eigentliche Steuerung des Displays findet in den Write- und Read-Funktionen statt:

```
/*************************************************************************
| CMG_LLIO_WriteA0
|-----------------------------------
|
+-------------------------------------------------------------------------
| @Params:          command/data byte
| @Return Value:    none
+*************************************************************************/
_PUBLIC
void CMG_LLIO_WriteA0( cmg_u8 byData )
{
    // set data port direction (all out)
    SET_DATA_PORT_DIRECTION_OUT;

    // set data port
    SET_DATA_PORT( byData );

    // ctrl pin sequence
    SET_CTRL_A0_0;
    SET_CTRL_WR_WRITE;

    // insert wait cycles
    LLIO_Write_WaitCycles();

    SET_CTRL_WR_CLEAR;

    // set data port std direction (all in)
    SET_DATA_PORT_DIRECTION_IN;
}
```

Listing 38 – CMG_LLIO_AVR_8D4CTL_8080.c, Teil 4

Der Reihe nach geschieht folgendes: Die Richtung des Data-Ports wird auf Ausgang geschaltet und anschließend der gewünschte Wert übergeben. Nun setzt der Prozessor die Adreßleitung `A0` auf Adresse 0 (wir sind in der Funktion …`WriteA0(…)`) und aktiviert das Write-Signal. Danach wartet er die definierte Anzahl der Takte ab und deaktiviert anschließend das Write-Signal wieder. Zuletzt wird die Richtung des Data-Registers wieder umgekehrt und auf Eingang gesetzt.

Wie man am Code deutlich sehen kann, lohnt sich hier der Einsatz von Makros. Dadurch wird die Übersichtlichkeit erheblich verbessert. Wenn hier nur `SET_BIT(…)` und `CLR_BIT(…)` oder sogar nur Konstrukte wie `PORTA &= 0x01;` stehen würden, könnte sich ein interessierter Mensch sicher nur mit erheblichem Aufwand einlesen.

Die Funktion `CMG_LLIO_WriteA1(…)` arbeitet genau gleich zu der gerade beschriebenen, aber anstatt Adresse 0 wird Adresse 1 mit dem Befehl `SET_CTRL_A0_1` ausgewählt.

Auch das Einlesen gestaltet sich sehr ähnlich:

```
/*****************************************************************************
| CMG_LLIO_ReadA0
|-----------------------------------
|
+----------------------------------------------------------------------------
| @Params:          none
| @Return Value:    data byte
+*****************************************************************************/
_PUBLIC
cmg_u8 CMG_LLIO_ReadA0( void )
{
    cmg_u8  byData;

    // ctrl pin sequence
    SET_CTRL_A0_0;
    SET_CTRL_RD_READ;

    // insert wait cycles
    LLIO_Read_WaitCycles();
    // one is _always_ necessary
    asm volatile ("nop");

    // get data port (std data port direction is in)
    byData = GET_DATA_PORT();

    // ctrl pin sequence
    SET_CTRL_RD_CLEAR;

    return byData;
}
```

Listing 39 – CMG_LLIO_AVR_8D4CTL_8080.c, Teil 5

Die Richtung des Data-Ports ist immer schon auf Eingang gesetzt. Wie zuvor beim Schreiben wird zuerst die Adresse ausgewählt, anschließend jedoch das /RD-Signal gesetzt. Danach wird gewartet und das Data-Byte eingelesen. Dieses liefert die Funktion dann auch als Ergebnis zurück, nachdem die /RD wieder zurückgesetzt hat.

Eine Besonderheit beim Warten ist das zusätzliche nop. Dies ist funktionsbedingt bei AVR-Prozessoren der Fall: Sie setzen einen Ausgang am **Ende** des Taktzyklus, lesen jedoch den Status eines Eingangs am **Anfang** des Zyklus'. Somit würde, komplett ohne Warte-nops, der Zustand des Data-Registers zu dem Zeitpunkt eingelesen werden, wenn sich /RD gerade zu ändern beginnt.

Die Funktion CMG_LLIO_ReadA1(…) läuft – bis auf die Adreßauswahl – wieder äquivalent dazu ab.

Mit diesen vier Funktionen läßt sich das Display komplett steuern.

Zusätzlich besitzt jede Schicht noch die Power-Management-Funktion: In der LLIO-Schicht wertet diese nur die Werte CMG_PWR_BACKLIGHT_ON und CMG_PWR_BACKLIGHT_OFF aus.

5.5.2. 6800-Bus

Die Datei für den 6800-Bus `CMG_LLIO_AVR_8D4CTL_6800.c` ist der für den 8080-Bus nahezu identisch. Lediglich im Detail unterschieden sich die Dateien: Die beiden Signale `/RD` und `/WR` werden durch die Signale `R/W` und `E` ersetzt, und die Steuerung wird entsprechend angepaßt:

`R/W` steht im Grundzustand immer auf Write, muß also nur in der Einlese-Funktion entsprechend geändert werden:

```
/*****************************************************************************
| CMG_LLIO_WriteA0
|-----------------------------------
|
+----------------------------------------------------------------------------
| @Params:          command/data byte
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_LLIO_WriteA0( cmg_u8 byData )
{
    // set data port direction (all out)
    SET_DATA_PORT_DIRECTION_OUT;

    // set data port
    SET_DATA_PORT( byData );

    // ctrl pin sequence
    SET_CTRL_A0_0;
    SET_CTRL_E_ENABLE;

    // insert wait cycles
    LLIO_Write_WaitCycles();

    SET_CTRL_E_DISABLE;

    // set data port std direction (all in)
    SET_DATA_PORT_DIRECTION_IN;
}
```

Listing 40 – CMG_LLIO_AVR_8D4CTL_6800.c, Teil 1

Bzw. das Einlesen:

```
/*****************************************************************************
| CMG_LLIO_ReadA0
|-----------------------------------
|
+----------------------------------------------------------------------------
| @Params:          none
| @Return Value:    data byte
+*****************************************************************************/
_PUBLIC
cmg_u8 CMG_LLIO_ReadA0( void )
{
    cmg_u8  byData;

    // ctrl pin sequence
    SET_CTRL_RiW_READ;
    SET_CTRL_A0_0;
    SET_CTRL_E_ENABLE;

    // insert wait cycles
    LLIO_Read_WaitCycles();
    // one is _always_ necessary
    asm volatile ("nop");

    // get data port (std data port direction is in)
    byData = GET_DATA_PORT();

    // ctrl pin sequence
    SET_CTRL_E_DISABLE;
    SET_CTRL_RiW_WRITE;

    return byData;
}
```

Listing 41 – CMG_LLIO_AVR_8D4CTL_6800.c, Teil 2

5.6. ARM

Für den ARM-Prozessor ist zur Zeit nur das 8080-Protokoll implementiert, das 6800er kann aber – wie im AVR-Beispiel gesehen – sehr einfach hinzugefügt werden.

5.6.1. 8080-Bus

Interessanterweise sehen die Funktionen in der Datei `CMG_LLIO_ARM_8D4CTL_8080.c` nahezu identisch wie die aus dem AVR-Pendant aus. Lediglich die Hilfsmakros ändern sich für die Plattformanpassung:

```
// bitmasks
#define     LLIO_CTRL_iCS_BITMASK       BIT( LLIO_CTRL_iCS_BIT )
#define     LLIO_CTRL_A0_BITMASK        BIT( LLIO_CTRL_A0_BIT )
#define     LLIO_CTRL_iRD_BITMASK       BIT( LLIO_CTRL_iRD_BIT )
#define     LLIO_CTRL_iWR_BITMASK       BIT( LLIO_CTRL_iWR_BIT )
#define     LLIO_DISPLAYPOWER_BITMASK   BIT( LLIO_DISPLAYPOWER_BIT )
#define     LLIO_BACKLIGHT_BITMASK      BIT( LLIO_BACKLIGHT_BIT )
#define     LLIO_DATA_BITMASK           ( 0x000000ffUL << LLIO_DATA_STARTBIT )


// macros for ctrl interaction
#define     SET_CTRL_CS_SELECT          LLIO_CTRL_iCS_PORT_CLR = LLIO_CTRL_iCS_BITMASK
#define     SET_CTRL_CS_DESELECT        LLIO_CTRL_iCS_PORT_SET = LLIO_CTRL_iCS_BITMASK

#define     SET_CTRL_A0_0               LLIO_CTRL_A0_PORT_CLR = LLIO_CTRL_A0_BITMASK
#define     SET_CTRL_A0_1               LLIO_CTRL_A0_PORT_SET = LLIO_CTRL_A0_BITMASK

#define     SET_CTRL_RD_READ            LLIO_CTRL_iRD_PORT_CLR = LLIO_CTRL_iRD_BITMASK
#define     SET_CTRL_RD_CLEAR           LLIO_CTRL_iRD_PORT_SET = LLIO_CTRL_iRD_BITMASK

#define     SET_CTRL_WR_WRITE           LLIO_CTRL_iWR_PORT_CLR = LLIO_CTRL_iWR_BITMASK
#define     SET_CTRL_WR_CLEAR           LLIO_CTRL_iWR_PORT_SET = LLIO_CTRL_iWR_BITMASK


// macros for data interaction
#define     SET_DATA_PORT_DIRECTION_IN  LLIO_DATA_PORT_DIR &= ( ~LLIO_DATA_BITMASK )
#define     SET_DATA_PORT_DIRECTION_OUT LLIO_DATA_PORT_DIR |= LLIO_DATA_BITMASK

#define     SET_DATA_PORT(data)         {                                   \
   cmg_u32  dwData = ( (cmg_u32)(data) ) << LLIO_DATA_STARTBIT;             \
   LLIO_DATA_PORT_SET = dwData;                                             \
   LLIO_DATA_PORT_CLR = ( ~dwData ) & LLIO_DATA_BITMASK; }
#define     GET_DATA_PORT()             ( (cmg_u8)( LLIO_DATA_PORT_PIN >> LLIO_DATA_STARTBIT ) )


// macros for power interaction
// display power
#ifdef LLIO_DISPLAYPOWER_ENABLE
    #ifndef LLIO_DISPLAYPOWER_INVERTED
        // standard power logic
        #define     SET_DISPLAYPOWER_ON         LLIO_DISPLAYPOWER_PORT_SET = LLIO_DISPLAYPOWER_BITMASK
        #define     SET_DISPLAYPOWER_OFF        LLIO_DISPLAYPOWER_PORT_CLR = LLIO_DISPLAYPOWER_BITMASK
    #else
        // inverted power logic
        #define     SET_DISPLAYPOWER_ON         LLIO_DISPLAYPOWER_PORT_CLR = LLIO_DISPLAYPOWER_BITMASK
        #define     SET_DISPLAYPOWER_OFF        LLIO_DISPLAYPOWER_PORT_SET = LLIO_DISPLAYPOWER_BITMASK
    #endif
#endif

// backlight
#ifdef LLIO_BACKLIGHT_ENABLE
    #ifndef LLIO_BACKLIGHT_INVERTED
        // standard backlight logic
        #define     SET_BACKLIGHT_ON            LLIO_BACKLIGHT_PORT_SET = LLIO_BACKLIGHT_BITMASK
        #define     SET_BACKLIGHT_OFF           LLIO_BACKLIGHT_PORT_CLR = LLIO_BACKLIGHT_BITMASK
    #else
        // inverted backlight logic
        #define     SET_BACKLIGHT_ON            LLIO_BACKLIGHT_PORT_CLR = LLIO_BACKLIGHT_BITMASK
        #define     SET_BACKLIGHT_OFF           LLIO_BACKLIGHT_PORT_SET = LLIO_BACKLIGHT_BITMASK
    #endif
#endif
```

Listing 42 – CMG_LLIO_ARM_8D4CTL_8080.c

Hier werden zuerst die jeweiligen Bitmasken für die einzelnen Signale berechnet, die dann später in den SET- und CLR-Registern zum Setzen und Löschen verwendet werden. Lediglich das Einlesen und Ausgeben des 8-Bit Data-Registers ist etwas aufwendiger.

5.7. X86

Möglich sind hier die Ausgabe über den Parallel-Port oder die USB-Schnittstelle mit Hilfshardware, wie einem FTDI-Chip.

Bei der Programmierung der parallelen Schnittstelle muß auf plattformabhängige Besonderheiten geachtet werden. Windows, zum Beispiel, bietet zwar eine einfache Möglichkeit zum Zugriff auf die Schnittstelle an, allerdings nur schreibend. Als Lösungsmöglichkeit muß man direkt auf die internen Ports der Schnittstelle zugreifen. Unter Windows kann man aber (außer in den veralteten Windows 9x -Versionen) aus Sicherheitsgründen nicht direkt auf Ports zugreifen. Zur Lösung des Problems muß man einen Kerneltreiber schreiben, der – intern im Ring 0 des Prozessors – die nötigen Rechte für den gewünschten Prozeß setzt. Es gibt schon einige fertige Treiber, wovon – meines Erachtens – der beste und einfachste *giveio*[25] ist. Dieser wird einmal installiert und kann dann in dem betreffenden Programm geladen werden. Danach steht diesem Prozeß der Port-Zugriff offen. Unter Linux gestaltet sich der Zugriff um einiges einfacher.

[25] Treiber giveio.sys von Dale Roberts, Installation von Paula Tomlinson, unterstützt von Andy Clark und Ekkehard Pofahl

6. LLIO – Emulatoren

Im Gegensatz zur eben beschriebenen LLIO-Schicht, greift die LLIO-Emulator-Schicht nicht direkt auf physikalisch vorhandene Ports und Displays zu, sondern emuliert die Displays mit ihren Ausgaben am PC.

Um dies zu erreichen und möglichst viel des CMG-Quellcodes damit testen bzw. die zu erwartenden Ausgaben möglichst originalgetreu simulieren zu können, muß die Schnittstelle auf der niedrigsten Ebene sein. Hierfür eignet sich die LLIO-Schicht optimal.

6.1. Aufbau

Die Sourcen und die Konfigurationsdatei liegen in `CMG/1_LLIO/EMU/`.

Zuerst eine kleine Übersicht über den Aufbau:

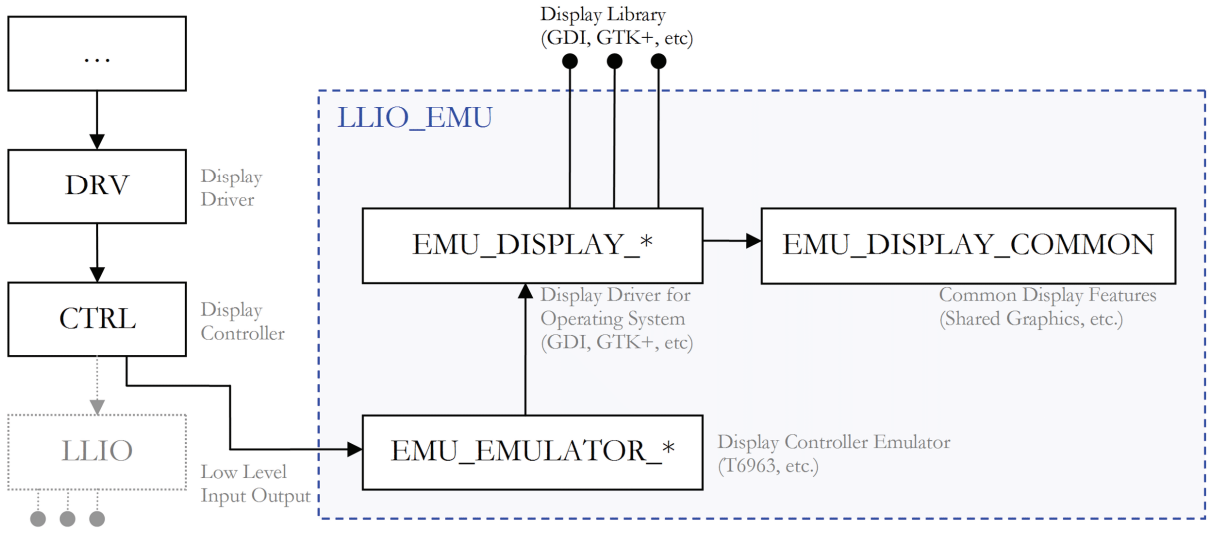


Abbildung 37 – LLIO_EMU-Aufbau

Links ist die normale Schichtenreihenfolge aufgeführt. Als Vorletztes kommt die CTRL-Schicht für den Displaycontroller. Diese spricht normalerweise die LLIO-Schicht an, welche direkt auf die Hardware geht. Soll das Display nun emuliert werden, wird die unterste Schicht LLIO umgangen und an deren Stelle die LLIO_EMU-Schicht angesprochen.

Die Schnittstelle der Emulationsschicht bleibt deshalb nach außen hin genau dieselbe wie bei der normalen LLIO-Schicht, nach innen jedoch emuliert sie einen Displaycontroller und stellt dessen Ergebnisse am Monitor dar.

Im folgenden Abschnitt werden die einzelnen Teile der Schicht näher erläutert.

6.1.1. EMU_EMULATOR_*

Diese Komponente bietet nach außen die LLIO-Schicht und emuliert einen beliebigen Display-Controller. Was normalerweise extern die Hardware übernimmt, muß dieser Treiber hier in Software nachbilden. Anschließend greift er auf die genormte Schnittstelle der Emulator-Ausgabe-Komponente (`EMU_DISPLAY`) zu, um die berechnete Ausgabe anzuzeigen.

6.1.2. EMU_DISPLAY_*

Diese Komponente greift auf die systemspezifischen Grafikfunktionen der verwendeten Grafik-Library zu, um das simulierte Display darzustellen. Im Groben besteht diese Darstellung aus zwei Teilen: Dem Hintergrund, also das leere Display, und darauf gezeichnet den eigentlichen Inhalt. Um nicht für jede unterstützte Ausgabe-Bibliothek die Bilder einzeln abzuspeichern und einzuladen, wurden diese Funktionen in die für alle Treiber gemeinsame Datei `CMG_LLIO_EMU_DISPLAY_COMMON.C` ausgelagert.

6.1.3. ANZEIGEBEISPIELE

Um sich das Ergebnis und die dafür notwendigen Schritte besser vorstellen zu können, hier zwei Screenshots mit Demo-Ausgabe:

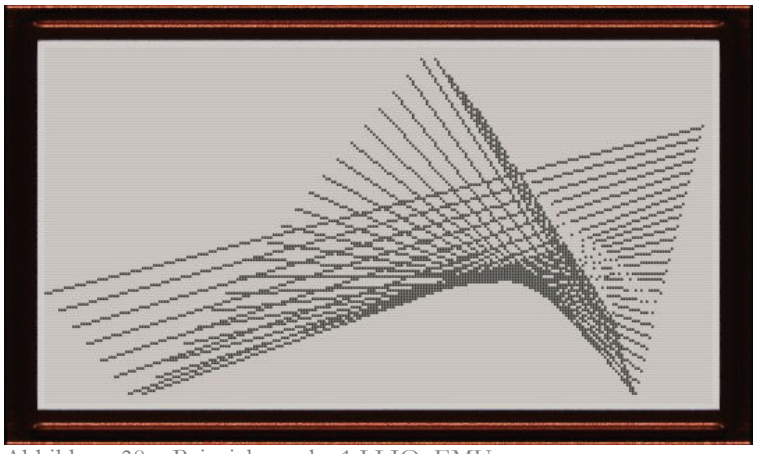


Abbildung 38 – Beispielausgabe 1 LLIO_EMU

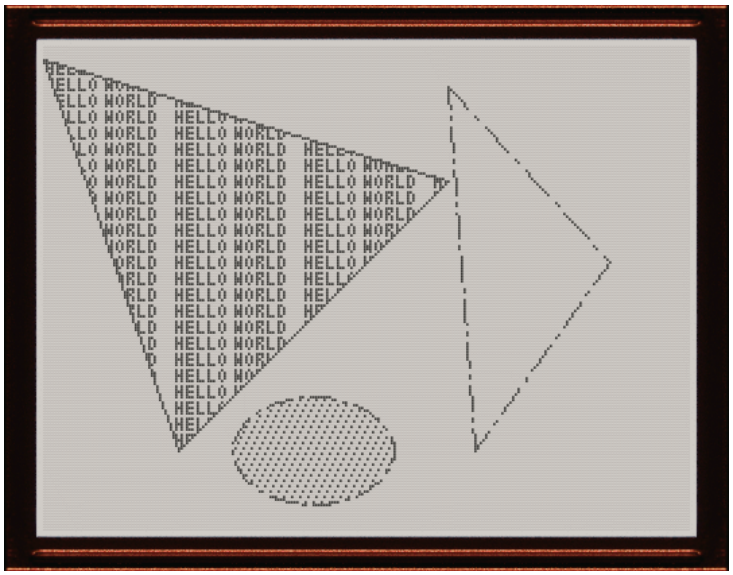


Abbildung 39 – Beispielausgabe 2 LLIO_EMU

Diese Bilder zeigen die Bildschirmausgabe des Emulators. Ziel ist es, das Display so realistisch wie möglich am PC nachzubilden. Als Vorlage für die Grafik dienten Fotos realer Displays, die anschießend bearbeitet wurden. Ein Pixel ist auf dem Bildschirm genau 2x2 Pixel groß. Für eine klarer simulierte Ausgabe kann auch auf einen reinen Schwarz-Weiß-Modus umgeschaltet werden.

6.2. SCHNITTSTELLEN

Die Schnittstelle bleibt – wie zuvor schon beschrieben – nach außen hin genau dieselbe wie bei der normalen LLIO-Schicht, nach innen jedoch emuliert sie einen Displaycontroller und stellt dessen Ergebnisse am Monitor dar.

Der folgende Abschnitt ist identisch zur LLIO-Schicht und kann somit im Austausch dafür benutzt werden:

```
    //-- Init/Exit ---------------------------------------------------------
        cmg_Result  CMG_LLIO_Init( void );
        void        CMG_LLIO_Exit( void );
        void        CMG_LLIO_PowerManagement( cmg_mu8 uState );
    //----------------------------------------------------------------------

    //-- Write -------------------------------------------------------------
        void        CMG_LLIO_WriteA0( cmg_u8 byData );
        void        CMG_LLIO_WriteA1( cmg_u8 byData );
    //----------------------------------------------------------------------

    //-- Read --------------------------------------------------------------
        cmg_u8      CMG_LLIO_ReadA0( void );
        cmg_u8      CMG_LLIO_ReadA1( void );
    //----------------------------------------------------------------------
```

Listing 43 – CMG_LLIO_EMU_DISPLAY.h

Die Display-Komponente besitzt ebenfalls eine genormte Schnittstelle. Sie bietet – neben den normalen Init/Exit-Funktionen und zwei Statusfunktionen nur eine einzige, wichtige Funktion: Das Setzen der Farbe von einzelnen Pixeln:

```
    //-- Init/Exit ---------------------------------------------------------
        cmg_Result  CMG_LLIO_DISPLAY_Init( void );
        void        CMG_LLIO_DISPLAY_Exit( void );
        void        CMG_LLIO_DISPLAY_PowerManagement( cmg_mu8 uState );
    //----------------------------------------------------------------------

    //-- Write -------------------------------------------------------------
        void        CMG_LLIO_DISPLAY_SetPixel( cmg_Coord iX, cmg_Coord iY, cmg_Color Color );
    //----------------------------------------------------------------------

    //-- Additionals -------------------------------------------------------
            #define     CMG_LLIO_DISPLAY_MODE_OFF           0
            #define     CMG_LLIO_DISPLAY_MODE_ON            1
        void        CMG_LLIO_DISPLAY_SetMode( cmg_mu8 uMode );
        void        CMG_LLIO_DISPLAY_ErrorMessage( cmg_char *strMessage );
    //----------------------------------------------------------------------
```

Listing 44 – CMG_LLIO_EMU_DISPLAY.h

Mit `SetMode(…)` kann das Display zur Simulation ein- und ausgeschaltet werden. `ErrorMessage(…)` reicht einen Fehler aus der Emulator-Schicht weiter, die zur Anzeige gebracht werden soll. Dies ist nötig wenn nicht alle Teile des Display-Controllers auch implementiert wurden. Nicht verwendete Befehle oder Kombinationen geben Fehler aus.

Zuletzt noch die Schnittstellendefinition für die gemeinsamen Display-Funktionen:

```
//-----------------------------------------------------------------------------
// typedefs
//-----------------------------------------------------------------------------
typedef struct
{
    cmg_u8      byBlue;
    cmg_u8      byGreen;
    cmg_u8      byRed;
    cmg_u8      byAlpha;
} cmg_PaletteColor;
//-----------------------------------------------------------------------------

[…]

    //-- Byte Order Setting -------------------------------------------------
            #define     CMG_LLIO_DISPLAY_COMMON_BYTEORDER_GDI                0x00
            #define     CMG_LLIO_DISPLAY_COMMON_BYTEORDER_GTK                0x01
        void CMG_LLIO_DISPLAY_COMMON_SetByteOrder( cmg_mu8 uByteOrder );
    //-----------------------------------------------------------------------

    //-- Image Creation -----------------------------------------------------
        cmg_Result CMG_LLIO_DISPLAY_COMMON_CreateLCDImage8( cmg_u8 **ppbyImage, cmg_PaletteColor **ppPal,
                            cmg_Coord iWidth, cmg_Coord iHeight );
        cmg_Result CMG_LLIO_DISPLAY_COMMON_CreateLCDImage32( cmg_u32 **ppdwImage, cmg_Coord iWidth,
                            cmg_Coord iHeight );
    //-----------------------------------------------------------------------

    //-- Pixel Color Creation -----------------------------------------------
            #define     CMG_LLIO_DISPLAY_COMMON_PIXELCOLORTYPE_PHOTO_BLON    0x00
            #define     CMG_LLIO_DISPLAY_COMMON_PIXELCOLORTYPE_PHOTO_BLOFF   0x01
            #define     CMG_LLIO_DISPLAY_COMMON_PIXELCOLORTYPE_EXACT         0x02
        void CMG_LLIO_DISPLAY_COMMON_CreatePixelColor_2x2_32bpp( cmg_u32 aadwPixelColor[256][4], cmg_mu8
                            uType );
    //-----------------------------------------------------------------------
```

Listing 45 – CMG_LLIO_EMU_DISPLAY_COMMON.h

Die Details dazu werden in 6.5.1 besprochen.

6.3. KONFIGURATION

Wie in allen weiteren Schichten auch, existiert in dieser Komponente nur noch eine einzige Konfigurationsdatei für die komplette Schicht:

```
// Emulated Display details:
// -------------------------
// Use values defined by CTRL driver below. Use only own values if you know
// what you do:
#define     CMG_LLIO_EMU_WIDTH                      CMG_CTRL_WIDTH
#define     CMG_LLIO_EMU_HEIGHT                     CMG_CTRL_HEIGHT
#define     CMG_LLIO_EMU_BITDEPTH                   CMG_CTRL_BITDEPTH
//
//
// Driver emulator tu use:
// -----------------------
// You should use the same driver as selected in DRV.
// Use only _ONE_ of this list:
#define     CMG_LLIO_EMU_EMULATOR_T6963
//
//
// Display presentation system to use:
// -----------------------------------
// * Use Microsoft Windows GDI: (Compatible with all Windows versions):
// Use only _ONE_ of this list:
//#define     CMG_LLIO_EMU_DISPLAY_GDI
#define     CMG_LLIO_EMU_DISPLAY_GTK
```

Listing 46 – CMG_LLIO_EMU.config

Hier wird die gewünschte Pixelauflösung und die Farbtiefe des Display angegeben. In dem obigen Fall die gleiche, die auch für die CTRL-Schicht verwendet wird.

Wie der Aufbau schon vermuten läßt, müssen der verwendete Displaycontroller-Emulator und der Display-Ausgabe-Treiber ausgewählt werden.

6.4. Controller-Emulatoren

Die genaue Beschreibung würde den Rahmen der Arbeit sprengen. Im folgenden werden nur die wichtigsten Punkte beschrieben.

Es können Controller-Emulatoren für beliebige Controller geschrieben werden. In CMG ist bis jetzt nur der T6963 implementiert, was aber normal auch ausreicht, denn damit läßt sich alles simulieren. Die einzig sinnvolle Aufgabe einen weiteren Treiber zu emulieren ist, damit den Controller-Treiber (CTRL-Schicht) explizit zu testen. Alle weiter oben liegenden Schichten sollten wieder gleich funktionieren.

Der Emulator `CMG_LLIO_EMU_EMULATOR_T6963.c` besitzt eine Liste mit den Befehlen des T6963s. Alle notwendigen Register des Controllers werden in globalen Variablen gespeichert. Darunter befinden sich die Daten-Register `D1`, `D2` und `DRet`, sowie die Zustandsregister für den `AddressPtr`, die `GraphicAddr`, die `GraphicArea` und den `AutoMode`:

```
//-----------------------------------------------------------------------------
// globals
//-----------------------------------------------------------------------------
cmg_u8      g_EMULATOR_byD1;                    // data 1 register
cmg_u8      g_EMULATOR_byD2;                    // data 2 register
cmg_u8      g_EMULATOR_byDRet;                  // data return register

cmg_u16     g_EMULATOR_wAddressPtr;             // current address pointer
cmg_u16     g_EMULATOR_wGraphicHomeAddr;        // graphic home address
cmg_u8      g_EMULATOR_byGraphicArea;           // graphic area (linesize in bytes)
cmg_u8      g_EMULATOR_byAutoMode;              // current automode

cmg_u8      *g_EMULATOR_pbyVideoMem = NULL;     // emulator native video memory
//-----------------------------------------------------------------------------
```

Listing 47 – CMG_LLIO_EMU_EMULATOR_T6963.c, Teil 1

`VideoMem` beinhaltet den kompletten Video-Speicher des Displays im 1:1-Fomat.

In `Init(…)` werden die Register auf die Grundwerte initialisiert und der Displayspeicher allokiert. Weiter wird der Init-Aufruf zu der Display-Komponente weitergeleitet.

`Exit()` gibt den Speicher wieder frei und beendet auch die Display-Komponente.

Die vier Funktionen zum Lesen und Schreiben der beiden Register emulieren den Controller.

Immer, sobald ein ausgeführter Befehl das interne `VideoMem` verändert, werden die betreffenden Pixel sofort mit der `Display_SetPixel`-Funktion auf der Oberfläche aktualisiert.

6.5. Bildschirmausgabe

Die Aufgabe der Ausgabemodule ist das Darstellen des Speicherinhalts des simulierten Displays.

Es besitzt nach außen nur wenige, notwendige Funktionen: Die `Init()`- und `Exit()`-, sowie die `SetPixel(…)`-Funktionen. Um den Programmfluß nicht zu unterbrechen, wird in der Initialisierung ein neuer Thread erzeugt, welcher sich um die Darstellung des Displays kümmert.

6.5.1. GEMEINSAME KOMPONENTEN

Die Datei `CMG_LLIO_EMU_DISPLAY_COMMON.c` bietet Funktionen zum Laden und Erzeugen des LCD-Hintergrundfotos sowie die Pixelwerte, zur Darstellung der jeweiligen LCD-Pixel, an.

`SetByteOrder(…)` setzt die richtige Byte-Reihenfolge für die Paletteneinträge und die zurückzuliefernden Bilder. Zur Auswahl stehen das GDI- und das GTK-Format.

`CreateLCDImage8(…)` erwartet die gewünschte Größe des zu erzeugenden Displayhintergrunds und gibt einen Pointer mit dem neu erzeugten Bild zurück, welches pro Pixel aus einem Byte besteht – also insgesamt eine Größe von $x * y$ Pixel besitzt.

Diese Funktion reserviert zuerst die benötigten Speicherbereiche und lädt anschließend die in eingebetteten Strings codierten Teilbilder zum Erzeugen des LCD-Hintergrunds. Diese Teilbilder bestehen aus einer LCD-Ecke und einem horizontalen sowie vertikalen Teilstück. Diese werden, je nach gewünschter Größe, richtig gespiegelt und übereinander gelegt in das Zielbild eingefügt.

Die Palette wird ebenfalls aus einem eingebetteten String geladen und mit dem fertig aufgebauten Bild zurückgeliefert.

`CreateLCDImage32(…)` liefert kein indiziertes Bild mit Palette zurück, sondern nur das fertige Bild mit 32 Bit Farbtiefe. Intern ruft es die Funktion `CreateLCDImage8(…)` auf und konvertiert das Bild anschließend in das neue.

`CreatePixelColor_2x2_32bpp(…)` liefert ein Array mit 256 Einträgen – den jeweiligen Helligkeitsstufen – zurück. Jede Helligkeitsstufe besteht aus einem Array mit 4 Einträgen, welche jeweils einen der 2x2 Pixel eines LCD-Pixels beschreibt. Jeder dieser Pixel ist wiederum ein 32-Bit-Wert mit der direkten Farbinformation.

Diese Werte werden anhand einer Hilfs-Tabelle, der Farbtabelle, gefüllt. Die Tabelle beinhaltet für jeden Farbkanal der vier Sub-Pixel den minimalen und maximalen Wert. Die Funktion füllt das Array nun linear mit Hilfe dieser Informationen auf und liefert das Ergebnis zurück.

Zurzeit sind drei Farbtabellen definiert:

Farbtabelle	Beschreibung	Beispiel
`aaPixMixMap_PhotoBlOn`	Fotorealistisch mit Hintergrundbeleuchtung	
`aaPixMixMap_PhotoBlOff`	Fotorealistisch ohne Hintergrundbeleuchtung	
`aaPixMixMap_Exact`	Exakte Darstellung	

Tabelle 22 – Emulator – Farbtabellen

6.5.2. GDI AUSGABE

GDI bedeutet *Graphics Device Interface* und ist die für die Grafikausgabe zuständige Komponente des *Windows*-Betriebssystems von *Microsoft*[26]. Es gibt mittlerweile einige Aufsätze und Erweiterungen dieser Schnittstelle, um dem Programmierer die Arbeit zu erleichtern und mächtigere Oberflächen zu erzeugen.

GDI besitzt zwei große Vorteile, welche es für CMG unter Windows zur ersten Wahl macht:

- Reines C-Interface
- Kompatibel mit allen Windows-Versionen

`DISPLAY_Init()` wird zu Beginn zum Erzeugen und Darstellen des Displays aufgerufen. Zuerst setzt es die globalen Variablen auf die Standardwerte und anschließend mit `DISPLAY_COMMON_SetByteOrder(…)` die GDI-Byte-Reihenfolge.

Danach reserviert die Funktion alle benötigten Speicherbereiche. Dazu gehören sowohl eine Kopie des aktuellen Display-Inhalts, als auch das leere LCD-Hintergrundbild, sowie das anzeigefertige Bild des Display-Inhalts mit den jeweils 2x2 Subpixeln.

Im letzten Teil wird der neue Thread erzeugt, der sich um die Darstellung kümmert. Davor erzeugt dieser Codeteil ein System-Event, damit der neue Thread dem Aufrufer die erfolgreiche Anzeige des Fensters mitteilen kann. Nachdem also der Thread neu erzeugt und gestartet wurde, wartet die Init-Funktion noch auf das Auslösen des Events. Wird ein Fehler zurückgeliefert oder der Event nicht innerhalb einer bestimmten Zeit ausgelöst, schlägt die Init-Funktion fehl und gibt die entsprechende Fehlerbeschreibung zurück.

Bei `DISPLAY_Exit()` wird dem Thread eine Nachricht zum Beenden geschickt, darauf gewartet und anschließend alle Speicherbereiche wieder freigegeben.

Der Thread selbst (`DISPLAY_ThreadProc()`) führt die notwendigen Schritte zum Darstellen des Fensters durch. Dazu muß er die Fensterklasse anlegen und registrieren, das Fenster erzeugen und darstellen, das Popup-Menü hinzufügen und einen Timer erstellen. Sind alle diese Schritte erfolgreich verlaufen, setzt er den Event zum Signalisieren des Erfolgs an den Aufrufer.

Anschließend begibt sich der Thread in die Nachrichtenschleife und wartet dort solange auf Ereignisse, bis das Fenster wieder geschlossen werden soll.

Die Callback-Funktion des Fensters implementiert das Neuzeichnen des Displays, das Erscheinungsbild sowie die Mausbehandlung.

Um die Aktualisierung des Fensters mit dem Displayinhalt zu erreichen, wird ein Timer verwendet. Das System ruft diesen alle 25ms auf, also mit maximal 40fps. Der Timer überprüft, mit Hilfe eines Flags, ob zwischenzeitlich eine Änderung des Displayinhaltes stattgefunden hat und zeichnet dann gegebenenfalls den Displayausschnitt neu.

Zuletzt sollte noch die Funktion `DISPLAY_SetPixel(…)` beachtet werden: Wird sie aufgerufen, ändert sie das betreffende Byte im internen Displayspeicher und anschließend die betreffenden vier Stellen (2x2 Pixel) in dem Bild, welches letztendlich gezeichnet werden soll. Diese Werte nimmt sie aus der Tabelle der zuvor beschriebenen Funktion `CreatePixelColor_2x2_32bpp(…)`. Danach muß nur noch das globale Flag gesetzt

[26] Microsoft Corporation, One Microsoft Way, Redmond, WA, USA, http://www.microsoft.com

werden, welches eine Änderung im Bild anzeigt, damit der Timer beim nächsten Aufruf das Neuzeichnen veranlaßt.

Das globale Flag ist notwendig, da sonst bei jeder Änderung eines Pixels vier Pixel neu gezeichnet werden müßten. Mit Hilfe des Flags sammelt das Modul die Änderungen und gibt diese auf einmal alle 25ms aus. Diese Verzögerung ist vom Auge nicht nachvollziehbar, bringt jedoch enorme Performanceverbesserungen.

6.5.3. GTK+ AUSGABE

Um nicht nur Windows zu unterstützen, sondern auch eine freie, plattformunabhängige Grafik-Schnittstelle, eignet sich *GTK+*[27] am besten. *GTK+* ist Teil des *GNU-Projekts*[28] und steht unter der *GNU-LGPL-Lizenz*[29].

GTK+ besitzt ebenfalls den Vorteil mit reinem ANSI-C ansprechbar zu sein. Zudem ist es für fast alle gängigen Betriebssysteme erhältlich. Einziger Nachteil ist, daß es systembedingt langsamer als GDI ist.

Der Aufbau der DISPLAY-Quellcodedatei ist nahezu identisch mit der GDI-Version – jedenfalls vom Ablauf und der Funktion her. Einzig und allein die Systemaufrufe zur Threadverwaltung und Steuerung der Fenster müssen den Aufrufen und dem Modell von GTK+ angepaßt werden.

[27] GTK+, The GIMP Toolkit, http://www.gtk.org
[28] GNU, GNU's Not Unix! - Free Software, Free Society, http://www.gnu.org
[29] LGPL, GNU Lesser General Public License, http://www.gnu.org/licenses/lgpl.html

7. CTRL – Controller

CTRL, der Controller-Treiber, hat die Aufgabe, die Komplexität des zu verwendenden LCD-Controllers vollständig zu verbergen.

Die Schnittstelle von CTRL ist eine Zusammenfassung von grundlegenden, notwendigen Funktionen um performant auf die Displays zugreifen zu können.

Sicherlich würde es auch ausreichen, nur ein `GetByte(…)` und ein `SetByte(…)` zu exportieren, dem die gewünschte Adresse mitgeliefert wird. Allerdings bieten fast alle Controllertypen Hardwareunterstützung für Teile oder sogar den kompletten Umfang der CTRL-Schnittstelle an. Diese können somit sehr vorteilhaft verwendet oder – falls nicht oder nur teilweise vom Controller unterstützt – in Software implementiert werden.

7.1. Aufbau

Die Sourcen und die Konfigurationsdatei liegen in `CMG/2_CTRL/`.

Zuerst die Übersicht des Aufbaus dieser Schicht:

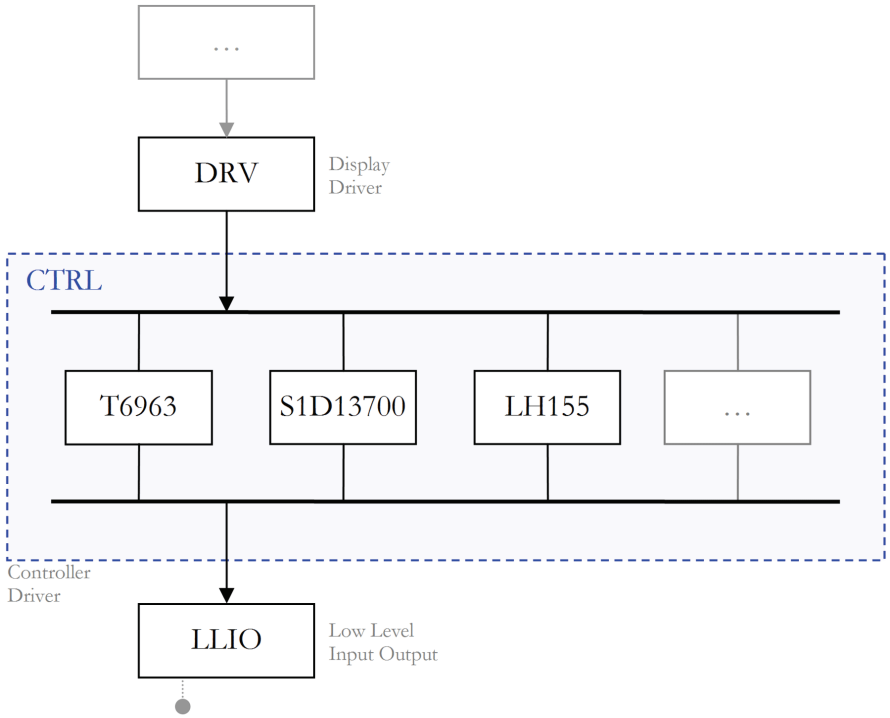


Abbildung 40 – CTRL-Aufbau

Die CTRL-Schicht hängt unter der DRV-Schicht und greift ausschließlich auf die LLIO-Schicht zu. Innerhalb der Schicht stehen verschiedene, gleichwertige Module zur Verfügung, die den jeweiligen LCD-Controller ansprechen können.

7.2. SCHNITTSTELLEN

Zuerst einmal die fertige Schnittstellendefinition:

```
//-----------------------------------------------------------------------------
// Globals
//-----------------------------------------------------------------------------
extern cmg_Coord   g_CTRL_iDisplayWidth;           // display x resolution
extern cmg_Coord   g_CTRL_iDisplayHeight;          // display y resolution
extern cmg_mu16    g_CTRL_uBytesPerLine;           // bytes per line
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
// Declarations      (for details see below)
//-----------------------------------------------------------------------------

    //-- Init/Exit ----------------------------------------------------------
        cmg_Result  CMG_CTRL_Init( void );
        void        CMG_CTRL_Exit( void );
        void        CMG_CTRL_PowerManagement( cmg_mu8 uState );
        cmg_Result  CMG_CTRL_ControllerSpecific( cmg_mu8 uFunction, cmg_void *pParam );
    //-----------------------------------------------------------------------

    //-- Functions ----------------------------------------------------------
        void        CMG_CTRL_FastClearScreen( cmg_u8 byCol );
        void        CMG_CTRL_SetAddressPtr( cmg_ScrAddr wScreenAddress );
    //-----------------------------------------------------------------------

    //-- Get/Set with indirect direction ------------------------------------
                                                          //_RD__WR_
            #define     CTRL_DIR_RD_NONE_WR_FASTEST     1       //  o   *
            #define     CTRL_DIR_RD_NONE_WR_RIGHT       2       //  o  ->
            #define     CTRL_DIR_RD_NONE_WR_DOWN        4       //  o   v
            #define     CTRL_DIR_RD_RIGHT_WR_INVALID    8       // ->   /
        void        CMG_CTRL_SetDirections( cmg_ms8 iDirections );

        void        CMG_CTRL_SetByte( cmg_u8 byData );
        cmg_u8      CMG_CTRL_GetByte( void );
    //-----------------------------------------------------------------------

//-----------------------------------------------------------------------------
```

Listing 48 – CMG_CTRL.h

Durch die enorme Vielzahl an verfügbaren Controllertypen und deren möglichen Hardwareunterstützungen war die Wahl der endgültigen Schnittstelle nicht einfach.

Zuerst mußte jeder Treiber noch zusätzlich einzelne Bit-Zugriffe unterstützen – also ein einzelnes Bit aus einem bestimmten Byte setzen oder löschen. Auch die Lese- und Schreibrichtungen konnten jeweils beliebig in den Richtungen *links*, *rechts*, *oben*, *unten* oder *nichts* verändert werden.

So konnten die Hardwarefunktionen möglichst gut ausgenutzt werden und bei den restlichen wurde dieser Teil eben in Software erledigt.

Beim Implementieren und Optimieren der DRV-Schicht – also der nächsten Schicht – stellte sich jedoch heraus, daß nur ein kleiner Bruchteil dieser Funktionen überhaupt notwendig war. Alle Funktionen kommen mit dem jetzigen Funktionsumfang aus. Teilweise stellten sich sogar die Bit-Zugriffe als langsamer heraus als ein Lesen mit anschließendem Schreiben eines kompletten Bytes.

Somit ergab sich obige Schnittstelle als beste Mischung aus Hardwareunterstützung und möglichst geringem Umfang des Controller-Treibers.

Je kleiner der Umfang des Treibers, um so kleiner ist natürlich auch der Aufwand einen neuen Treiber – für einen noch nicht unterstützten Controllertyp – hinzuzufügen. Alle anderen Softwarekomponenten bleiben von so einer Anpassung unberührt.

7.3. FUNKTIONSWEISE DES MODELLS

Als bestes Modell zur Abstraktion der Funktionalität stellte sich das pointerbasierte Adreßsystem mit byteweisem Lese- und Schreibzugriff heraus.

Dieses muß für alle Controller-Treiber implementiert werden, notfalls auch emuliert.

Zur vollständigen Unterstützung sind folgende vier Funktionen nötig:

- `SetAddressPtr(…)`
- `SetDirections(…)`
- `GetByte()`
- `SetByte(…)`

Das Speicherabbild des Displays ist ein linearer Speicherbereich. Links oben befindet sich der erste Pixel und geht bitweise nach rechts. Am Ende der Zeile wird zum Anfang der nächsten, eins tieferen Zeile, gesprungen.

Es gibt zu jedem Zeitpunkt genau einen Adreßpointer, der auf eine bestimmte Adresse im Speicher des Displays zeigt. Dieser Pointer kann mit der Funktion `SetAddressPtr(…)` gesetzt werden.

Auf den Inhalt dieser Adresse wird die nächste Operation angewandt. Es kann entweder ein Byte mit `GetByte()` gelesen oder eines mit `SetByte(…)` geschrieben werden.

Die letzte Funktion `SetDirections(…)` legt fest, was mit dem Adreßpointer nach einer Lese- oder Schreib-Operation geschieht:

Direction-Wert	Lesen (GetByte)	Schreiben (SetByte)
`DIR_RD_NONE_WR_FASTEST`	Keine Änderung	Beliebig (je nach Controller)
`DIR_RD_NONE_WR_RIGHT`	Keine Änderung	`addr++` (eins nach rechts)
`DIR_RD_NONE_WR_DOWN`	Keine Änderung	`addr+line` (eins nach unten)
`DIR_RD_RIGHT_WR_INVALID`	`addr++` (eins nach rechts)	Ungültig, darf nicht aufgerufen werden.

Tabelle 23 – Adreßbelegung T6963

Diese vier Werte reichen aus um alle benötigten Fälle der nächsten Schicht abzudecken.

7.4. KONFIGURATION

Die Konfigurationsdatei der CTRL-Schicht ist relativ einfach aufgebaut:

```
// Select the used display controller.
// Use only _ONE_ of this list:
//
// * Toshiba T6963C (compatible with: CMG_DRV1_UNI_1BPP)
#define     CMG_CTRL_T6963
//
//
// * Sharp LH155 (compatible with: CMG_DRV1_UNI_1BPP)
//#define     CMG_CTRL_LH155
//
//
// * Epson S1D13700 (compatible with: CMG_DRV1_UNI_1BPP)
//#define     CMG_CTRL_S1D13700
// For the Epson S1D13700 you have to define the following values, too.
// You should find them in the controller's or display's datasheet.
// If graphics becomes unstable or there are streaks decrease the framerate.
// If screen flickers increase frame rate.
// The framerate should be around 60 Hz.
//#define     CMG_CTRL_CONTROLLER_CLOCK_HZ          8000000
//#define     CMG_CTRL_FRAMERATE                    60
//
//
//
// Controller config:
// ==================
// Configure the controller
#define     CMG_CTRL_WIDTH                        240
#define     CMG_CTRL_HEIGHT                       128
#define     CMG_CTRL_BITDEPTH                     1
```

Listing 49 – CMG_CTRL.config

Es muß nur der Bereich des jeweiligen Controllers auskommentiert werden. Für die Controller T6963 und LH155 sind keine weiteren Einstellungen vorzunehmen. Der Controller S1D13700 benötigt jedoch noch zusätzlich zwei Werte, die je nach verwendeter Hardware zu setzen und in den jeweiligen Datenblättern zu finden sind.

Am Ende der Datei muß nur noch die vorhandene Auflösung und Farbtiefe festgelegt werden.

7.5. T6963

Der T6963C ist ein sehr verbreiteter und universeller Controller von Toshiba[30].

Als Busprotokoll bietet er nur das 8086-Protokoll an und stellt damit nach außen die acht Datenleitungen und vier Steuerleitungen (A0, /CS, /WR, /RD) zur Verfügung.

7.5.1. ADREßBELEGUNG

Die Adreßbelegung sieht wie folgt aus:

LLIO-Adresse	CTRL-Alias Definition
CMG_LLIO_WriteA1	CMG_LLIO_WriteCommand
CMG_LLIO_WriteA0	CMG_LLIO_WriteData
CMG_LLIO_ReadA1	CMG_LLIO_ReadStatus
CMG_LLIO_ReadA0	CMG_LLIO_ReadData

Tabelle 24 – Adreßbelegung T6963

[30] Hitachi Display Product Group, Hitachi Europe Ltd, Whitebrook Park, Lower Cookham Road, Maidenhead, Berkshire, United Kingdom, http://www.hitachi-displays-eu.com/

Der T6963 ist ein statusbasierter Controller. Das bedeutet, daß vor jedem Schreiben eines Befehls sowie vor dem Schreiben und Lesen eines Datenworts der aktuelle Status abgefragt werden muß. Nur wenn dieser Status anzeigt, daß der Controller für neue Befehle oder Daten bereit ist, dürfen diese gesendet werden.

7.5.2. BEFEHLE

Der Controller kann auf unterschiedliche Arten angesprochen werden: Ein Befehl hat entweder keinen, einen oder zwei Byte-Parameter:

CTRL-Funktion	Parameter	Beschreibung
`SendCommand`		Nur ein Befehl ohne Daten
`SendCommand_1`	1 Data Byte	Befehl mit einem Byte
`SendCommand_2Word`	2 Data Bytes (Word)	Befehl mit zwei Bytes
`SendCommand_Recv`	Returns 1 Data Byte	Befehl mit einem Ergebnisbyte als Rückgabe
`WriteData_AutoMode`	1 Data Byte	Nur im Auto-Mode zum fortlaufenden Schreiben von Daten
`ReadData_AutoMode`	Returns 1 Data Byte	Nur im Auto-Mode zum fortlaufenden Lesen von Daten

Tabelle 25 – Befehle T6963

Es gibt einen sogenannten *Auto-Mode*, in den der Controller mit einem Befehl gesetzt werden kann. Der *Auto-Mode* besitzt drei Zustände: *Aus*, *Auto-Lesen*, *Auto-Schreiben*. Der Aus-Zustand ist der Normalfall. Hier können alle anderen Befehle ausgeführt werden. Befindet sich der Controller jedoch im Lesen- oder Schreiben-Zustand, so kann er ausschließlich die Befehle `WriteData_AutoMode(…)`, `ReadData_AutoMode()` und den Befehl zum Beenden des jeweiligen Modus' empfangen. Diese zwei Zustände verwendet CMG allerdings nicht, da sie sehr speziell sind. Nur die Funktion `FastClearScreen()` verwendet aus Performancegründen das sequentielle Schreiben.

In der Abbildung ist der prinzipielle Ablauf der Befehle näher erläutert:

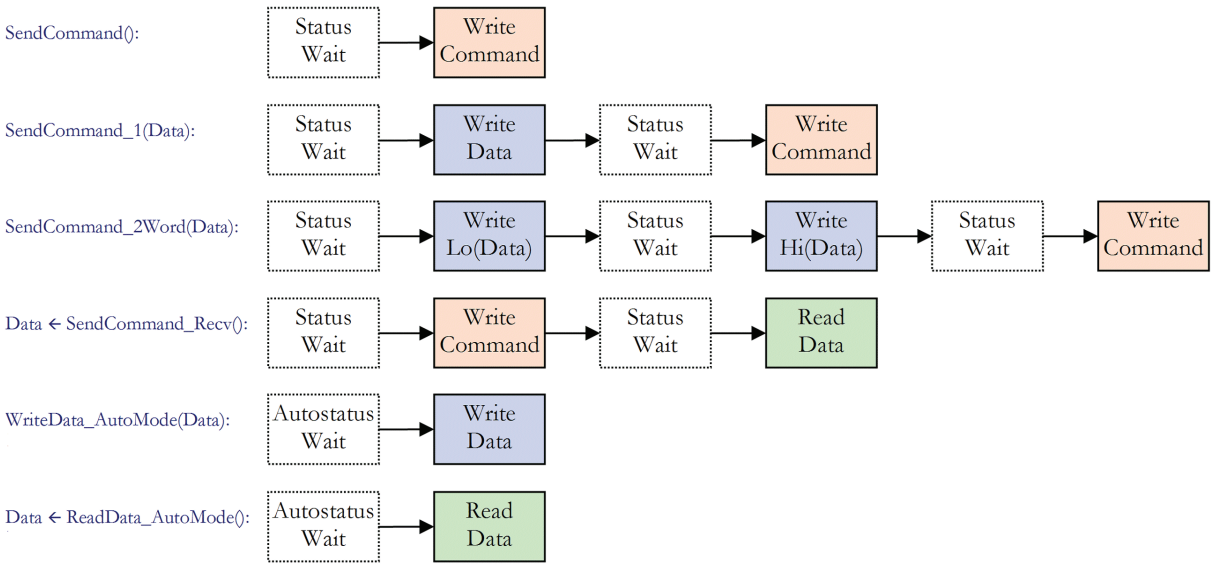


Abbildung 41 – Befehlsablauf T6963

Hier sieht man, daß die Daten **vor** dem eigentlichen Befehl übertragen werden müssen.

Der Quellcode ist analog dazu aufgebaut. Zuerst die Funktion `StatusWait()`:

```
/*-------------------------------------------------------------------------
| _CTRL_StatusWait
|-----------------------------------
| wait until lcd driver is ready
+------------------------------------------------------------------------
| @Params:          none
| @Return Value:    none
+-------------------------------------------------------------------------*/
_PRIVATE
void _CTRL_StatusWait( void )
{
    while ( ( CMG_LLIO_ReadStatus() & WAIT_COMMAND_DATA_BITS ) != WAIT_COMMAND_DATA_BITS );
}
```

Listing 50 – T6963 StatusWait

Die Funktion liest solange den Status des Displays ein, bis das Warte-Flag nicht mehr gesetzt ist und der Controller somit für neue Befehle oder Daten bereit ist.

Nachfolgend die CTRL-Befehle (aus Platzgründen ohne Kommentare):

```
_PRIVATE
void _CTRL_SendCommand( cmg_u8 byCommand )
{
    _CTRL_StatusWait();
    CMG_LLIO_WriteCommand( byCommand );
}


_PRIVATE
void _CTRL_SendCommand_1( cmg_u8 byCommand, cmg_u8 byData )
{
    _CTRL_StatusWait();
    CMG_LLIO_WriteData( byData );
    _CTRL_StatusWait();
    CMG_LLIO_WriteCommand( byCommand );
}


_PRIVATE
void _CTRL_SendCommand_2( cmg_u8 byCommand, cmg_u8 byDataLo, cmg_u8 byDataHi )
{
    _CTRL_StatusWait();
    CMG_LLIO_WriteData( byDataLo );
    _CTRL_StatusWait();
    CMG_LLIO_WriteData( byDataHi );
    _CTRL_StatusWait();
    CMG_LLIO_WriteCommand( byCommand );
}


#define _CTRL_SendCommand_2Word(cmd,wData)   _CTRL_SendCommand_2( cmd, LO_BYTE(wData), HI_BYTE(wData) )


_PRIVATE
cmg_u8 _CTRL_SendCommand_Recv( cmg_u8 byCommand )
{
    _CTRL_StatusWait();
    CMG_LLIO_WriteCommand( byCommand );
    _CTRL_StatusWait();
    return CMG_LLIO_ReadData();
}
```

Listing 51 – CTRL-Funktionen

Mit diesen Funktionen kann der Treiber nun alle Befehle des T6963-Controllers ausführen. Nachfolgend werden alle verfügbaren Befehle mit Beschreibung aufgelistet. Die Spalte V zeigt an, ob der Befehl von CMG, also der CTRL-Schicht, verwendet wird.

CMD_* Befehl	Wert	Parameter	Beschreibung	V
CURSOR_PTR	0x21	2 (X, Y addr)	Cursor-Pointer auf X, Y	
OFFSET_REG	0x22	2 (addr, 0x00)	Offset für Character Generator RAM	
ADDRESS_PTR	0x24	2 (16-Bit)	Setzt den Adreß-Pointer	X
TEXT_HOME_ADDR	0x40	2 (16-Bit)	Text-Startadresse im RAM	
TEXT_AREA	0x41	2 (col, 0x00)	Anzahl der Textzeichen pro Zeile	
GRAPHIC_HOME_ADDR	0x42	2 (16-Bit)	Grafik-Startadresse im RAM	X
GRAPHIC_AREA	0x43	2 (col, 0x00)	Anzahl der Grafik-Bytes pro Zeile	X
MODE_OR	0x80		Anzeigemodus = Grafik | Text	X
MODE_XOR	0x81		Anzeigemodus = Grafik ^ Text	

MODE_AND	0x83		Anzeigemodus = Grafik & Text	
MODE_TEXT_ATTRIB	0x84		Schaltet in den Text-Attributs-Modus	
DISPLAY_MODE_OFF	0x90		Schaltet das Display aus	X
D…_M…_GRAPHIC_ONLY	0x98		Schaltet auf „Nur Grafik“	X
…	0x9x		Andere, nicht verwendete Modi	
CURSOR_PATT…_SELECT	0xax		Höhe (Anzahl der Zeilen) für Cursor	
DATA_AUTO_WRITE	0xb0		Controller in Auto-Write-Modus	X
DATA_AUTO_READ	0xb1		Controller in Auto-Read-Modus	
DATA_AUTO_RESET	0xb2		Controller in Normal-Modus	X
DATA_WRITE_INC	0xc0	1 (Data)	Datenbyte an Adresse, Postinkrement	X
DATA_READ_INC	0xc1		Byte von Adresse lesen, Postinkrement	X
DATA_WRITE_DEC	0xc2	1 (Data)	Datenbyte an Adresse, Postdekrement	
DATA_READ_DEC	0xc3		Byte von Adresse lesen, Postdekrement	
DATA_WRITE	0xc4	1 (Data)	Datenbyte an Adresse	X
DATA_READ	0xc5		Byte von Adresse lesen	X
BIT_SETCLR	0xfx		Setzt oder löscht ein Bit an Adresse	

Tabelle 26 – Befehlssatz T6963, [T6963C]

7.5.3. CTRL-FUNKTIONALITÄT

Die globalen Variablen `iDisplayWidth` und `iDisplayHeight` definieren die Breite und Höhe des Displays in Pixel. Diese Werte sind reine Hardwarewerte und ändern sich nicht durch Skalierung, Rotation oder Spiegelung in höheren Schichten.

Eine weitere globale Variable ist `uBytesPerLine`, welche die Anzahl der Bytes je Grafikzeile angibt. Dies ist insbesondere für Operationen notwendig, die eine komplette Zeile bearbeiten sowie, um einen Adreßpointer im Speicher zeilenweise zu verschieben. Auch kann damit eine X/Y-Adresse leicht in eine Adresse umgerechnet werden:

```
Address = ( Y * uBytesPerLine ) + X
```

Die `Init()`-Funktion des CTRL-Teibers ist so aufgebaut:

```
/*************************************************************************
| CMG_CTRL_Init
|-----------------------------------
| Init (init downwards: CMG_LLIO_Init)
+------------------------------------------------------------------------
| @Params:          none
| @Return Value:    result
+************************************************************************/
_PUBLIC
cmg_Result CMG_CTRL_Init( void )
{
    cmg_Result  iResult;

    // init CMG_LLIO_*
    iResult = CMG_LLIO_Init();
    CHECKRESULT();

    // . . . . . . . . . . . . . . . . . . . . .
    // set global variables
    g_CTRL_iDisplayWidth  = CMG_CTRL_WIDTH;
    g_CTRL_iDisplayHeight = CMG_CTRL_HEIGHT;
    g_CTRL_uBytesPerLine  = GET_BYTES_FROM_BITS( g_CTRL_iDisplayWidth );

    // . . . . . . . . . . . . . . . . . . . . .
    // init display driver
    // turn display off
    _CTRL_SendCommand( CMD_DISPLAY_MODE_OFF );

    // set graphic mode details
    _CTRL_SendCommand_2Word( CMD_GRAPHIC_HOME_ADDR, 0x0000 );
    _CTRL_SendCommand_2Word( CMD_GRAPHIC_AREA, g_CTRL_uBytesPerLine );
    _CTRL_SendCommand( CMD_MODE_OR );

    // clear screen
    CMG_CTRL_FastClearScreen( 0x00 );

    // reset cursor
    CMG_CTRL_SetAddressPtr( 0x0000 );
```

```
    // turn display on in graphics mode
    _CTRL_SendCommand( CMD_DISPLAY_MODE_GRAPHIC_ONLY );


    // . . . . . .
    // init globals
    g_CTRL_iDirections = CTRL_DIR_RD_NONE_WR_FASTEST;
    g_CTRL_wCurrentScreenAddress = 0;

    // return success
    return CMG_OK;

    //.........................
    // on error free everything
_Error:
    CMG_CTRL_Exit();
    return iResult;
}
```

Listing 52 – T6963, Init-Funktion

Zuerst schaltet `Init(…)` das Display aus, setzt die notwendigen Einstellungen (nur Grafikmodus, ab Adresse 0x0000), löscht den Displayspeicher, setzt den Adreßpointer auf 0x0000 und schaltet zum Ende das Display wieder ein. Danach steht ein initialisiertes, leeres Display zur Verfügung.

`Exit()` selbst schaltet nur das Display aus und gibt danach sofort an LLIO weiter.

Das Powermanagement unterstützt das Ein- und Ausschalten des Displays.

Der T6963 besitzt allerdings keine controller-spezifischen Funktionen.

Die zu implementierende Funktion `SetAddressPtr(…)` gestaltet sich aufgrund der Controllereigenschaften als sehr einfach:

```
/*****************************************************************************
| CMG_CTRL_SetAddressPtr
|-----------------------------------
| set the address pointer
+----------------------------------------------------------------------------
| @Params:
|    * wAddress:     address pointer to set
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_CTRL_SetAddressPtr( cmg_ScrAddr wScreenAddress )
{
    _CTRL_SendCommand_2Word( CMD_ADDRESS_PTR, wScreenAddress );
    g_CTRL_wCurrentScreenAddress = wScreenAddress;
}
```

Listing 53 – T6963, SetAddressPtr

Die zu setzende Adresse wird direkt herausgeschrieben und für spätere Zwecke in einer internen, globalen Variablen gespeichert.

Mindestens genauso einfach verhält sich `SetDirections(…)`:

```
/*****************************************************************************
| CMG_CTRL_SetDirections
|-----------------------------------
| set cursor pointer directions
+----------------------------------------------------------------------------
| @Params:
|    * iDirections:      directions
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_CTRL_SetDirections( cmg_ms8 iDirections )
{
    g_CTRL_iDirections = iDirections;
}
```

Listing 54 – T6963, SetDirections

Hier speichert die Funktion ausschließlich den Wert mit den Richtungen für die `Set-` und `GetByte`-Funktionen:

```
/*****************************************************************************
| CMG_CTRL_SetByte
|-----------------------------------
| set a data byte and modify the address pointer accordingly
+-----------------------------------------------------------------------------
| @Params:
|   * byData:       data byte
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_CTRL_SetByte( cmg_u8 byData )
{
    switch ( g_CTRL_iDirections )
    {
        // ->
        case CTRL_DIR_RD_NONE_WR_FASTEST:
        case CTRL_DIR_RD_NONE_WR_RIGHT:
            _CTRL_SendCommand_1( CMD_DATA_WRITE_INC, byData );
            g_CTRL_wCurrentScreenAddress++;
            break;

        // v
        case CTRL_DIR_RD_NONE_WR_DOWN:
            _CTRL_SendCommand_1( CMD_DATA_WRITE, byData );
            g_CTRL_wCurrentScreenAddress += g_CTRL_uBytesPerLine;
            _CTRL_SendCommand_2Word( CMD_ADDRESS_PTR, g_CTRL_wCurrentScreenAddress );
            break;

        // this one is INVALID in write mode
        //case CTRL_DIR_RD_RIGHT_WR_INVALID:
        //    break;
    }
}
```

Listing 55 – T6963, SetByte

`SetByte(…)` schreibt im *Fastest-* und *Right-Modus* das Datenbyte mit dem vom Controller unterstützten Increment-Befehl und erhöht ebenfalls die lokale Kopie des Adreßpointers um eins. Im *Down-Modus* schreibt die Funktion das Datenbyte mit dem normalen Schreib-Befehl und erhöht danach die lokale Kopie des Adreßpointers um eine ganze Zeile. Nachdem der Controller das Verschieben des Pointers um eine Zeile nach unten nicht unterstützt, setzt die Funktion jetzt die lokale Kopie des Pointers in den Controller. Nur aus diesem Grund muß überhaupt lokal eine Kopie des Adreßpointers geführt werden.

Das Einlesen mit `GetByte()` gestaltet sich etwas einfacher:

```
/*****************************************************************************
| CMG_CTRL_GetByte
|-----------------------------------
| get a data byte and modify the address pointer accordingly
+-----------------------------------------------------------------------------
| @Params:          none
| @Return Value:    data byte
+*****************************************************************************/
_PUBLIC
cmg_u8 CMG_CTRL_GetByte( void )
{
    cmg_u8  byData = 0;

    switch ( g_CTRL_iDirections )
    {
        // o
        case CTRL_DIR_RD_NONE_WR_FASTEST:
        case CTRL_DIR_RD_NONE_WR_RIGHT:
        case CTRL_DIR_RD_NONE_WR_DOWN:
            byData = _CTRL_SendCommand_Recv( CMD_DATA_READ );
            break;

        // ->
        case CTRL_DIR_RD_RIGHT_WR_INVALID:
            byData = _CTRL_SendCommand_Recv( CMD_DATA_READ_INC );
            g_CTRL_wCurrentScreenAddress++;
            break;
    }
    return byData;
}
```

Listing 56 – T6963, GetByte

Im *None-Modus* muß der Adreßpointer nicht verändert werden. Dies geschieht mit dem normalen `CMD_DATA_READ`-Command. Im *Right-Modus* wird die vom Controller unterstützte Funktion aufgerufen und die lokale Kopie des Adreßpointers – wie schon beim Schreiben – um eins erhöht.

`FastClearScreen(…)` wird mit Hilfe des Auto-Write-Modes des Controllers realisiert.

7.6. LH155

Der LH155 ist ein sehr einfacher Controllertyp von Sharp[31] und wird hauptsächlich in eher kleineren LCDs eingesetzt. Die maximale Auflösung beträgt 128x64 Pixel und auch der interne Speicher von 1kB ist nicht gerade verschwenderisch.

Allerdings unterstützt der Controller sowohl ein serielles Interface, als auch ein paralleles – entweder mit 8086- oder 6800-Protokoll. Die zu verwendende Schnittstelle muß direkt an den Controllerpins eingestellt werden und ist meist schon von dem Displayhersteller fest vorgegeben.

7.6.1. ADREßBELEGUNG

Die Adreßbelegung sieht wie folgt aus:

LLIO-Adresse	CTRL-Alias Definition
`CMG_LLIO_WriteA1`	`CMG_LLIO_WriteCommand`
`CMG_LLIO_WriteA0`	`CMG_LLIO_WriteData`
`CMG_LLIO_ReadA0`	`CMG_LLIO_ReadData`

Tabelle 27 – Adreßbelegung LH155

Es fällt im Vergleich zum T6963 sofort auf, daß es keine Funktion zum Einlesen des Status' gibt. Das liegt daran, daß der Controller gar keinen Status besitzt. Es gibt keine Möglichkeit von außen zu prüfen, ob das Display bereit ist. Es sind allerdings genaue Maximalzeiten im Datenblatt angegeben, die ein Befehl dauern kann.

7.6.2. BEFEHLE

Auch die Ansteuerung des Controllers mit Befehlen ist zum T6963 gänzlich unterschiedlich:

Die Befehle besitzen keine separaten Parameter, sondern werden direkt in den Befehl integriert. Für den Befehl selbst sind vier Bit reserviert, für die Daten die restlichen vier.

Soll ein ganzes Byte übertragen werden, so wird der Befehl in zwei Unterbefehle aufgeteilt. Die dafür notwendigen Befehle sind die LOHI-Befehle.

Auch das Lesen und Schreiben von Daten gestaltet sich einfacher, weil die Daten direkt mit den LLIO-Funktionen gelesen und geschrieben werden können.

CTRL-Funktion	Parameter	Beschreibung
`SendCommand`	4-Bit Parameter	Sendet einen Befehl, verknüpft mit dem Parameter
`SendCommand_LOHI`	8-Bit Parameter	Sendet zwei Befehle, verknüpft mit den Parametern

Tabelle 28 – Befehle LH155

[31] Sharp Electronics (Europe) GmbH, Hamburg, Deutschland, http://www.sharp-world.com

Der Quellcode dazu lautet wie folgt:

```
/*-------------------------------------------------------------------------
| _CTRL_SendCommand
|-----------------------------------
| send a command + 4 bit parameter to the driver
+-------------------------------------------------------------------------
| @Params:
|    * byCommand:    command
|    * byParam:      parameter
| @Return Value:    none
+-------------------------------------------------------------------------*/
#define _CTRL_SendCommand(byCommand,byParam)
                              CMG_LLIO_WriteCommand( (cmg_u8)( (byCommand) + ( (byParam) & 0x0f ) ) )


/*-------------------------------------------------------------------------
| _CTRL_SendCommand_LOHI
|-----------------------------------
| send two commands + 4 bit parameter each to the driver
| the command must be the _LO_ command
+-------------------------------------------------------------------------
| @Params:
|    * byCommandLO:  LO command
|    * byParam:      parameter
| @Return Value:    none
+-------------------------------------------------------------------------*/
_PRIVATE
void _CTRL_SendCommand_LOHI( cmg_u8 byCommandLO, cmg_u8 byParam )
{
    // write lo command
    _CTRL_SendCommand( byCommandLO       , byParam );
    // write hi command
    _CTRL_SendCommand( byCommandLO + 0x10, byParam >> 4 );
}
```

Listing 57 – LH155 SendCommands

Mit diesen beiden Funktionen können nun alle verfügbaren Befehle ausgeführt werden:

CMD_* Befehl	Wert	Beschreibung	V
CURSOR_X_LO	0x0x	LO-Teil der X-Adresse im Display-RAM	X
CURSOR_X_HI	0x1x	HI-Teil der X-Adresse im Display-RAM	X
CURSOR_Y_LO	0x2x	LO-Teil der Y-Adresse im Display-RAM	X
CURSOR_Y_HI	0x3x	HI-Teil der Y-Adresse im Display-RAM	X
STARTLINE_LO	0x4x	LO-Teil der Startline	X
STARTLINE_HI	0x5x	HI-Teil der Startline	X
ALTERNATING_NLINE_LO	0x6x	LO-Teil der *Alternating Reverse Line*	X
ALTERNATING_NLINE_HI	0x7x	HI-Teil der *Alternating Reverse Line*	X
DISPLAY_CTRL_1	0x8x	Setzt das *Display Control Register 1*	X
DISPLAY_CTRL_2	0x9x	Setzt das *Display Control Register 2*	X
INCREMENT	0xax	Setzt den Modus zur Pointer-Veränderung bei Zugriffen	X
POWER_CTRL_1	0xbx	Setzt das *Power Control Register 1*	X
POWER_CTRL_2	0xdx	Setzt das *Power Control Register 2*	X
POWER_CTRL_3	0xex	Setzt das *Power Control Register 3*	X
RE_FLAG	0xfx	Setzt oder löscht RE-Flag (erweiterter Modus)	X

Tabelle 29 – Befehlssatz LH155, [LH155]

Dazu sind als Parameter auch einige Flags möglich:

CMD_* Flags	Wert	Beschreibung
DC1_SHIFT	0x08	Umkehrung der Grafikausgabe an den LCD-Pins
DC1_SEGON	0x04	*Segment Display* angeschalten (nicht verwendet)
DC1_ALLON	0x02	Grafischer Modus eingeschaltet
DC1_ON	0x01	Display An/Aus
DC2_REV	0x08	Invertierung des Displays
DC2_NLIN	0x04	Spezialeinstellung
DC2_SWAP	0x02	Die Bits in den *geschriebenen* Datenbytes werden vertauscht
DC2_REF	0x01	Spezialeinstellung
INC_AIM	0x04	Erhöhung bei Read *und* Write (0)oder nur bei Write (1)
INC_YI	0x02	Erhöhung des Y-Cursors

INC_XI	0x01	Erhöhung des X-Cursors
PC1_BIAS	0x08	Bias Einstellung: 1/9 oder 1/7
PC1_HALT	0x04	Energiesparmodus
PC1_PON	0x02	Grafik eingeschaltet
PC1_ACL	0x01	Display Reset ausführen
PC2_DVOL_OFF	0x0f	Vordefinierter Wert lt. Datenblatt, wenn die elektronische Kontrastspannung nicht vorhanden oder nicht verwendet wird
PC3_EXA	0x02	Icon-Anzeige Takteinstellung
PC3_ICON	0x01	Icon-Anzeige aktivieren
RE_RE	0x01	Register bekommen erweiterte, interne Einstellungen

Tabelle 30 – Flags LH155, [LH155]

7.6.3. CTRL-FUNKTIONALITÄT

Die globalen Variablen `iDisplayWidth`, `iDisplayHeight` und `uBytesPerLine` sind, wie in der Schnittstelle definiert, verwendet. Sie beschreiben die physikalische Auflösung des Displays und die Anzahl der Bytes pro Zeile und sind Hardwarewerte.

Zur Erinnerung noch einmal die Funktion zum Berechnen der Adresse aus den X/Y-Koordinaten:

```
Address = ( Y * uBytesPerLine ) + X
```

Zusätzlich gibt es noch drei weitere, für diesen Controller wichtige, globale Variablen:

- `uNeedDummyRead`: Zeigt an, ob der Controller vor dem nächsten Lesezugriff einen Dummy-Lesezugriff benötigt.
- `byCurrentCursorX`: Die aktuelle Cursor-X-Position.
- `byCurrentCursorY`: Die aktuelle Cursor-Y-Position.

Diese Variablen werden notwendig, da der Controller spezielle Eigenschaften zur Ansteuerung besetzt:

- Er benötigt vor einem Lesezugriff – also vor dem Auslesen des Datenbytes – in folgenden Situationen einen zusätzlichen Lesezugriff – den sogenannten Dummy-Lesezugriff:
  - Die Cursor-Position wurde seit dem letzten Lesezugriff neu gesetzt.
  - Es wurde seit dem letzten Lesezugriff ein Schreibzugriff ausgeführt.

  Diese Information steht in der Variablen `uNeedDummyRead`. Der erste ausgelesene Wert wird verworfen.

- Der Controller besitzt keinen linearen Adreßpointer, mit welchem man ein bestimmtes Byte – mit Hilfe der oben genannten Formel – adressieren kann. Statt dessen benötigt er dafür zwei Werte, die X- und Y-Position des Cursors. Die Y-Position gibt direkt die Zeile an, die X-Position das jeweilige Byte in der Zeile, also nicht das einzelne Pixel.

  Um nun den von der CTRL-Schicht nach außen hin benötigten, linearen Adreßpointer anzubieten, muß dieser vor jedem Setzen in die X- und Y-Werte umgerechnet werden.

Dazu sind eine Division und eine Modulo-Operation nötig:
`CursorX = Address % BytesPerLine` (mod)
`CursorY = Address / BytesPerLine` (div)

Diese beiden Operationen sind aber speziell für Mikrocontroller sehr aufwendig. Hinzu kommt, daß die Funktion zum Setzen des Adreßpointers sehr häufig aufgerufen wird.

Um diese Berechnung zu beschleunigen, greife ich zu einem kleinen Trick: Wenn die Anzahl der Bytes pro Zeile künstlich auf 256 festgelegt wird ergibt sich folgende, einfache Berechnung:
`CursorX = Address & 0x00ff` (and)
`CursorY = Address >> 8` (shift)

Die `Init()`-Funktion des CTRL-Teibers ist so aufgebaut:

```
/*****************************************************************************
| CMG_CTRL_Init
|----------------------------------
| Init (init downwards: CMG_LLIO_Init)
+-----------------------------------------------------------------------------
| @Params:          none
| @Return Value:    result
+*****************************************************************************/
_PUBLIC
cmg_Result CMG_CTRL_Init( void )
{
    cmg_Result  iResult;

    // init CMG_LLIO_*
    iResult = CMG_LLIO_Init();
    CHECKRESULT();

    // . . . . . . . . . . . . . . . . . . . . . .
    // set global variables
    g_CTRL_iDisplayWidth  = CMG_CTRL_WIDTH;
    g_CTRL_iDisplayHeight = CMG_CTRL_HEIGHT;
    g_CTRL_uBytesPerLine  = 0x0100; // set fake linesize for faster decoding
    g_CTRL_uNeedDummyRead = 1;
    g_CTRL_byCurrentCursorX = 0;
    g_CTRL_byCurrentCursorY = 0;

    // . . . . . . . . . . . . . . . . . . . . . .
    // setup display driver
    // reset
    _CTRL_SendCommand( CMD_POWER_CTRL_1, CMD_PC1_PON | CMD_PC1_ACL );
    CMG_Sleep_ms( 10 );

    // power control
    _CTRL_SendCommand( CMD_RE_FLAG, 0 );
    _CTRL_SendCommand( CMD_POWER_CTRL_1, CMD_PC1_PON );
    _CTRL_SendCommand( CMD_POWER_CTRL_2, CMD_PC2_DVOL_OFF );
    _CTRL_SendCommand( CMD_POWER_CTRL_3, 0 );

    // display control
    _CTRL_SendCommand( CMD_DISPLAY_CTRL_1, 0 );
    _CTRL_SendCommand( CMD_DISPLAY_CTRL_2, CMD_DC2_SWAP );

    // other defaults
    _CTRL_SendCommand_LOHI( CMD_STARTLINE_LOHI, 0 );
    _CTRL_SendCommand_LOHI( CMD_ALTERNATING_NLINE_LOHI, 0 );

    // clear screen
    CMG_CTRL_FastClearScreen( 0x00 );

    // turn display on in graphics mode
    _CTRL_SendCommand( CMD_DISPLAY_CTRL_1, CMD_DC1_ON );

    // . . . . . . . . . . . . . . . . . . . . . . .
    // init
    CMG_CTRL_SetDirections( CTRL_DIR_RD_NONE_WR_FASTEST );
    CMG_CTRL_SetAddressPtr( 0x0000 );

    // return success
    return CMG_OK;

    //.............................
    // on error free everything
_Error:
    CMG_CTRL_Exit();
    return iResult;
}
```

Listing 58 – LH155, Init-Funktion

Hier werden alle Initialisierungen und die notwendigen Einstellungen vorgenommen, das Display resettet, der Displayspeicher gelöscht und in den Grafikmodus geschaltet.

`Exit()` selbst schaltet nur das Display aus und gibt danach an LLIO weiter.

Das Powermanagement unterstützt das Ein- und Ausschalten des Displays.

Der LH155 besitzt, ebenfalls wie der T6963, keine controllerspezifischen Funktionen.

Der Hintergrund der Berechnungen in `SetAddressPtr(…)` wurde oben bereits beschrieben. Hier der Quellcode der Funktion:

```
/*****************************************************************************
| CMG_CTRL_SetAddressPtr
|-----------------------------------
| set the address pointer
+-----------------------------------------------------------------------------
| @Params:
|   * wAddress:     address pointer to set
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_CTRL_SetAddressPtr( cmg_ScrAddr wScreenAddress )
{
    // due the fact we faked a linesize of 256 bytes, we can easily split
    // the address to x and y
    g_CTRL_byCurrentCursorX = wScreenAddress & 0x00ff;
    g_CTRL_byCurrentCursorY = wScreenAddress >> 8;

    _CTRL_SendCommand_LOHI( CMD_CURSOR_X_LOHI, g_CTRL_byCurrentCursorX );
    _CTRL_SendCommand_LOHI( CMD_CURSOR_Y_LOHI, g_CTRL_byCurrentCursorY );

    g_CTRL_uNeedDummyRead = 1;
}
```

Listing 59 – LH155, SetAddressPtr

Nach dem Setzen der Adresse muß vor dem nächsten Lesen ein Dummy-Lesezugriff ausgeführt werden.

Das Festlegen der Lese- und Schreibrichtung sieht wie folgt aus:

```
/*****************************************************************************
| CMG_CTRL_SetDirections
|-----------------------------------
| set cursor read/write directions
+-----------------------------------------------------------------------------
| @Params:
|   * iDirections:    directions
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_CTRL_SetDirections( cmg_ms8 iDirectionSet )
{
    switch ( iDirectionSet )
    {
        case CTRL_DIR_RD_NONE_WR_FASTEST:
            _CTRL_SendCommand( CMD_INCREMENT, 0 );
            break;

        case CTRL_DIR_RD_NONE_WR_RIGHT:
            _CTRL_SendCommand( CMD_INCREMENT, CMD_INC_AIM | CMD_INC_AXI );
            break;

        case CTRL_DIR_RD_NONE_WR_DOWN:
            _CTRL_SendCommand( CMD_INCREMENT, CMD_INC_AIM | CMD_INC_AYI );
            break;

        case CTRL_DIR_RD_RIGHT_WR_INVALID:
            _CTRL_SendCommand( CMD_INCREMENT, CMD_INC_AXI );
            break;
    }

    // (re) set the last setted cursor address (can be destroyed while
    // changing direction - according to datasheet)
    _CTRL_SendCommand_LOHI( CMD_CURSOR_X_LOHI, g_CTRL_byCurrentCursorX );
    _CTRL_SendCommand_LOHI( CMD_CURSOR_Y_LOHI, g_CTRL_byCurrentCursorY );
}
```

Listing 60 – LH155, SetDirections

Auf dem LH155-Controller läßt sich das CTRL-Modell sehr schön mit dem Befehl CMD_INCREMENT – und den dafür passenden Parametern – abbilden. Allerdings muß nach dem Festlegen der Richtungen die aktuelle Cursorposition erneut gesetzt werden, da diese – laut Datenblatt [LH155] – nach dem Increment-Befehl falsche Werte annehmen kann.

Das Schreiben eines Bytes ist sehr einfach:

```
/*****************************************************************************
| CMG_CTRL_SetByte
|-----------------------------------
| set a data byte and modify the address pointer accordingly
+----------------------------------------------------------------------------
| @Params:
|    * byData:       data byte
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_CTRL_SetByte( cmg_u8 byData )
{
    // set data byte
    CMG_LLIO_WriteData( byData );
    g_CTRL_uNeedDummyRead = 1;
}
```

Listing 61 – LH155, SetByte

Auch nach einem Schreibzugriff ist ein Dummy-Lesezugriff notwendig.

Durch den Befehl `_SendCommand(DISPLAY_CTRL_2, DC2_SWAP)` in der Init-Funktion wurde festgelegt, daß beim Herausschreiben von Daten die einzelnen Bits vertauscht werden. Dieser Schritt ist notwendig, da der Controller die Bits leider andersherum abbildet, als von CMG benötigt. Der LH155 bildet nicht, wie normal, Bit 7 – also das MSB – auf den ersten Pixel ab, sondern Bit 0 – also das LSB. Dieser Umstand kann – zumindest bei Schreibzugriffen mit der oben beschriebenen Einstellung – korrigiert werden.

Das Auslesen gestaltet sich deshalb etwas komplizierter:

```
/*****************************************************************************
| CMG_CTRL_GetByte
|-----------------------------------
| get a data byte and modify the address pointer accordingly
+----------------------------------------------------------------------------
| @Params:          none
| @Return Value:    data byte
+*****************************************************************************/
_PUBLIC
cmg_u8 CMG_CTRL_GetByte( void )
{
    cmg_u8  byData, byDataSwapped, byBit;

    // need a dummy read? (after address set and write cycle)
    if ( g_CTRL_uNeedDummyRead )
    {
        // dummy read
        CMG_LLIO_ReadData();
        g_CTRL_uNeedDummyRead = 0;
    }

    // read byte
    byData = CMG_LLIO_ReadData();

    // swap byte :-(
    byDataSwapped = 0;
    for ( byBit = 0; byBit < 8; byBit++ )
    {
        byDataSwapped <<= 1;
        if ( byData & 0x01 )
            byDataSwapped++;
        byData >>= 1;
    }

    // return data byte
    return byDataSwapped;
}
```

Listing 62 – LH155, ReadByte

Falls nötig, wird hier zuerst der Dummy-Lesezugriff ausgeführt und nach dem Lesen die Bitreihenfolge im Byte vertauscht.

7.7. S1D13700

Der S1D13700 ist ein weitverbreiteter Controller von Epson[32] mit hohem Funktionsumfang für größere Displays. Die Auflösungen reichen bis 320 x 200 Pixel. Neben reinem Schwarzweiß (1bpp) unterstützt er auch noch 4 und 16 Graustufen (2bpp und 4bpp), allerdings bei der höchsten Farbtiefe nicht mehr mit voller Auflösung.

Er bietet sowohl das 8086-, als auch das 6800-Busprotokoll an.

7.7.1. ADREßBELEGUNG

Die Adreßbelegung ist zu den anderen Controllern leicht verändert:

LLIO-Adresse	CTRL-Alias Definition
`CMG_LLIO_WriteA1`	`CMG_LLIO_WriteCommand`
`CMG_LLIO_WriteA0`	`CMG_LLIO_WriteData`
`CMG_LLIO_ReadA1`	`CMG_LLIO_ReadData`

Tabelle 31 – Adreßbelegung S1D13700

Der Controller ist – wie der LH155 – ein Controller ohne Statusrückmeldung. Hier sind ebenfalls genaue Maximalzeiten definiert, welche von der Ansteuerung eingehalten werden müssen.

7.7.2. BEFEHLE

Der Controller S1D13700 besitzt zum Ausführen von Befehlen wieder eine ganz andere Strategie: Zuerst wird der Befehl an sich mit `WriteCommand(…)` übertragen; anschließend die Anzahl der nötigen Parameter mit `WriteData(…)`. Erst danach wird der Befehl ausgeführt. Je nach aktivem Befehl besitzen nachfolgende Daten-Zugriffe unterschiedliche Wirkungen.

Zuerst einmal eine Auflistung von verfügbaren und verwendeten Befehlen:

CMD_* Befehl	Wert	Params	Beschreibung	V
`SYSTEM_SET_8`	0x40	8	Systeminitialisierung und Fenstereinstellungen	X
`SLEEP_IN_0`	0x53	0	Energiesparmodus	
`DISP_OFF_1`	0x58	1	Display ausschalten	X
`DISP_ON_1`	0x59	1	Display einschalten	X
`SCROLL_10`	0x44	10	Display Startadresse und Anzeigebereich einstellen	X
`CSRFORM_2`	0x5d	2	Cursor-Form festlegen	X
`CSRDIR_RIGHT_0`	0x4c	0	Cursorrichtung für nächsten Zugriff festlegen	X
`CSRDIR_LEFT_0`	0x4d	0	Cursorrichtung für nächsten Zugriff festlegen	
`CSRDIR_UP_0`	0x4e	0	Cursorrichtung für nächsten Zugriff festlegen	
`CSRDIR_DOWN_0`	0x4f	0	Cursorrichtung für nächsten Zugriff festlegen	X
`OVLAY_1`	0x5b	1	Einstellungen für den Overlay-Modus	X
`CGRAM_ADR_2`	0x5c	2	Startadresse für Zeichenspeicher	
`HDOT_SCR_1`	0x5a	1	Horizontaler Richtungspunkt und Scrollposition	X
`GRAY_SCALE_1`	0x60	1	Farbtiefe setzen	X
`CSRW_2`	0x46	2	Cursor-Adresse setzen	X
`CSRR_2`	0x47	2	Cursor-Adresse lesen	
`MWRITE`	0x42	0	In Display-Speicher schreiben	X
`MREAD`	0x43	0	Aus Display-Speicher lesen	X

Tabelle 32 – Befehlssatz S1D13700, [S1D13700]

[32] Epson Europe Electronics GmbH, München, Deutschland, http://www.epson-electronics.de/

7.7.3. CTRL-FUNKTIONALITÄT

Die Implementierung des CMG-Interfaces gestaltet sich bei diesem Controller aufgrund der Funktionsweise etwas aufwendiger.

Der Treiber besitzt neben den normalen, für das CTRL-Interface notwendige, globale Variable, zusätzlich noch vier controllerspezifische:

```
//-----------------------------------------------------------------------------
// globals
//-----------------------------------------------------------------------------
cmg_Coord   g_CTRL_iDisplayWidth;           // display x resolution
cmg_Coord   g_CTRL_iDisplayHeight;          // display y resolution
cmg_mu16    g_CTRL_uBytesPerLine;           // bytes per line

// controler specific
cmg_ms8     g_CTRL_iDirections;             // next selected cursor dir
    #define     DIR_RIGHT                       0
    #define     DIR_DOWN                        1
cmg_ms8     g_CTRL_iCurrentDirection;       // currently selected cursor dir
cmg_ScrAddr g_CTRL_wCurrentScreenAddress;   // current address ptr
    #define     CMD_OTHER                       0
    #define     CMD_WRITE                       1
    #define     CMD_READ                        2
cmg_mu8     g_CTRL_uLastCommand;            // last command
//-----------------------------------------------------------------------------
```

Listing 63 – S1D13700, globale Variablen

`iDirections` speichert die vom Aufrufer festgelegte Richtungswahl.

`iCurrentDirection` enthält die gerade im Controller aktive Richtungswahl, da der S1D13700 nur eine gemeinsame Richtungswahl unterstützt – also nicht wie benötigt Lese- und Schreibzugriffe unterschiedlich.

`wCurrentScreenAddress` speichert – wie der Name schon vermuten läßt – den aktuellen Adreßpointer.

In `uLastCommand` wird festgehalten welcher Befehlstyp zuletzt ausgeführt wurde. Das ist später beim Lesen und Schreiben für Performanceoptimierungen sehr wichtig.

Doch zuerst zum Initialisierungscode. Hier aufgrund der Länge die gekürzte Version ohne Funktionskommentare:

```
_PRIVATE
void CMG_CTRL_Init_SystemSet( void )
{
    // system set command
    CMG_LLIO_WriteCommand( CMD_SYSTEM_SET_8 );
    CMG_LLIO_WriteData( CMG_SS_P1 | CMG_SS_P1_IV ); // default cgrom, single screen, no screen orign corr.
    CMG_LLIO_WriteData( CMG_SS_P2_WF | ( CMG_CTRL_FX - 1 ) );
    CMG_LLIO_WriteData( CMG_CTRL_FY - 1 );
    CMG_LLIO_WriteData( CMG_CTRL_CR - 1 );
    CMG_LLIO_WriteData( CMG_CTRL_TCR - 1 );
    CMG_LLIO_WriteData( CMG_CTRL_LF - 1 );
    CMG_LLIO_WriteData( CMG_CTRL_AP );
    CMG_LLIO_WriteData( 0x00 );
}

_PUBLIC
cmg_Result CMG_CTRL_Init( void )
{
    cmg_Result  iResult;

    // init CMG_LLIO_*
    iResult = CMG_LLIO_Init();
    CHECKRESULT();

    // . . . . . .
    // set global variables
    g_CTRL_iDisplayWidth  = CMG_CTRL_WIDTH;
    g_CTRL_iDisplayHeight = CMG_CTRL_HEIGHT;
    g_CTRL_uBytesPerLine  = GET_BYTES_FROM_BITS( g_CTRL_iDisplayWidth );

    // . . . . . .
    // init display driver
    // reset
    CMG_CTRL_Init_SystemSet();
    CMG_Sleep_ms(10);
```

```
    // system set
    CMG_CTRL_Init_SystemSet();

    // turn display off
    CMG_LLIO_WriteCommand( CMD_DISP_OFF_1 );
    CMG_LLIO_WriteData( CMG_DO_FC_CURSOR_OFF | CMG_DO_FC_SCREEN1_OFF | CMG_DO_FC_SCREEN2_OFF );

    // set grayscale mode
    CMG_LLIO_WriteCommand( CMD_GRAY_SCALE_1 );
    CMG_LLIO_WriteData( CMG_GS_P1_1BPP );

    // overlay set (two screens: 1: graphic, 2: graphic)
    CMG_LLIO_WriteCommand( CMD_OVLAY_1 );
    CMG_LLIO_WriteData( CMG_OL_P1_MX01_OR | CMG_OL_P1_DM1 | CMG_OL_P1_DM2 );

    // scroll - set display start and area (screen1 starts on 0x0000)
    CMG_LLIO_WriteCommand( CMD_SCROLL_10 );
    CMG_LLIO_WriteData( 0x00 );             // first screen address: 0x0000
    CMG_LLIO_WriteData( 0x00 );
    CMG_LLIO_WriteData( CMG_CTRL_LF - 1 );  // srceen line count
    CMG_LLIO_WriteData( (cmg_u8)( CMG_SCREEN_SIZE_BYTES ) );
    CMG_LLIO_WriteData( (cmg_u8)( CMG_SCREEN_SIZE_BYTES >> 8 ) );
    CMG_LLIO_WriteData( 0x00 );             // second screen has 1 dummy line
    CMG_LLIO_WriteData( 0x00 );             // unused
    CMG_LLIO_WriteData( 0x00 );             // unused
    CMG_LLIO_WriteData( 0x00 );             // unused
    CMG_LLIO_WriteData( 0x00 );             // unused

    // cursor form (8x8)
    CMG_LLIO_WriteCommand( CMD_CSRFORM_2 );
    CMG_LLIO_WriteData( 0x00 );             // cusor x size == 1
    CMG_LLIO_WriteData( 0x00 );             // cusor y size == 1

    // horizontal dot scrolling (disable)
    CMG_LLIO_WriteCommand( CMD_HDOT_SCR_1 );
    CMG_LLIO_WriteData( 0x00 );             // no scrolling

    // clear screen
    CMG_CTRL_FastClearScreen( 0x00 );

    // reset cursor
    CMG_LLIO_WriteCommand( CMD_CSRDIR_RIGHT_0 );
    CMG_CTRL_SetAddressPtr( 0x0000 );

    // turn display on (screen 1 only)
    CMG_LLIO_WriteCommand( CMD_DISP_ON_1 );
    CMG_LLIO_WriteData( CMG_DO_FC_CURSOR_OFF | CMG_DO_FC_SCREEN1_ON | CMG_DO_FC_SCREEN2_OFF );

    // . . . . . .
    // init globals
    g_CTRL_iDirections = CTRL_DIR_RD_NONE_WR_FASTEST;
    g_CTRL_wCurrentScreenAddress = 0;
    g_CTRL_iCurrentDirection = DIR_RIGHT;
    g_CTRL_uLastCommand = CMD_OTHER;

    // return success
    return CMG_OK;

    //.........................
    // on error free everything
_Error:
    CMG_CTRL_Exit();
    return iResult;
}
```

Listing 64 – S1D13700, Init

Die Funktionen zum Initialisieren des Systems sind in `Init_SystemSet(…)` ausgelagert, da sie zweimal hintereinander aufgerufen werden sollen.

`Exit()` ist dafür wieder einfacher. Es schaltet nur das Display aus und gibt danach sofort an LLIO weiter.

Das Powermanagement unterstützt das Ein- und Ausschalten des Displays.

Der S1D13700 besitzt ebenfalls keine controllerspezifischen Funktionen.

`SetAddressPtr(…)` und `SetDirections(…)` sind auch eher kurz, da sie die Informationen nur zwischenspeichern:

```
/*************************************************************************
| CMG_CTRL_SetAddressPtr
|-----------------------------------
| set the address pointer
+-------------------------------------------------------------------------
| @Params:
|   * wAddress:     address pointer to set
| @Return Value:    none
+*************************************************************************/
_PUBLIC
void CMG_CTRL_SetAddressPtr( cmg_ScrAddr wScreenAddress )
{
    // send set cursor command
    CMG_LLIO_WriteCommand( CMD_CSRW_2 );
    CMG_LLIO_WriteData( (cmg_u8)( wScreenAddress & 0x00ff ) );
    CMG_LLIO_WriteData( (cmg_u8)( wScreenAddress >> 8 ) );

    // store current cursor
    g_CTRL_wCurrentScreenAddress = wScreenAddress;
    g_CTRL_uLastCommand = CMD_OTHER;
}


/*************************************************************************
| CMG_CTRL_SetDirections
|-----------------------------------
| set cursor pointer directions
+-------------------------------------------------------------------------
| @Params:
|   * iDirections:      directions
| @Return Value:    none
+*************************************************************************/
_PUBLIC
void CMG_CTRL_SetDirections( cmg_ms8 iDirections )
{
    // just store next direction
    g_CTRL_iDirections = iDirections;
}
```

Listing 65 – S1D13700, SetAddressPtr, SetDirections

In `SetByte(…)` und `GetByte(…)` steckt die ganze Funktionalität:

```
/*************************************************************************
| CMG_CTRL_SetByte
|-----------------------------------
| set a data byte and modify the address pointer accordingly
+-------------------------------------------------------------------------
| @Params:
|   * byData:       data byte
| @Return Value:    none
+*************************************************************************/
_PUBLIC
void CMG_CTRL_SetByte( cmg_u8 byData )
{
    // check for a direction change
    switch ( g_CTRL_iDirections )
    {
        // -> (right)
        case CTRL_DIR_RD_NONE_WR_RIGHT:
            // if current direction is NOT right
            if ( g_CTRL_iCurrentDirection != DIR_RIGHT )
            {
                // set right direction
                CMG_LLIO_WriteCommand( CMD_CSRDIR_RIGHT_0 );
                g_CTRL_iCurrentDirection = DIR_RIGHT;
                g_CTRL_uLastCommand = CMD_OTHER;
            }
            // adjust cursor
            g_CTRL_wCurrentScreenAddress++;
            break;

        // v (down)
        case CTRL_DIR_RD_NONE_WR_DOWN:
            // if current direction is NOT down
            if ( g_CTRL_iCurrentDirection != DIR_DOWN )
            {
                // set down direction
                CMG_LLIO_WriteCommand( CMD_CSRDIR_DOWN_0 );
                g_CTRL_iCurrentDirection = DIR_DOWN;
                g_CTRL_uLastCommand = CMD_OTHER;
            }
            // adjust cursor
            g_CTRL_wCurrentScreenAddress += g_CTRL_uBytesPerLine;
            break;

        // if *WR_FASTEST it doesn't matter in wich mode we are...
        //case CTRL_DIR_RD_NONE_WR_FASTEST:
        //    break;

        // this one is INVALID in write mode
        //case CTRL_DIR_RD_RIGHT_WR_INVALID:
        //    break;
```

```
    }

    // if the last command was a write we do not need to send it again
    if ( g_CTRL_uLastCommand != CMD_WRITE )
        CMG_LLIO_WriteCommand( CMD_MWRITE );

    // write data byte
    CMG_LLIO_WriteData( byData );

    // set that last command was a write
    g_CTRL_uLastCommand = CMD_WRITE;
}
```

Listing 66 – S1D1370, SetByte

`SetByte(…)` prüft zuerst die gewünschte Richtung für Schreibzugriffe. Für *Fastest* und *Invalid* werden keine Anpassungen oder Änderungen vorgenommen. Bei *Right* oder *Down* wird jedoch die aktuelle, im Controller gewählte Richtung überprüft. Stimmt diese nicht mit der Zielrichtung überein, wird sie neu gesetzt und der letzte Befehlstyp auf *Other* gesetzt. Anschließend muß noch die lokale Kopie des Adreßpointers aktualisiert werden.

Danach folgt der eigentliche Schreibzugriff. Dafür wird zuerst geprüft ob der letzte Befehl der *Write*-Befehl war. Ansonsten muß die Funktion den Controller in diesen Modus versetzen. Danach wird endlich das entsprechende Byte geschrieben.

Zum Abschluß wird der *Write*-Befehl als letzter Befehl gespeichert.

`GetByte(…)` ist auch nur unwesentlich kürzer:

```
/*****************************************************************************
| CMG_CTRL_GetByte
|-----------------------------------
| get a data byte and modify the address pointer accordingly
+----------------------------------------------------------------------------
| @Params:          none
| @Return Value:    data byte
+*****************************************************************************/
_PUBLIC
cmg_u8 CMG_CTRL_GetByte( void )
{
    cmg_u8  byData = 0;

    // PRE-check for a direction change
    switch ( g_CTRL_iDirections )
    {
        // -> (right)
        case CTRL_DIR_RD_RIGHT_WR_INVALID:
            // if current direction is NOT right
            if ( g_CTRL_iCurrentDirection != DIR_RIGHT )
            {
                // set right direction
                CMG_LLIO_WriteCommand( CMD_CSRDIR_RIGHT_0 );
                g_CTRL_iCurrentDirection = DIR_RIGHT;
                g_CTRL_uLastCommand = CMD_OTHER;
            }
            // adjust cursor
            g_CTRL_wCurrentScreenAddress++;
            break;

        // on none direction we have to set back the cursor afterwards
        // because the controller doesn't support a none command :-(
        //case CTRL_DIR_RD_NONE_WR_FASTEST:
        //case CTRL_DIR_RD_NONE_WR_RIGHT:
        //case CTRL_DIR_RD_NONE_WR_DOWN:
    }

    // if the last command was a read we do not need to send it again
    if ( g_CTRL_uLastCommand != CMD_READ )
        CMG_LLIO_WriteCommand( CMD_MREAD );

    // read data byte
    byData = CMG_LLIO_ReadData();

    // set that last command was a read
    g_CTRL_uLastCommand = CMD_READ;

    // POST-check for a direction change
    switch ( g_CTRL_iDirections )
    {
        // o (none)
        case CTRL_DIR_RD_NONE_WR_FASTEST:
        case CTRL_DIR_RD_NONE_WR_RIGHT:
        case CTRL_DIR_RD_NONE_WR_DOWN:
            // on every none direction we must set back the cursor afterwards
            CMG_LLIO_WriteCommand( CMD_CSRW_2 );
            CMG_LLIO_WriteData( (cmg_u8)( g_CTRL_wCurrentScreenAddress ) );
            CMG_LLIO_WriteData( (cmg_u8)( g_CTRL_wCurrentScreenAddress >> 8 ) );
```

```
            g_CTRL_uLastCommand = CMD_OTHER;
            break;
    }

    // return data
    return byData;
}
```

Listing 67 – S1D13700, GetByte

Auch hier wird zuerst die gewünschte Zielrichtung überprüft, allerdings ist vor dem Lesen nur *Right* von Bedeutung: Wie beim Schreiben wird bei *Right* geprüft, welche Richtung bereits gesetzt ist und diese dann entsprechend angepaßt.

Danach gibt die Funktion den *Read*-Befehl aus, wenn sich der Controller noch nicht darin befindet. Jetzt kann das Byte eingelesen werden.

Nach dem Einlesen müssen alle *None*-Fälle vom Lesen – also die restlichen Drei – abgehandelt werden. Der Controller besitzt leider keine Möglichkeit den Cursor nach einem Zugriff nicht zu verändern. Die CTRL-Schnittstelle verlangt aber im *None*-Zustand, daß der Cursor nach einem Read-Zugriff nicht verändert wird. Zur Lösung des Problems setzt die Funktion einfach den alten Adreßpointer-Wert vor dem Lesezugriff zurück.

8. DRV – Driver

8.1. Ziele und Funktionsweise

Die Graphic-Driver-Schicht DRV abstrahiert das CTRL-Interface – vom Zugriff auf einen linearen Speicherbereich – zu einem Interface der grundlegendsten Grafikfunktionen.

Folgende Ziele sind dabei zu verwirklichen:

- Unabhängigkeit von der verwendeten Farbtiefe (1bpp, 2bpp, 4bpp, Farbe, etc…)
- Zeichen mit wahlweise einer festen Farbe oder mit einem benutzerdefinierten Stift (ähnlich einer 1-dimensionalen Textur)
- Auswahl der Verknüpfungsart vom Hintergrund mit dem zu zeichnenden Bereich

Im Moment soll diese grobe Übersicht reichen, die genaue Beschreibung dieser Punkte folgt weiter unten in diesem Abschnitt.

Ich habe lange über folgende Frage nachgedacht: „Was sind eigentlich die grundlegendsten Grafikfunktionen?“. Zuerst kamen Ideen wie: Linien, Rechtecke, Kreise, usw. Die Lösung ist aber wesentlich einfacher: Wenn man das Problem auf die oben genannten drei Punkte reduziert, ist die grundlegendste Funktion das Zeichnen eines Pixels!

Es geht in dieser Schicht also nicht darum, erweiterte Figuren zu zeichnen. Das Zeichnen einer Linie, eines Rechtecks oder eines Kreises ist unabhängig von der verwendeten Farbtiefe, der aktuellen Textur und der Verknüpfungsart, wenn man die Aufgabe auf die Pixelebene reduziert.

Jetzt stehen wir aber vor einem Problem: Einerseits soll nur eine Pixelfunktion zur Verfügung stehen, andererseits soll das Zeichnen aber möglichst performant sein. Daraus resultiert die Frage: „Gibt es weitere, elementare Grafikfunktionen, die sich für das Hardwaremodell besonders gut optimieren lassen?“. Wenn man das Füllen von Bereichen genauer betrachtet, kommt man auf die Lösung: Horizontale und vertikale Linien. Diese liegen, nach dem linearen Speichermodell unserer CTRL-Schicht, direkt nacheinander im Speicher. Das bedeutet bei einer Farbtiefe von 1bpp einen maximalen Performancegewinn vom Faktor 8, da in einer Operation acht nebeneinander liegende Pixel verarbeitet werden können.

Bis jetzt klingt die Aufgabe dieser Schicht noch eher einfach, doch der nächste Abschnitt wird zeigen, daß die Schicht komplizierter wird als vielleicht zuerst angenommen. Um den Aufbau dieser Schicht, ihren Quelltext und die Funktionsweise von CMG besser verstehen zu können, möchte ich die oben genannten drei Punkte noch einmal genauer erläutern:

8.1.1. Unabhängigkeit von der Farbtiefe

Je nach der zu verwendenden Farbtiefe muß das passende DRV-Modul ausgewählt werden. Zur Zeit ist nur das DRV-Modul für die Farbtiefe 1bpp – also schwarz/weiß – implementiert, weitere lassen sich aber nach dem gleichen Schema einfach hinzufügen.

Der einzige Unterschied bei verschiedenen Farbtiefen ist der pixelweise Zugriff und die damit beeinflußten Bits. Das bedeutet: Je nach Farbtiefe ändert sich das Offset eines Pixels und die Länge der Bits für eben dieses Pixel.

Zu beachten ist weiter, daß die Farbtiefe des Displays und die der verwendeten Stifte (Texturen) gleich sein muß. Sollte die Textur in einer anderen Tiefe vorliegen, so muß sie zuvor von einer höheren Schicht konvertiert werden.

8.1.2. Farben und Stifte (Colors und Pens)

Das CMG-Modell und damit auch das dieser Schicht ist statusbasierend. Das bedeutet, daß zum Zeichen selbst nur die Koordinaten angegeben werden. Die Farben, Füllungen, Verknüpfungsarten, also alle notwendigen Nebeninformationen zum Zeichnen, müssen zuvor über die dafür vorgesehenen Funktionen gesetzt werden und behalten auch nach dem Funktionsaufruf ihre Gültigkeit.

Zum Festlegen der zu verwendenden Farbe für das Zeichnen von Pixeln und geraden Linien gibt es nun zwei Möglichkeiten:

- **Feste Farben (Solid Color):**
  Wird eine feste Farbe gesetzt, so werden alle darauffolgenden Pixel mit genau dieser Farbe gezeichnet.

- **Stifte (Pen):**
  Ein Stift ist vergleichbar mit einer eindimensionalen Textur. Vor jedem Pixel, der gezeichnet wird, ermittelt eine Funktion den nächsten Farbwert. Dieser ist die aktuelle Position im Stift. Danach wird die Position um eins erhöht.

  Zudem gibt es eine Möglichkeit diese aktuelle Position zu verändern: Entweder relativ um eine Anzahl von Pixeln oder absolut, also bezogen auf ihren Ursprung. Diese Aktion wird im Folgenden *Spooling* genannt.

  Um die Sache gleich richtig mächtig und kompliziert zu gestalten, kann ein Stift zusätzlich noch beliebig skaliert werden. Damit kann die Textur stufenlos vergrößert bzw. verkleinert werden.

8.1.3. Verknüpfungsarten (ROP)

Die Verknüpfungsart beschreibt die Kombination des zu zeichnenden Pixels mit dem bestehenden Hintergrund. Diese Verknüpfungsart nennt sich ROP (*engl.: rasterization operation*).

Bis jetzt sind in CMG folgende ROPs definiert:

- **COPY**: Der Hintergrundwert ist egal, und die zu setzende Farbe wird ohne Veränderung einfach gesetzt. Dies ist der normale Modus.

- **XOR**: Der Hintergrundwert und die zu setzende Farbe werden mit dem XOR-Operator verknüpft. Wird als Farbe der feste Wert 1 verwendet, damit also jeder Pixelwert mit XOR verknüpft, so findet eine Invertierung statt. Wenn auf einen Bereich eine Invertierung angewendet wird, ist auch jeder betroffene Pixelwert verändert; die

Änderung ist somit auf jeden Fall sichtbar. Wird die gleiche Operation auf den gleichen Bereich erneut angewendet, ergibt sich wieder der ursprüngliche Bereich; die Änderung ist somit vollständig rückgängig gemacht worden.

- **AND**: Hier ergibt sich die Zielfarbe aus der logischen Und-Verknüpfung zwischen Hintergrund und neuem Farbwert.

- **OR**: Hier ergibt sich die Zielfarbe aus der logischen Oder-Verknüpfung zwischen Hintergrund und neuem Farbwert.

8.2. AUFBAU

Die Sourcen und die Konfigurationsdatei liegen in `CMG/3_DRV/`.

Zuerst wieder die Übersicht des Aufbaus dieser Schicht:

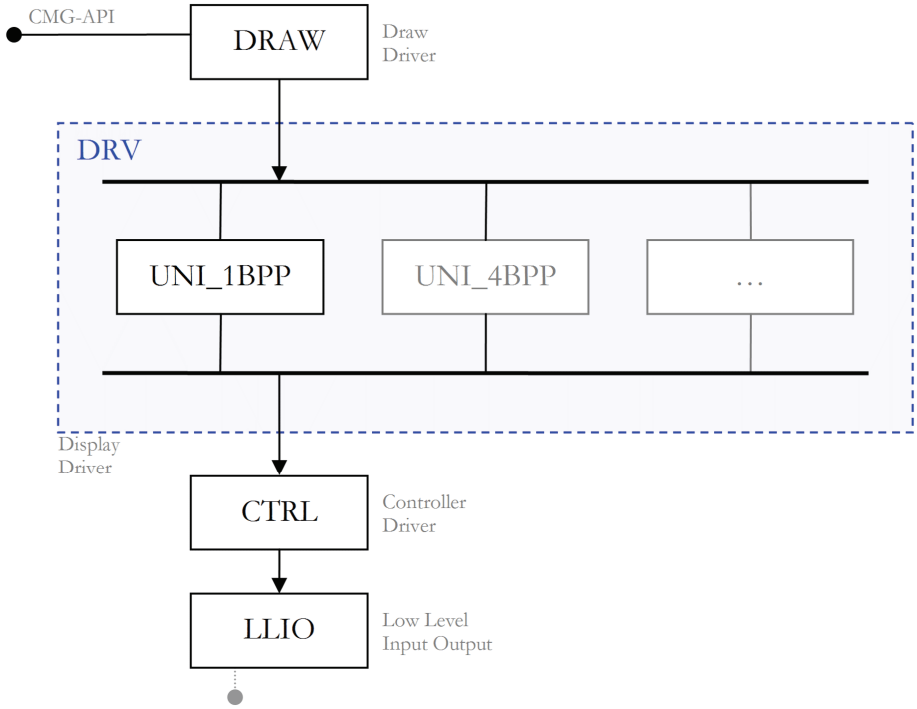


Abbildung 42 – DRV Aufbau

Die DRV-Schicht hängt unter der DRAW-Schicht, der API von CMG für den direkten Modus und greift wiederum ausschließlich auf die darunterliegende CTRL-Schicht zu. Innerhalb der Schicht stehen verschiedene, gleichwertige Module zur Verfügung, die die jeweilige Farbtiefe unterstützen.

DRV abstrahiert den Zugriff des linearen Displayspeichers aus CTRL zu einem koordinatenbezogenen, pixelweisen Zugriff.

8.3. SCHNITTSTELLEN

Zuerst, wie gewohnt, die komplette Schnittstellendefinition:

```
// ==================
// ==== Init/Exit ====
// ==================

    //-- Init/Exit ------------------------------------------------------------
        cmg_Result  CMG_DRV_Init( void );
        void        CMG_DRV_Exit( void );
        void        CMG_DRV_PowerManagement( cmg_mu8 uState );
        cmg_Result  CMG_DRV_ControllerSpecific( cmg_mu8 uFunction, cmg_void *pParam );
    //-----------------------------------------------------------------------


// ===========================
// ==== BPP Calculations =====
// ===========================

    //-- byte/bit calculations ------------------------------------------------
        #define     CMG_DRV_BIT_FROM_X(x)           ( (cmg_u8) ( ( ~(x) ) & 0x07 ) )
        #define     CMG_DRV_BYTE_FROM_X(x)          ( (x) >> 3 )
        #define     CMG_DRV_FULL_BYTES_FROM_X(x)    ( GET_BYTES_FROM_BITS( x ) )
    //-----------------------------------------------------------------------


// ===========================
// ==== Mode Management =====
// ===========================

    //-- ROP ------------------------------------------------------------------
            #define     DRV_ROPMODE_COPY                0x00
            #define     DRV_ROPMODE_XOR                 0x01
            #define     DRV_ROPMODE_AND                 0x02
            #define     DRV_ROPMODE_OR                  0x03
        void        CMG_DRV_SetROPMode( cmg_mu8 uROPMode );
    //-----------------------------------------------------------------------

    //-- Colors / Pens --------------------------------------------------------
        void        CMG_DRV_SetSolidColor( cmg_Color SolidColor );
            #define     DRV_SCALE_NONE                  ( INT_TO_SFP( 1 ) )
        void        CMG_DRV_SetPen( cmg_u8 *pbyPenStartAddr, cmg_mu16 uPenLengthBytes, cmg_sfp1616
                        sfpScaleFactor );
        void        CMG_DRV_SpoolPen( cmg_Coord iPixels );
        void        CMG_DRV_SpoolPenRelative( cmg_Coord iPixels );
        void        CMG_DRV_PenState_SaveRestore( cmg_DRV_PenState *pPenState_SaveTo, cmg_DRV_PenState
                        *pPenState_RestoreFrom );
    //-----------------------------------------------------------------------


// ==================
// ==== Drawing =====
// ==================

    //-- Output ---------------------------------------------------------------
        void        CMG_DRV_Pixel( cmg_Coord iX, cmg_Coord iY );
        void        CMG_DRV_HLine( cmg_Coord iX, cmg_Coord iY, cmg_Coord iLength );
        void        CMG_DRV_VLine( cmg_Coord iX, cmg_Coord iY, cmg_Coord iLength );
    //-----------------------------------------------------------------------

    //-- Input ----------------------------------------------------------------
        void        CMG_DRV_GetHLine( cmg_u8 *pDestBuffer, cmg_Coord iX, cmg_Coord iY, cmg_Coord iLength );
    //-----------------------------------------------------------------------
```

Listing 68 – CMG_DRV.h, Teil 1

Die Funktionen `Init()`, `Exit()`, `PowerManagement(…)` und `ControllerSpecific(…)` sind nach dem gewohnten Prinzip aufgebaut und bedürfen keiner genaueren Beschreibung.

Die folgenden drei Makros dienen zur Berechnung von Bit- und Byte-Offsets.

Der Rest besteht aus den zwei wichtigen Teilen: Dem Mode-Management – zum Setzen des aktuellen Status' – und dem Zeichenbereich (Drawing), welcher für das eigentliche Zeichnen zuständig ist.

Das Mode-Management teilt sich in den Bereich der Verknüpfungsart, dem ROP-Mode und dem Farbmodus auf. Beim Umgang mit Stiften kann der aktuelle Stand gesichert und geladen werden. Diese Aufgabe übernimmt die Funktion `PenState_SaveRestore(…)` und erwartet dafür den benutzerdefinierten Typ `PenState`:

```
typedef struct _cmg_DRV_PenState
{
    cmg_bool    g_DRV_bPenScaled;           // if scaled pen is used
    union {
        cmg_u8      g_DRV_byPenCurrentBit;      // current pen bit
        cmg_sfp1616 g_DRV_sfpPenCurrentBitPos;  // current fp pen bit
    };
} cmg_DRV_PenState;
```

Listing 69 – CMG_DRV.h, Teil 2

Diese Struktur beinhaltet nur die aktuelle Position innerhalb eines Stiftes, entweder für normale Stifte als Bitwert oder für skalierte Stifte als Festkommawert.

Der Drawing-Bereich besitzt die beschriebenen Funktionen zum Zeichen von Pixeln und geraden Linien sowie das Einlesen von Bildschirmbereichen.

8.4. Konfiguration

Die Konfiguration der DRV-Schicht ist die einfachste von CMG:

```
//############################################################################
// (3) CMG_DRV - CONFIG
//############################################################################
//
// CMG_DRV_*
// =====================
// Display driver configuration.
//
//
// For internal correct compilation.
// *** DO NOT CHANGE! ***
#define     CMG_DRV_CONFIGURED
//
//
//============================================================================
// User Area:
// ==========
// Read the sections carefully and change the values accordingly to
// your needs.
//
//
// Select one DRV interface.
// Check compatiblility with CTRL module (see CTRL module).
// Use only _ONE_ of this list:
//
// * univeral 1bpp
#define     CMG_DRV_UNI_1BPP
//
//
//############################################################################
//############################################################################
```

Listing 70 – CMG_DRV.config

Es muß nur das zu verwendende DRV-Modul ausgewählt werden – im Moment nicht allzu schwierig, da es zur Zeit nur UNI_1BPP gibt.

8.5. Schwarz/Weiß – 1-Bit per Pixel

Wie schon in der Schnittstellendefinition beschrieben, arbeiten die Init-/Exit-Funktionen wie gehabt. `Init(…)` setzt die Standardwerte für den Status: Den *ROP*-Modus auf *COPY* und die Farbe auf den konstanten Wert 0xff.

DRV definiert intern ein Makro zum Umrechnen einer X/Y-Koordinate in eine lineare Speicheradresse für die CTRL-Schicht:

```
#define     ADDR_FROM_XY(x,y)       ( (cmg_ScrAddr) ( ( (y) * g_CTRL_uBytesPerLine ) + CMG_DRV_BYTE_FROM_X(
                                    x ) ) )
```

Listing 71 – CMG_DRV_UNI_1BPP.c, ADDR_FROM_XY

Das Offset berechnet sich sehr einfach nach folgendem Schema: Zuerst wird der Zeilenanfang berechnet: Es ergibt sich aus der aktuellen Zeile *y*, multipliziert mit der Anzahl an Bytes pro Zeile. Zu diesem Wert wird noch der X-Offset addiert. Der X-Offset gibt das für das gewählte Pixel zuständige Byte zurück: Im Falle von 1bpp *(x >> 3)*.

Das in diesem Byte zuständige Bit für den gewählten X-Wert muß zusätzlich noch mit dem Makro `BIT_FROM_X(…)` berechnet werden. Hierbei ist zu beachten, daß Pixel 0 bei Bit 7 anfängt, bis schließlich Pixel 7 bei Bit 0 aufhört. Um diese Operation effizient zu berechnen, eignet sich folgender Aufruf: *(~x & 0x07)*.

Da der Zeichenmodus statusbasiert ist, muß der aktuelle Status auch irgendwo gespeichert werden. Dies geschieht in diesen globalen Variablen, wieder nach Bereichen aufgeteilt:

```
    //-- Modes ---------------------------------------------------------------
        cmg_mu8     g_DRV_uROPMode;             // current rasterization operation mode
    //------------------------------------------------------------------------

    //-- Color ---------------------------------------------------------------
            #define     DRV_COLORMODE_SOLIDCOLOR        0x00
            #define     DRV_COLORMODE_PEN               0x01
        cmg_mu8     g_DRV_uColorMode;           // current color mode (solid - pen)

        // solid color
        cmg_Color   g_DRV_SolidColor;           // current solid color

        // pen
        cmg_u8     *g_DRV_pbyPenStartAddr;      // pen start ptr
        cmg_u8     *g_DRV_pbyPenStopAddr;       // pen stop ptr
        cmg_u8     *g_DRV_pbyPenCurrentAddr;    // current pen ptr
        cmg_u8      g_DRV_byPenCurrentBit;      // current pen bit

        // scaled pen
        cmg_bool    g_DRV_bPenScaled;           // if scaled pen is used
        cmg_sfp1616 g_DRV_sfpPenScaleFactor;    // pen scale factor
        cmg_sfp1616 g_DRV_sfpPenCurrentBitPos;  // current fp pen bit
    //------------------------------------------------------------------------
```

Listing 72 – CMG_DRV_UNI_1BPP.c, globale Variablen

Der ROP-Modus benötigt nur eine Variable – der aktuelle Farb-Modus jedoch mehrere:

In `uColorMode` steht, ob aktuell eine konstante Farbe oder ein Stift verwendet wird.

Wird eine Farbe verwendet, steht deren Wert in `SolidColor`. Wird ein Stift verwendet, benötigt das Modul die Startadresse, die Stopadresse und die aktuelle Adresse innerhalb dieser Schranken. Zusätzlich wird innerhalb dieser Adresse, also dem aktuellen Byte, noch das aktuelle Bit gespeichert. `bPenScaled` gibt an, ob der Stift skaliert ist und wenn ja, stehen die dafür notwendigen Details in den letzten beiden Variablen. Sollte der Stift skaliert sein, steht die aktuelle Position im Festkommawert `sfpPenCurrentBitPos` und `byPenCurrentBit` ist ungültig.

8.5.1. Mode-Management

Beim Setzen des ROP-Modes von außen mit `SetROPMode(…)` wird der übergebene Wert lediglich in die lokale Variable kopiert.

Zum Setzen einer konstanten Farbe ist diese Funktion zuständig:

```
/*****************************************************************************
| CMG_DRV_SetSolidColor
|-----------------------------------
| Set draw mode to pen and initializes pen
+----------------------------------------------------------------------------
| @Params:
|   * SolidColor:       color to select
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_DRV_SetSolidColor( cmg_Color SolidColor )
{
    // set color mode
    g_DRV_uColorMode = DRV_COLORMODE_SOLIDCOLOR;
    // set solid color
    g_DRV_SolidColor = SolidColor;
}
```

Listing 73 – CMG_DRV_UNI_1BPP.c, SetSolidColor

Auch hier wird nur die gewählte Farbe in der globalen Variable gespeichert und der Modus auf `SOLIDCOLOR` gesetzt.

Soll ein Stift gesetzt werden, gestaltet sich die Aufgabe schon etwas umfangreicher:

```
/*****************************************************************************
| CMG_DRV_SetPen
|-----------------------------------
| Set draw mode to scaled pen and initializes pen
+----------------------------------------------------------------------------
| @Params:
|   * pbyPenStartAddr:  pen start address
|   * uPenLengthBytes:  pen length in bytes
|   * sfpScaleFactor:   pen scale or DRV_SCALE_NONE
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_DRV_SetPen( cmg_u8 *pbyPenStartAddr, cmg_mu16 uPenLengthBytes, cmg_sfp1616 sfpScaleFactor )
{
    // set color mode
    g_DRV_uColorMode       = DRV_COLORMODE_PEN;

    // set pen
    g_DRV_pbyPenStartAddr   = pbyPenStartAddr;
    g_DRV_pbyPenStopAddr    = pbyPenStartAddr + uPenLengthBytes;
    g_DRV_pbyPenCurrentAddr = pbyPenStartAddr;

    if ( sfpScaleFactor == DRV_SCALE_NONE )
    {
        // set normal pen
        g_DRV_byPenCurrentBit   = 7;
        g_DRV_bPenScaled        = false;
    }
    else
    {
        // set scaled pen
        g_DRV_bPenScaled            = true;
        g_DRV_sfpPenScaleFactor     = sfpScaleFactor;
        g_DRV_sfpPenCurrentBitPos   = INT_TO_SFP( 8 ) - 1;
    }
}
```

Listing 74 – CMG_DRV_UNI_1BPP.c, SetPen

Zu Beginn wird der Modus auf `PEN` gesetzt. Es werden nur die Startadresse und die Länge des Stiftes übergeben, da die Stopadresse redundant ist und nur aus Performancegründen berechnet und zwischengespeichert wird. Falls der Stift nicht skaliert werden soll, wird der erste Pixel – Sie erinnern sich – mit Bit 7 darin markiert. Bei einem skalierten Stift speichert die Funktion den Faktor ab und setzt die Position auf den kleinstmöglichen Wert unter 8.0.

Richtig kompliziert wird es erst beim Verändern der aktuellen Position in einem Stift:

```
/*****************************************************************************
| CMG_DRV_PenSpool
|-----------------------------------
| spool pen position (absolute)
+----------------------------------------------------------------------------
| @Params:
|   * iPixels:      number of pixels
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_DRV_SpoolPen( cmg_Coord iPixels )
{
    // . . . . . . . . . . . . . . . . . . . . . .
    // reset bit position

    // reset addr
    g_DRV_pbyPenCurrentAddr = g_DRV_pbyPenStartAddr;

    // reset bit pos
    if ( !g_DRV_bPenScaled )
        g_DRV_byPenCurrentBit   = 7;
    else
        g_DRV_sfpPenCurrentBitPos   = INT_TO_SFP( 8 ) - 1;


    // spool to position
    CMG_DRV_SpoolPenRelative( iPixels );
}
```

Listing 75 – CMG_DRV_UNI_1BPP.c, SpoolPen

Die Variante für die absolute Positionierung setzt zuerst die Position komplett zurück und ruft anschließend die relative Variante auf:

```
/*****************************************************************************
| CMG_DRV_SpoolPenRelative
|-----------------------------------
| spool pen position forward or reward
+----------------------------------------------------------------------------
| @Params:
|   * iPixels:      number of pixels
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_DRV_SpoolPenRelative( cmg_Coord iPixels )
{
    cmg_mu16    uBytes;

    // . . . . . . . . . . . . . . . . . . . . . . .
    // spool forward
    if ( iPixels > 0 )
    {
        if ( !g_DRV_bPenScaled )
        {
            // normal pen

            // get bits and bytes to spool
            uBytes = CMG_DRV_BYTE_FROM_X( iPixels );

            // calc bit
            g_DRV_byPenCurrentBit -= ( iPixels * CMG_CTRL_BITDEPTH ) & 0x07;

            // on byte overflow (should be on <0)
            if ( g_DRV_byPenCurrentBit > 7 )
            {
                g_DRV_byPenCurrentBit += 8;
                g_DRV_pbyPenCurrentAddr++;
            }
        }
        else
        {
            // scaled pen

            // adjust next bit position (sub scale factor)
            g_DRV_sfpPenCurrentBitPos -= g_DRV_sfpPenScaleFactor * iPixels;

            uBytes = 0;

            // if the byte is used up
            if  ( g_DRV_sfpPenCurrentBitPos < 0 )
            {
                // invert
                g_DRV_sfpPenCurrentBitPos = -g_DRV_sfpPenCurrentBitPos;

                // calc number of bytes
                uBytes = ( SFP_TO_INT( g_DRV_sfpPenCurrentBitPos ) >> 3 ) + 1;
                // calc new bit position
                g_DRV_sfpPenCurrentBitPos = INT_TO_SFP( 8 ) - ( g_DRV_sfpPenCurrentBitPos & ( INT_TO_SFP( 8
                            ) - 1 ) );
            }
        }

        // adjust byte pos
        g_DRV_pbyPenCurrentAddr += uBytes;
```

```
        // adjust pen address
        while ( ( g_DRV_pbyPenCurrentAddr < g_DRV_pbyPenStartAddr ) || ( g_DRV_pbyPenCurrentAddr >=
                            g_DRV_pbyPenStopAddr ) )
            g_DRV_pbyPenCurrentAddr -= g_DRV_pbyPenStopAddr - g_DRV_pbyPenStartAddr;
    }

    // . . . . . .
    // spool backwards
    if ( iPixels < 0 )
    {
        // adjust number of pixels
        iPixels = -iPixels;

        if ( !g_DRV_bPenScaled )
        {
            // normal pen

            // get bits and bytes to spool
            uBytes = CMG_DRV_BYTE_FROM_X( iPixels );

            // calc bit
            g_DRV_byPenCurrentBit += ( iPixels * CMG_CTRL_BITDEPTH ) & 0x07;

            // on byte overflow
            if ( g_DRV_byPenCurrentBit > 7 )
            {
                g_DRV_byPenCurrentBit -= 8;
                g_DRV_pbyPenCurrentAddr--;
            }
        }
        else
        {
            // scaled pen

            // adjust next bit position (sub scale factor)
            g_DRV_sfpPenCurrentBitPos += g_DRV_sfpPenScaleFactor * iPixels;

            uBytes = 0;

            // if the byte is used up
            if  ( g_DRV_sfpPenCurrentBitPos >= INT_TO_SFP( 8 ) )
            {
                // calc number of bytes
                uBytes = ( (cmg_mu16)( SFP_TO_INT( g_DRV_sfpPenCurrentBitPos ) ) ) >> 3;
                // calc new bit position
                g_DRV_sfpPenCurrentBitPos = g_DRV_sfpPenCurrentBitPos & ( INT_TO_SFP( 8 ) - 1 );
            }
        }

        // adjust byte pos
        g_DRV_pbyPenCurrentAddr -= uBytes;

        // adjust pen address
        while ( ( g_DRV_pbyPenCurrentAddr < g_DRV_pbyPenStartAddr ) || ( g_DRV_pbyPenCurrentAddr >=
                            g_DRV_pbyPenStopAddr ) )
            g_DRV_pbyPenCurrentAddr += g_DRV_pbyPenStopAddr - g_DRV_pbyPenStartAddr;
    }
}
```

Listing 76 – CMG_DRV_UNI_1BPP.c, SpoolPenRelative

SpoolPenRelative(…) teilt sich in zwei Teile auf: Bei einem positiven Argument wird nach vorne gespult, bei einem negativen zurück. Jeder dieser Teile teilt sich wiederum in zwei Abschnitte auf: Den ersten für normale Stifte, den zweiten für skalierte Stifte.

Innerhalb dieser Abschnitte wird immer der neue Wert für das aktuelle Bit und die Anzahl an ganzen Bytes berechnet. Diese müssen je nach Situation noch an ihre Grenzen angepaßt werden.

Anschließend wird nun noch die soeben ermittelte Anzahl an ganzen Bytes weitergespult und der `CurrentAddr`-Zeiger an den Start- und Stop-Adressen ausgerichtet.

Die Funktion `PenState_SaveRestore(…)` gibt nur die entsprechende Bitposition zurück und/oder setzt diese. Falls eine der beiden Funktionen nicht benötigt wird, kann für diesen Pointer `NULL` übergeben werden.

8.5.2. ROP-Modes

Der Inhalt dieses Abschnitts ist eher theoretisch und recht umfangreich. Er ist jedoch für die weiteren Schichten und Funktionen unerläßlich und sollte gut verstanden werden.

Um die Arbeitsweise der nächsten Funktionen verstehen zu können, muß man sich vor Augen führen, wie das eigentliche Zeichnen abläuft. Leider ist es nämlich im 1bpp-Modus nicht so einfach möglich einen einzelnen Pixel, geschweige denn mehrere, zusammenhängige nach bestimmten Vorgaben zu setzen, da man nur byteweise auf den Speicher zugreifen kann. Zusätzlich muß auch noch beachtet werden ob mit einer konstanten Farbe oder mit einem Stift (Textur) gezeichnet werden soll sowie wie diese Quellpixel mit dem Hintergrund verknüpft werden sollen (ROP-Mode).

Für diese Aufgabe gibt es nun drei Funktionen:

```
cmg_u8  _DRV_ROP_MaskSolidColor( cmg_u8 byData, cmg_u8 byBitMask );
cmg_u8  _DRV_ROP_MaskSource( cmg_u8 byData, cmg_u8 byBitMask, cmg_u8 bySource );
cmg_u8  _DRV_ROP_Source( cmg_u8 byData, cmg_u8 bySource );
```

Listing 77 – CMG_DRV_UNI_1BPP.c, ROP-Funktionen Definition

Für den Umgang mit konstanten Farben ist die erste Version zuständig; für den Umgang mit Stiften die beiden anderen.

Allen Funktionen ist gleich, daß sie in `byData` den aktuellen Byte-Wert des betreffenden Bytes aus dem Displayspeicher bekommen und als Rückgabewert das Ergebnis der ROP zurückliefern, welches dann wieder in den Displayspeicher gesetzt werden muß.

Nachstehend werden die drei Funktionen näher erläutert:

8.5.2.1. ROP MASKSOLIDCOLOR

`MaskSolidColor(…)` bearbeitet die Displaydaten mit der gewählten, konstanten Farbe. Der zweite Parameter ist eine Bitmaske die angibt, welche Bits aus dem Displaybyte bearbeitet werden sollen. Soll also nur ein einzelnes Bit gesetzt werden, so besitzt die Maske an dieser Stelle eine 1, sollen mehrere Bits gesetzt werden, besitzt die Maske an jeder dieser Stellen eine 1:

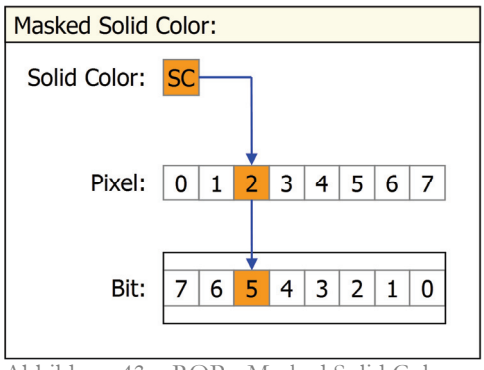


Abbildung 43 – ROPs, Masked Solid Color

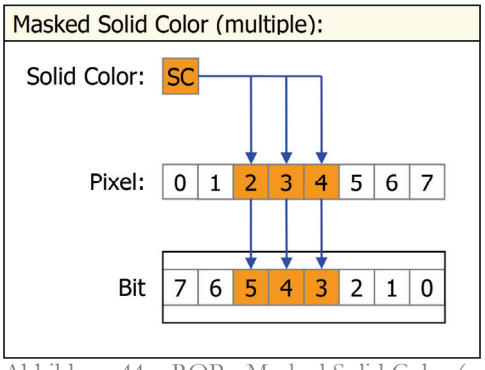


Abbildung 44 – ROPs, Masked Solid Color (multiple)

Im ersten Fall besitzt die Maske den binären Wert *00100000b*, im zweiten den Wert *00111000b*.

Je nach zuvor eingestelltem ROP-Mode werden die in `byBitMask` selektierten Quellbits aus `byData` mit der globalen Farbe `g_DRV_SolidColor` verknüpft:

```
/*-------------------------------------------------------------------------
| _DRV_ROP_Mask
|-----------------------------------
| Rasterization Operation
| apply the bitmask and the global color
| with the selected ropmode to the data byte
+-------------------------------------------------------------------------
| @Params:
|    * byData:       data byte
|    * byBitMask:    bit mask
| @Return Value:    modified data byte
+-------------------------------------------------------------------------*/
_PRIVATE
cmg_u8  _DRV_ROP_MaskSolidColor( cmg_u8 byData, cmg_u8 byBitMask )
{
    // manipulate pixels
    switch ( g_DRV_uROPMode )
    {
        case DRV_ROPMODE_COPY:
            if ( g_DRV_SolidColor )
                byData |= byBitMask;
            else
                byData &= ~byBitMask;
            break;

        case DRV_ROPMODE_XOR:
            byData ^= byBitMask;
            break;

        case DRV_ROPMODE_OR:
            if ( g_DRV_SolidColor )
                byData |= byBitMask;
            break;

        case DRV_ROPMODE_AND:
            if ( !g_DRV_SolidColor )
                byData &= ~byBitMask;
            break;
    }
    return byData;
}
```

Listing 78 – CMG_DRV_UNI_1BPP.c, ROP_MaskSolidColor

8.5.2.2. ROP MASKSOURCE

`MaskSource(…)` bearbeitet die Displaydaten mit den, ebenfalls mitübergebenen, Quelldaten. Der zweite Parameter ist wieder die Bitmaske die angibt, welche Bits aus dem Display- bzw. Quellbyte bearbeitet werden sollen:

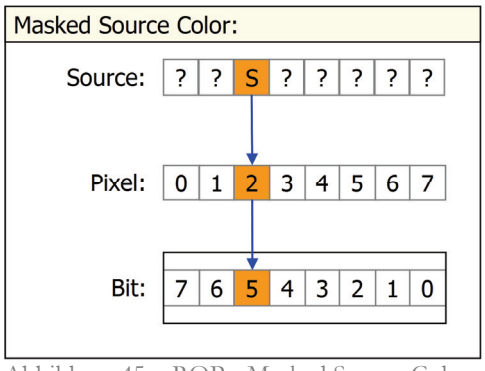


Abbildung 45 – ROPs, Masked Source Color

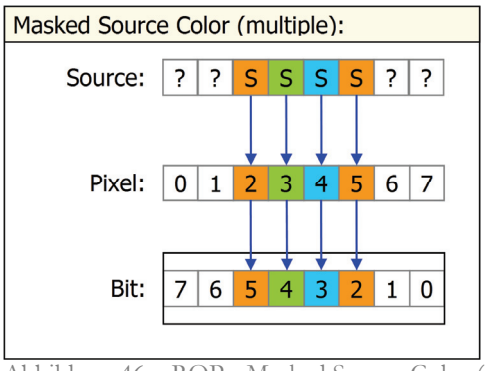


Abbildung 46 – ROPs, Masked Source Color (multiple)

Bei allen gesetzten Bits aus der Maske werden die Quell- mit den Displaydaten mit dem gewählten ROP-Mode verknüpft:

```
/*-------------------------------------------------------------------------
| _DRV_ROP_MaskSource
|-----------------------------------
| Rasterization Operation
| apply the bitmask and the source
| with the selected ropmode to the data byte
+-------------------------------------------------------------------------
| @Params:
```

```
|   * byData:       input data byte
|   * byBitMask:    bit mask
|   * bySource:     source byte
| @Return Value:    modified data byte
+-------------------------------------------------------------------------*/
_PRIVATE
cmg_u8  _DRV_ROP_MaskSource( cmg_u8 byData, cmg_u8 byBitMask, cmg_u8 bySource )
{
    // manipulate pixels
    switch ( g_DRV_uROPMode )
    {
        case DRV_ROPMODE_COPY:
            byData = ( bySource & byBitMask ) | ( byData & ( ~byBitMask ) );
            break;

        case DRV_ROPMODE_XOR:
            byData = ( bySource & byBitMask ) ^ byData;
            break;

        case DRV_ROPMODE_OR:
            byData = ( bySource & byBitMask ) | byData;
            break;

        case DRV_ROPMODE_AND:
            byData = ( bySource | ( ~byBitMask ) ) & byData;
            break;
    }
    return byData;
}
```

Listing 79 – CMG_DRV_UNI_1BPP.c, ROP_MaskSource

8.5.2.3. ROP SOURCE

`Source(…)` ist beim genauen Betrachten nur eine optimierte Vereinfachung der `MaskSource(…)`-Funktion. Bei größeren Datenbereichen kommt es öfters vor, daß ein komplettes Byte bearbeitet werden soll, also acht Pixel in Folge. Diese Situation würde der Bitmaske 11111111b entsprechen:

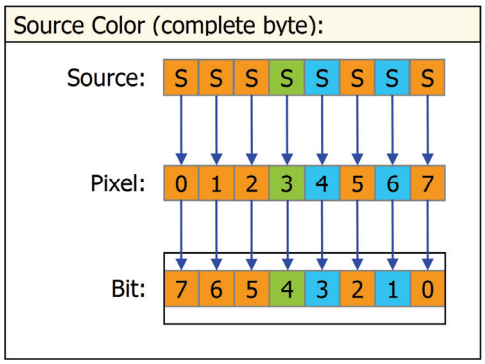


Abbildung 47 – ROPs, Source Color (komplettes Byte)

Bei dieser Funktion werden also alle Bits mit den Quellbits verknüpft:

```
/*-------------------------------------------------------------------------
| _DRV_ROP_MaskSource
|-----------------------------------
| Rasterization Operation
| apply the source on the complete byte
| with the selected ropmode to the data byte
+-------------------------------------------------------------------------
| @Params:
|   * byData:       input data byte
|   * bySource:     source byte
| @Return Value:    modified data byte
+-------------------------------------------------------------------------*/
_PRIVATE
cmg_u8  _DRV_ROP_Source( cmg_u8 byData, cmg_u8 bySource )
{
    // manipulate pixels
    switch ( g_DRV_uROPMode )
    {
        case DRV_ROPMODE_COPY:
            byData = bySource;
            break;

        case DRV_ROPMODE_XOR:
            byData = bySource ^ byData;
            break;
```

```
        case DRV_ROPMODE_OR:
            byData = bySource | byData;
            break;

        case DRV_ROPMODE_AND:
            byData = bySource & byData;
            break;
    }
    return byData;
}
```

Listing 80 – CMG_DRV_UNI_1BPP.c, ROP_Source

8.5.3. GETBITS

Für das Zeichnen mit Stiften können wir nun die letzten zwei ROP-Funktionen benutzen. Zusätzlich bekommen wir auch die Displaydaten. Was uns jetzt noch fehlt sind die Daten aus dem Stift, die sogenannten Source-Bits. Auch hier könnte es so einfach sein, wenn die Bitpositionen der zu verknüpfenden Bytes übereinstimmen würden. Die Stiftdaten werden aber seriell aus den Stiftdaten gelesen – bitweise und ohne Rücksicht auf etwaige Bytegrenzen.

Im folgenden Beispiel wird die Funktion `GetBits(…)` dreimal hintereinander aufgerufen – mit den Werten 4, 2 und 5. Sie holt bei jedem Aufruf die gewünschte Anzahl an Bits aus dem Pen-Bitstrom und richtet die Bits rechts, also am LSB, aus. Man kann sich die Operation auch wie ein Linksshiften in das Zielbyte vorstellen:

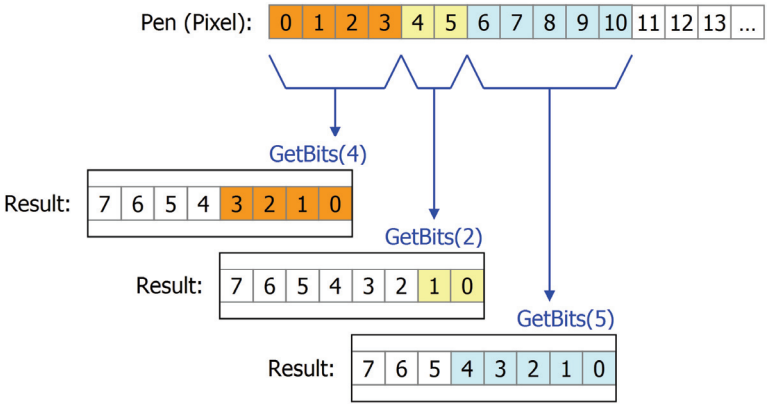


Abbildung 48 – GetBits

Im DRV-Treiber wählt die Funktion `GetBits(…)` zwischen dem normalen und dem skalierten Modus aus:

```
/*-------------------------------------------------------------------------
| _DRV_GetBits
|-----------------------------------
| Get next byLenght source bits from pen. Upper unused bits are NOT defined!
| If you need the exact data you must AND the result with the amount of bits.
+-------------------------------------------------------------------------
| ! Returns normal pen bits or scaled pen bits according to the current mode.
+-------------------------------------------------------------------------
| @Params:
|   * byLength:     number of bits to get (maximum == 8)
| @Return Value:    data bits (right aligned)
+-------------------------------------------------------------------------*/
_PRIVATE
cmg_u8 _DRV_GetBits( cmg_u8 byLength )
{
    if ( !g_DRV_bPenScaled )
        return ( _DRV_GetBits_Normal( byLength ) );
    else
        return ( _DRV_GetBits_Scaled( byLength ) );
}
```

Listing 81 – CMG_DRV_UNI_1BPP.c, GetBits

Der normale und oben beschriebene Fall wird so implementiert:

```
/*-----------------------------------------------------------------------------
| _DRV_GetBits
|-----------------------------------
| Get next byLenght source bits from pen. Upper unused bits are NOT defined!
| If you need the exact data you must AND the result with the amount of bits.
+----------------------------------------------------------------------------
| @Params:
|   * byLength:     number of bits to get (maximum == 8)
| @Return Value:    data bits (right aligned)
+----------------------------------------------------------------------------*/
_PRIVATE
_INLINE_FORCE
cmg_u8 _DRV_GetBits_Normal( cmg_u8 byLength )
{
    cmg_u8  byData, byHelper;

    // helper var
    byHelper = g_DRV_byPenCurrentBit + 1;

    // can we get all needed bits from the current byte?
    if ( byHelper >= byLength )
    {
        // get and align bits
        byData = *g_DRV_pbyPenCurrentAddr >> ( byHelper - byLength );
        // update bit position
        g_DRV_byPenCurrentBit -= byLength;
        // if all bits of this byte are used
        if ( g_DRV_byPenCurrentBit > 7 )    // underflow
        {
            // next byte with reset on pen end
            g_DRV_byPenCurrentBit = 7;
            if ( ++g_DRV_pbyPenCurrentAddr == g_DRV_pbyPenStopAddr )
                g_DRV_pbyPenCurrentAddr = g_DRV_pbyPenStartAddr;
        }
    }
    else
    {
        // we need to get the bits from two bytes
        // modify helper
        byHelper = byLength - byHelper;
        // get first half
        byData = *g_DRV_pbyPenCurrentAddr << byHelper;
        // next byte with reset on pen end
        if ( ++g_DRV_pbyPenCurrentAddr == g_DRV_pbyPenStopAddr )
            g_DRV_pbyPenCurrentAddr = g_DRV_pbyPenStartAddr;
        // get second half
        byData |= *g_DRV_pbyPenCurrentAddr >> ( 8 - byHelper );
        // update bit position
        g_DRV_byPenCurrentBit = g_DRV_byPenCurrentBit + 8 - byLength;
    }

    // return bits
    return byData;
}
```

Listing 82 – CMG_DRV_UNI_1BPP.c, GetBits_Normal

Mit dieser Funktion können maximal 8 neue Bits auf einmal eingelesen werden. Die zusätzliche Schwierigkeit ist, daß beim Einlesen ein Byteumbruch erfolgen kann, also wenn mehr Bits angefordert werden, als das gerade aktuelle Byte des Stiftes noch besitzt. Diese Funktion ist optimiert und benötig normalerweise einen Schritt, bei einem Byteumbruch jedoch zwei Schritte.

Beim skalierten Einlesen von Bits werden – je nach Skalierungsfaktor – manche Quellbits mehrfach eingelesen bzw. komplett ausgelassen. Beim Faktor 2.0 wird nur genau jedes zweite Bit verwendet:

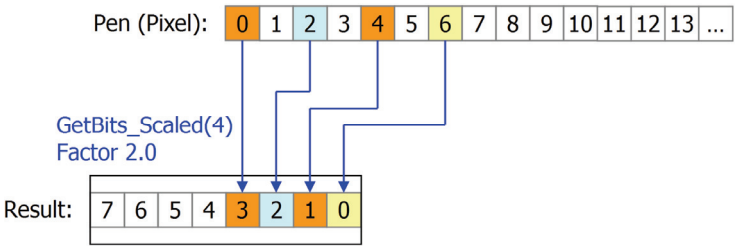


Abbildung 49 – GetBits_Scaled, Faktor 2.0

Beim Faktor 0.5 hingegen wird jedes Bit zweimal verwendet:

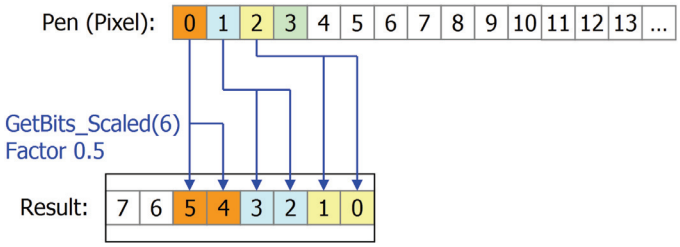


Abbildung 50 – GetBits_Scaled, Faktor 0.5

Der Quellcode dieser Funktion läßt sich nicht so stark optimierten und arbeitet Bit für Bit. Er braucht also für acht Bit auch genau acht Schritte:

```
/*-------------------------------------------------------------------------
| _DRV_GetBits
|-----------------------------------
| Get next byLenght source bits from pen. Upper unused bits are NOT defined!
| If you need the exact data you must AND the result with the amount of bits.
+-------------------------------------------------------------------------
| @Params:
|   * byLength:     number of bits to get (maximum == 8)
| @Return Value:    data bits (right aligned)
+-------------------------------------------------------------------------*/
_PRIVATE
_INLINE_FORCE
cmg_u8 _DRV_GetBits_Scaled( cmg_u8 byLength )
{
    cmg_u8  byData  ;

    // reset result
    byData = 0;

    // for each bit to get
    while ( byLength-- )
    {
        // shift current data byte left, and fill bit0 with the next source bit (g_DRV_sfpPenCurrentBitPos)
        byData = ( byData << 1 ) | ( ( *g_DRV_pbyPenCurrentAddr >> SFP_TO_INT( g_DRV_sfpPenCurrentBitPos )
                        ) & 0x01 );

        // adjust next bit position (sub scale factor)
        g_DRV_sfpPenCurrentBitPos -= g_DRV_sfpPenScaleFactor;

        // if the source byte is used
        while ( g_DRV_sfpPenCurrentBitPos < 0 )
        {
            // adjust bit position (add an new byte)
            g_DRV_sfpPenCurrentBitPos += INT_TO_SFP( 8 );

            // get next source byte with reset on pen end
            if ( ++g_DRV_pbyPenCurrentAddr == g_DRV_pbyPenStopAddr )
                g_DRV_pbyPenCurrentAddr = g_DRV_pbyPenStartAddr;
        }
    }

    // return bits
    return byData;
}
```

Listing 83 – CMG_DRV_UNI_1BPP.c, GetBits_Scaled

8.5.4. Zeichenfunktionen

Mit diesen Hilfsmitteln können nun die gewünschten Grafikfunktionen auch tatsächlich implementiert werden. Zur Erinnerung: Es standen ein Pixel, eine horizontale sowie eine vertikale Linie zur Auswahl.

8.5.4.1. Pixel

Die jeweils erste Aufgabe in den Ausgabefunktionen ist es, die benötigte Lese- und Schreibrichtung in der CTRL-Schicht festzulegen. Beim Setzen eines Pixels wird nur ein einzelnes Bit verändert. Das betreffende Byte muß eingelesen, das entsprechende Bit verändert und anschließend das Ergebnis auf die gleiche Adresse wieder herausgeschrieben werden. Damit

stehen die Reihenfolgen fest: Beim Lesen ohne Änderung und beim Schreiben ist es egal. Damit kommt `RD_NONE_WR_FASTEST` zum Einsatz:

```
/*************************************************************************
| CMG_DRV_Pixel
|-----------------------------------
| Set a pixel at x/y with the color
+------------------------------------------------------------------------
| @Params:
|   * iX:           x coord
|   * iY:           y coord
|   * byColor:      color
| @Return Value:    none
+************************************************************************/
_PUBLIC
void CMG_DRV_Pixel( cmg_Coord iX, cmg_Coord iY )
{
    cmg_u8      byBit, byBitMask;

    // set directions
    CMG_CTRL_SetDirections( CTRL_DIR_RD_NONE_WR_FASTEST );

    // set address
    CMG_CTRL_SetAddressPtr( ADDR_FROM_XY( iX, iY ) );

    // get bit / bitmask
    byBit = CMG_DRV_BIT_FROM_X( iX );
    byBitMask = BIT( byBit );

    // draw
    if ( g_DRV_uColorMode == DRV_COLORMODE_SOLIDCOLOR )
        CMG_CTRL_SetByte( _DRV_ROP_MaskSolidColor( CMG_CTRL_GetByte(), byBitMask ) );
    else
        CMG_CTRL_SetByte( _DRV_ROP_MaskSource( CMG_CTRL_GetByte(), byBitMask, (cmg_u8) ( _DRV_GetBits( 1 )
                          << byBit ) ) );
}
```

Listing 84 – CMG_DRV_UNI_1BPP.c, Pixel

Danach muß die Funktion die Zieladresse berechnen und setzen. Die Adresse zeigt auf das Byte, in welchem sich das passende Bit für den Pixel befindet. Das bereits weiter oben beschriebene Makro `ADDR_FROM_XY(…)` übernimmt genau diese Aufgabe.

Anschließend muß das dem Pixel zugehörige Bit selektiert werden. Dafür wird `BIT_FROM_X(…)` eingesetzt. Aus dem Zahlenwert des zu verwendenden Bits wird mit dem Makro `Bit(…)` eine Maske erzeugt, die genau aus einer Eins an der gewählten Bitposition besteht.

Das eigentliche Zeichen teilt sich in zwei Teile auf: Entweder in das Zeichen mit konstanter Farbe oder das Zeichnen mit einem Stift.

Bei der konstanten Farbe wird das komplette Displaybyte eingelesen, mit der ROP-Funktion `MaskSolidColor(…)` anhand der berechneten Maske mit der Farbe verknüpft und anschließend mit `SetByte(…)` wieder in den Speicher zurückgeschrieben.

Bei einem Stift wird das eingelesene Byte mit der ROP-Funktion `MaskSource(…)` anhand der Bitmaske bearbeitet. Nur diesmal dient das von `GetBits( 1 )` zurückgelieferte Bit, welches an die richtige Stelle geshiftet wurde, als Quelle.

8.5.4.2. Horizontale Linie

Die horizontale Linie ist, da sie stark optimiert ist, die komplizierteste Variante der Grafikbefehle:

```
/*************************************************************************
| CMG_DRV_HLine
|-----------------------------------
| draws a horizantal line rightwards (increasing x)
| (optimized)
+------------------------------------------------------------------------
| @Params:
|   * iX:           x start point
|   * iY:           y start point
|   * iLength:      number of pixels to draw rightwards
| @Return Value:    none
+************************************************************************/
```

```
_PUBLIC
void CMG_DRV_HLine( cmg_Coord iX, cmg_Coord iY, cmg_Coord iLength )
{
    cmg_u8      byDestAddrBit, byBitMask, bySource;

    // set directions
    CMG_CTRL_SetDirections( CTRL_DIR_RD_NONE_WR_RIGHT );

    // set address
    CMG_CTRL_SetAddressPtr( ADDR_FROM_XY( iX, iY ) );

    // get bit
    byDestAddrBit = CMG_DRV_BIT_FROM_X( iX );

    // small line fitting in one byte
    if ( ( ((cmg_Coord)byDestAddrBit) - iLength ) >= 0 )
    {
        // detailed pixel mask calculation:
        // byBitMask = ( ( 1 << ( byBit + 1 ) ) - 1 ) - ( ( 1 << ( byBit + 1 - iLength ) ) - 1 );
        // optimized:
        byBitMask = ( BIT( byDestAddrBit ) - BIT( byDestAddrBit - iLength ) ) << 1;

        // draw
        if ( g_DRV_uColorMode == DRV_COLORMODE_SOLIDCOLOR )
        {   // solid color
            CMG_CTRL_SetByte( _DRV_ROP_MaskSolidColor( CMG_CTRL_GetByte(), byBitMask ) );
        }
        else
        {   // pen
            bySource = _DRV_GetBits( (cmg_u8) iLength ) << ( byDestAddrBit + 1 - iLength );
            CMG_CTRL_SetByte( _DRV_ROP_MaskSource( CMG_CTRL_GetByte(), byBitMask, bySource ) );
        }
        // _END_ line
    }
    else
    {
        // first byte (align on 8 bit)
        if ( byDestAddrBit != 7 )
        {
            // draw
            byBitMask = ( 1 << ( byDestAddrBit + 1 ) ) - 1;
            if ( g_DRV_uColorMode == DRV_COLORMODE_SOLIDCOLOR )
            {   // solid color
                CMG_CTRL_SetByte( _DRV_ROP_MaskSolidColor( CMG_CTRL_GetByte(), byBitMask ) );
            }
            else
            {   // pen
                bySource = _DRV_GetBits( (cmg_u8)( byDestAddrBit + 1 ) );
                CMG_CTRL_SetByte( _DRV_ROP_MaskSource( CMG_CTRL_GetByte(), byBitMask, bySource ) );
            }

            iLength -= byDestAddrBit + 1;
        }

        // calc number of mid bytes, store temp in iX
        iX = CMG_DRV_BYTE_FROM_X( iLength );
        // calc number of end bits, store temp in iLength
        iLength &= 0x07;

        // mid bytes (aligned on 8 bit)
        // rastermode?
        if ( g_DRV_uROPMode == DRV_ROPMODE_COPY )
        {
            // copy rastermode
            // draw
            if ( g_DRV_uColorMode == DRV_COLORMODE_SOLIDCOLOR )
            {   // solid color
                bySource = g_DRV_SolidColor ? 0xff : 0x00;
                while ( iX-- )
                    CMG_CTRL_SetByte( bySource );
            }
            else
            {   // pen
                while ( iX-- )
                    CMG_CTRL_SetByte( _DRV_GetBits( 8 ) );
            }
        }
        else
        {
            // other rastermodes
            // draw
            if ( g_DRV_uColorMode == DRV_COLORMODE_SOLIDCOLOR )
            {   // solid color
                while ( iX-- )
                    CMG_CTRL_SetByte( _DRV_ROP_MaskSolidColor( CMG_CTRL_GetByte(), 0xff ) );
            }
            else
            {   // pen
                while ( iX-- )
                    CMG_CTRL_SetByte( _DRV_ROP_Source( CMG_CTRL_GetByte(), _DRV_GetBits( 8 ) ) );
            }
        }

        // last byte
        if ( iLength > 0 )
        {
            // draw
            byBitMask = -( 1 << ( 8 - iLength ) );
            if ( g_DRV_uColorMode == DRV_COLORMODE_SOLIDCOLOR )
            {   // solid color
                CMG_CTRL_SetByte( _DRV_ROP_MaskSolidColor( CMG_CTRL_GetByte(), byBitMask ) );
            }
```

```
            else
            {   // pen
                bySource = _DRV_GetBits( (cmg_u8) iLength ) << ( 8 - iLength );
                CMG_CTRL_SetByte( _DRV_ROP_MaskSource( CMG_CTRL_GetByte(), byBitMask, bySource ) );
            }
        }
    }
}
```

Listing 85 – CMG_DRV_UNI_1BPP.c, HLine

Auch diese lange Funktion setzt im ersten Teil die Richtungen und die Startadresse.

Danach teilt sich die Funktion in zwei Teile, von denen nur genau eine ausgeführt wird:

- Der erste Teil, wenn sich die Linie maximal auf ein Displaybyte beschränkt, also maximal acht Pixel lang ist und über keine Bytegrenze hinweg geht.
- Der zweite Teil, wenn dies nicht der Fall ist und die Linie mindestens über eine Bytegrenze geht.

Der erste Teil läuft ähnlich der Pixel-Funktion ab, nur die Bitmaske und die Anzahl der Bits aus dem Stift sind unterschiedlich. Die Bitmaske wird so berechnet, daß genau die betreffenden Bits für die Pixel der Linie gesetzt werden.

Der zweite Teil teilt sich wiederum in drei weitere Teile auf, die jedoch alle nacheinander ausgeführt werden. Je nach Lage der Linie können einzelne Teile auch komplett ausgelassen werden:

- Der erste Teil bearbeitet alle Bits bis zur ersten Bytegrenze. Falls die Linie genau auf einer Grenze beginnt, fällt dieser Schritt aus.
- Der zweite Teil bearbeitet alle darauf folgenden kompletten Bytes, also jeweils acht Pixel in Folge. Dieser Schritt wird ausgelassen, wenn die Linie nicht genügend lang ist und kein ganzes Byte mehr zustande kommt.
- Der letzte Teil bearbeitet die noch restlichen Bits vom letzten Byteanfang an. Er fällt aus, wenn die Linie genau auf einer Bytegrenze endet.

8.5.4.3. VERTIKALE LINIE

Die vertikale Linie ist – aufgrund des linearen Speichermodells – nicht so sehr zu optimieren wie ihr horizontales Pendant. Sie geht einfach Zeile für Zeile durch und bearbeitet den jeweils betreffenden Pixel:

```
/*****************************************************************************
| CMG_DRV_VLine
|-----------------------------------
| draws a vertical line downwards (increasing y)
| (optimized)
+----------------------------------------------------------------------------
| @Params:
|   * iX:           x start point
|   * iY:           y start point
|   * iLength:      number of pixels to draw downwards
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_DRV_VLine( cmg_Coord iX, cmg_Coord iY, cmg_Coord iLength )
{
    cmg_u8      byBit, byBitMask;

    // set directions
    CMG_CTRL_SetDirections( CTRL_DIR_RD_NONE_WR_DOWN );

    // set address
    CMG_CTRL_SetAddressPtr( ADDR_FROM_XY( iX, iY ) );

    // get bit
    byBit = CMG_DRV_BIT_FROM_X( iX );
    byBitMask = BIT( byBit );

    // draw
    if ( g_DRV_uColorMode == DRV_COLORMODE_SOLIDCOLOR )
    {
        while ( (iLength--) > 0 )
```

```
            CMG_CTRL_SetByte( _DRV_ROP_MaskSolidColor( CMG_CTRL_GetByte(), byBitMask ) );
    }
    else
    {
        while ( (iLength--) > 0 )
            CMG_CTRL_SetByte( _DRV_ROP_MaskSource( CMG_CTRL_GetByte(), byBitMask, (cmg_u8) ( _DRV_GetBits(
                        1 ) << byBit ) ) );
    }
}
```

Listing 86 – CMG_DRV_UNI_1BPP.c, VLine

8.5.4.4. GETHLINE

Der Funktion GetHLine(…) wurde bisher noch gar keine Aufmerksamkeit gewidmet. Sie verändert nichts am Displayspeicher, sondern liest lediglich die betreffende Linie ein:

```
/*****************************************************************************
| CMG_DRV_GetHLine
|-----------------------------------
| Get screen bits into memory buffer
+-----------------------------------------------------------------------------
| @Params:
|   * pDestBuffer:  ptr to destination buffer that receives bits (make sure
|                   it large enough)
|   * iX:           x start point
|   * iY:           y start point
|   * iLength:      number of pixels get
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_DRV_GetHLine( cmg_u8 *pDestBuffer, cmg_Coord iX, cmg_Coord iY, cmg_Coord iLength )
{
    cmg_u8      byBit;

    // set directions
    CMG_CTRL_SetDirections( CTRL_DIR_RD_RIGHT_WR_INVALID );

    // set address
    CMG_CTRL_SetAddressPtr( ADDR_FROM_XY( iX, iY ) );

    // get bit
    byBit = CMG_DRV_BIT_FROM_X( iX );

    // calc number of bytes to process, store back in iLength
    iLength = CMG_DRV_FULL_BYTES_FROM_X( iLength );


    // if screen x is byte aligned we can just copy byte for byte
    if ( byBit == 7 )
    {
        // for all bytes
        while ( iLength-- )
            // store dest byte
            *pDestBuffer++ = CMG_CTRL_GetByte();
    }
    // if screen x is _NOT_ byte aligned we must compound the destination
    // byte by shifting the two screen bytes
    else
    {
        cmg_u8      byDestByte, byScreenByte, byShiftLeftBits;

        // calculate shift bit counts (left and right (stored in byByt to safe one local var))
        byShiftLeftBits  = 7 - byBit;
        byBit++;

        // get first screen byte
        byScreenByte = CMG_CTRL_GetByte();

        // for all bytes
        while ( iLength-- )
        {
            // adjust first half
            byDestByte = byScreenByte << byShiftLeftBits;
            // get next screen byte
            byScreenByte = CMG_CTRL_GetByte();
            // adjust second half
            byDestByte |= byScreenByte >> byBit;
            // store dest byte
            *pDestBuffer++ = byDestByte;
        }
    }
}
```

Listing 87 – CMG_DRV_UNI_1BPP.c, GetHLine

Der Anfang ist wieder genau wie bei den anderen Funktionen, nur wird diesmal der Modus `RD_RIGHT_WR_INVALID` benutzt, da keine Schreibzugriffe stattfinden.

Die Funktion teilt sich in zwei Teile:

- Der erste Teil wird ausgeführt, wenn sich der Start des einzulesenden Bildschirmbereichs genau am Byteanfang, also an einer Bytegrenze, befindet. Dadurch können die betreffenden Bytes einfach direkt kopiert werden.
- Der zweite Teil wird ausgeführt, wenn der Start eben nicht auf so einer Grenze liegt. Die Teile werden dann – jeweils um die richtige Bitanzahl verschoben – gespeichert.

8.6. Überlegungen für weitere Module

Um weitere Module mit anderen Farbtiefen zu implementieren, sollte man sich folgende Gedanken über die notwendigen Änderungen machen: Im Prinzip sind die Änderungen nicht sehr komplex. Es ändert sich lediglich die Bit-Breite eines Pixels. Allerdings beeinflußt diese Tatsache nahezu jede Funktion dieser Schicht.

Die oben vorgestellte Quelldatei ist nur aus dem Grund so umfangreich, weil nicht direkt auf die einzelnen Pixel zugegriffen werden kann, sondern nur auf ganze Bytes. Am einfachsten ist diese Schicht mit einer Farbtiefe von acht Bit zu implementieren, da hier die Abbildung eins zu eins stimmt. Damit würde nur ein Bruchteil des obigen Aufwandes anfallen, da keinerlei Sonderfälle zu beachten sind und es keine Bit- bzw. Byteüberläufe gibt. Die Stiftimplementierung würde damit genauso einfach werden.

9. DRAW – DRAWING

9.1. AUFBAU

Die DRAW-Schicht stellt die API für die Anwendungsprogrammierung für den direkten Modus zur Verfügung. Sie verwendet und erweitert die grundlegenden Grafikfunktionen aus der DRV-Schicht und bietet allgemeinere Grafikfunktionen an.

Die DRAW-Schicht teilt sich in vier große Teile, die jeweils einige Aufgaben zu erfüllen haben:

- 1. Teil: **Main** (gemeinsame Komponenten)
- 2. Teil: **Pixel/Line** (Pixel und Linien)
- 3. Teil: **Solids** (füllbare, geometrische Figuren)
- 4. Teil: **Blitting**[33] (Bit-Block-Transfer)

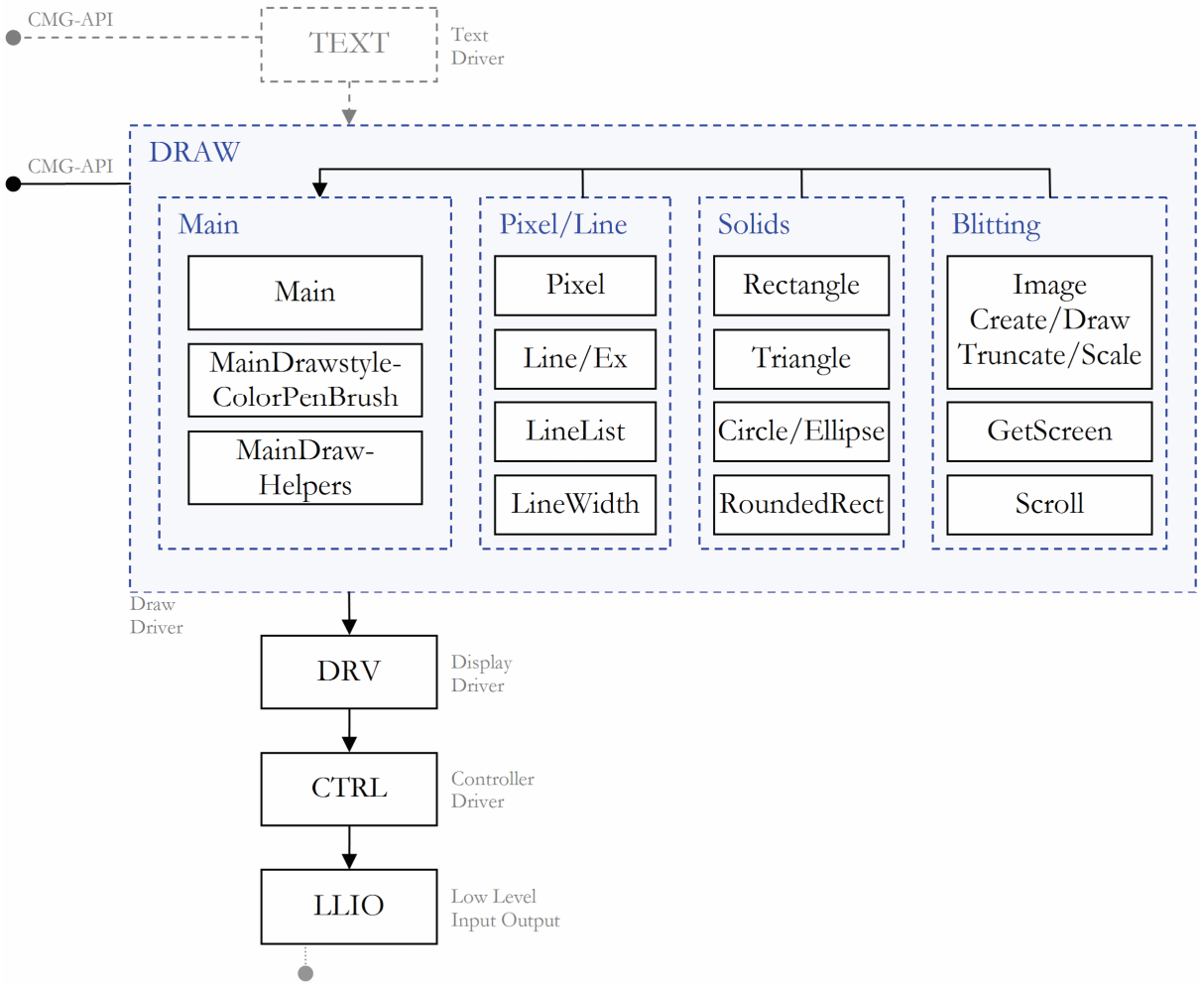


Abbildung 51 – DRAW Aufbau

[33] **Blitting:** *engl.* Bit Blit, Bit Block Transfer, Kopieren und Verschieben von Speicherinhalten (aus Wikipedia, Bit blit, 20. Juni 2007, 21:39 UTC, http://de.wikipedia.org/w/index.php?title=Bit_blit&oldid=33421808)

Im folgenden Abschnitt sollen der Aufbau und die Aufgaben der vier Bereiche näher erläutert werden:

- **Main, Gemeinsame Komponenten**
  Dieser Bereich beinhaltet alle, für jede Schicht notwendigen Funktionen (Init, Exit, etc.), alle von der Schicht benutzten, globalen Variablen sowie alle Funktionen zum Setzen der verschiedenen Statūs zum Zeichnen.

  Die DRAW-Schicht erweitert konsequent das eindimensionale Prinzip der Stifte aus DRV. Zusätzlich zu diesen Stiften, die weiterhin für die breitenlosen Zeichenfunktionen verwendet werden, führt DRAW die Pinsel ein (*engl. Brush*). Ein Pinsel ist eine zweidimensionale Textur, die auf auszufüllende Bereiche gelegt wird. Technisch gesehen ist ein Pinsel eine zeilenweise Aneinanderreihung von Stiften. Auch Pinsel können ebenfalls skaliert werden.

  Weitere Details zu den Pinseln entnehmen Sie bitte dem Abschnitt 9.4.

  Eine zusätzliche Erweiterung ist die Einführung des Clippings. Es kann zu jedem Zeitpunkt ein Rechteck auf dem Display definiert werden, auf welches die Zeichenausgaben beschränkt sein sollen. Weiter können auch Koordinaten außerhalb dieses Bereichs verwendet werden – CMG schneidet die Figuren jedoch an den Grenzen ab. Somit können – ganz ohne Aufwand – auch nur Teile komplizierter Zeichnungen gezeichnet werden. Zusätzlich wird die GUI-Schicht von diesem Feature ausführlich Gebrauch machen und beim Zeichnen eines jeden Fensters die Ausgabe auf den darin enthaltenen Bereich beschränken.

  Die letzte, sicher nicht unwichtigste Erweiterung, ist das Trennen der Displayorientierung von der logischen Orientierung: Alle Koordinaten der DRAW-Schicht sind logische Koordinaten, welche in den Funktionen in physikalische umgerechnet werden. Der Anwender kann verschiedene Flags setzen: Das Spiegeln an der X-Achse, das Spiegeln an der Y-Achse sowie das Vertauschen der X- und Y-Werte. Durch geschickte Kombinationen der Flags erreicht man das Rotieren des Displayinhalts um 90, 180 und 270 Grad. Somit kann ein Display unabhängig von der Einbaurichtung, ohne jegliche Programmänderung, verwendet werden.

- **Pixel/Line**
  Dieser Bereich deckt die Grundfunktionen nicht füllbarer, geometrischer Figuren ab. Dazu gehören Pixel und Linien mit Sonderformen wie Listen von Linien und Linien mit bestimmter Breite.

  Weiter würden in diesen Bereich Kurven und Bögen – wie Benzier, Splines und Arcs – gehören. Diese sind jedoch noch nicht implementiert.

- **Solids**
  Hier befinden sich alle geometrischen Figuren, die füllbar sind. Hierzu zählen bis jetzt Rechtecke, Dreiecke, Kreise, Ellipsen und abgerundete Rechtecke. Weiter denkbar wären unter anderem Kreissegmente und mehrpunktige Polygone.

  Alle Solids besitzen einen äußeren Rand um die Figur und einen inneren Füllbereich. Der Rand wird mit einem Stift gezeichnet, der Füllbereich mit einem Pinsel. Selbstverständlich können diese auch wieder konstante Farben sein.

Es gibt drei Zeichenmodi für Solids:

- **Nur Rand:** Es wird nur der äußere Rand gezeichnet. Die Innenfläche bleibt transparent (1. Bild jeweils mit konstanter Farbe, 2. Bild jeweils mit Stift und Pinsel):

- **Nur Füllen:** Die komplette Figur – inklusive Rand – wird mit dem Pinsel gefüllt. Es ist also kein expliziter Rand zu sehen.

- **Rand mit Füllen:** Es wird zuerst der Rand gezeichnet und dann der innere Bereich – ohne Rand – gefüllt.

- **Blitting**
  Unter Blitting versteht CMG das Zeichnen von Pixelbildern, also das Kopieren und Bearbeiten ganzer Speicherbereiche.

  Zum einen gibt es die Pixelbilder-Funktionen (*engl. Image*), wie das Erstellen von Bildern, das normale Zeichnen, das beschnittene Zeichnen sowie das skalierte Zeichnen dieser; zum anderen aber auch Funktionen um rechteckige Bildschirmbereiche einzulesen sowie ganze Regionen auf dem Bildschirm mit minimalem Speicheraufwand zu verschieben.

9.2. SCHNITTSTELLEN

Die DRAW-Schicht bietet gleich eine ganze Sammlung an Schnittstellen; die einzig öffentliche ist jedoch CMG_DRAW.h, welche zugleich die direkte API darstellt. Die anderen Definitionen stellen nur geometrische Basisfunktionalität für andere DRAW-Funktionen zur Verfügung.

9.2.1. ÖFFENTLICHE SCHNITTSTELLE

Zuerst die offizielle DRAW-Schnittstelle:

```
// ==================
// ==== Init/Exit ====
// ==================

    //-- Init/Exit ------------------------------------------------------------
        cmg_Result  CMG_DRAW_Init( void );
        void        CMG_DRAW_Exit( void );
        void        CMG_PowerManagement( cmg_mu8 uState );
        cmg_Result  CMG_ControllerSpecific( cmg_mu8 uFunction, cmg_void *pParam );
    //-----------------------------------------------------------------------


// ======================
// ==== Mode Exports =====
// ======================

        void        CMG_GetScreenSize( cmg_Coord *piScreenSizeX, cmg_Coord *piScreenSizeY );

    //-- Display Orientation -------------------------------------------------
            #define     DISPLAYORIENTATION_NORMAL       0x00
            #define     DISPLAYORIENTATION_XYSWAP       0x01
            #define     DISPLAYORIENTATION_XMIRROR      0x02
            #define     DISPLAYORIENTATION_YMIRROR      0x04
```

```
            #define     DISPLAYORIENTATION_ROTATE90     ( DISPLAYORIENTATION_XYSWAP |
                           DISPLAYORIENTATION_XMIRROR )
            #define     DISPLAYORIENTATION_ROTATE180    ( DISPLAYORIENTATION_XMIRROR |
                           DISPLAYORIENTATION_YMIRROR )
            #define     DISPLAYORIENTATION_ROTATE270    ( DISPLAYORIENTATION_XYSWAP |
                           DISPLAYORIENTATION_YMIRROR )
        void        CMG_SetDisplayOrientation( cmg_mu8 byDisplayOrientationMask );
        cmg_mu8     CMG_GetDisplayOrientation( void );
    //-------------------------------------------------------------------------

    //-- Clipping -----------------------------------------------------------
        void        CMG_SetClippingWindow( cmg_Coord iX1, cmg_Coord iY1, cmg_Coord iX2, cmg_Coord iY2 );
        void        CMG_GetClippingWindow( cmg_Coord *piX1, cmg_Coord *piY1, cmg_Coord *piX2, cmg_Coord
                       *piY2 );
    //-------------------------------------------------------------------------

    //-- ROP ----------------------------------------------------------------
            #define     ROPMODE_COPY                    0x00
            #define     ROPMODE_XOR                     0x01
            #define     ROPMODE_AND                     0x02
            #define     ROPMODE_OR                      0x03
        void        CMG_SetROPMode( cmg_mu8 uROPMode );
    //-------------------------------------------------------------------------

    //-- Drawstyle / Color / Pen / Brush ------------------------------------
            #define     DRAWSTYLE_BORDER                0x01
            #define     DRAWSTYLE_FILL                  0x02
            #define     DRAWSTYLE_BOTH                  0x03
        void        CMG_SetDrawStyle( cmg_mu8 uDrawStyle );
        void        CMG_SetColor( cmg_Color Color );
        void        CMG_SetFillColor( cmg_Color FillColor );
        void        CMG_SetPen( cmg_Pen *pPen );
        void        CMG_SetBrush( cmg_Brush *pBrush );
        void        CMG_CreatePen( cmg_Pen *pPen, cmg_u8 *pbyPenStartAddr, cmg_mu16 uPenLengthBytes );
        void        CMG_CreatePenEx( cmg_Pen *pPen, cmg_u8 *pbyPenStartAddr, cmg_mu16 uPenLengthBytes,
                       cmg_Coord iPenSpoolPixels, float fPenScaleFactor );
        void        CMG_CreateBrush( cmg_Brush *pBrush, cmg_u8 *pbyBrushStartAddr, cmg_mu16
                       uBrushLineLengthBytes, cmg_mu16 uBrushLines );
        void        CMG_CreateBrushEx( cmg_Brush *pBrush, cmg_u8 *pbyBrushStartAddr, cmg_mu16
                       uBrushLineLengthBytes, cmg_mu16 uBrushLines, cmg_Coord iBrushSpoolPixelsX,
                       cmg_Coord iBrushSpoolPixelsY, float fBrushScaleFactorX, float fBrushScaleFactorY
                       );
    //-------------------------------------------------------------------------


// ==========================
// ==== Drawing Exports =====
// ==========================

    //-- Pixel / Line / Wideless --------------------------------------------
        void        CMG_Pixel( cmg_Coord iX, cmg_Coord iY );

            #define     LINESUPPRESS_NONE               0x00
            #define     LINESUPPRESS_FIRST              0x01
            #define     LINESUPPRESS_LAST               0x02
        #define     CMG_Line(iX1,iY1,iX2,iY2)           CMG_LineEx( iX1, iY1, iX2, iY2, LINESUPPRESS_NONE )
        void        CMG_LineEx( cmg_Coord iX1, cmg_Coord iY1, cmg_Coord iX2, cmg_Coord iY2, cmg_u8
                       uSuppressPixelMask );
        void        CMG_LineList( cmg_Coord (*paiCoords)[2], cmg_mu16 uNumberOfPoints, cmg_bool bCloseList
                       );
        void        CMG_LineWidth( cmg_Coord iX1, cmg_Coord iY1, cmg_Coord iX2, cmg_Coord iY2, cmg_Coord
                       iWidth );
        //?? Arc
    //-------------------------------------------------------------------------

    //-- Solids -------------------------------------------------------------
        void        CMG_Rectangle( cmg_Coord iX1, cmg_Coord iY1, cmg_Coord iX2, cmg_Coord iY2 );
        void        CMG_Triangle( cmg_Coord iX1, cmg_Coord iY1, cmg_Coord iX2, cmg_Coord iY2, cmg_Coord
                       iX3, cmg_Coord iY3 );
        #define     CMG_Circle(iXCenter,iYCenter,iRadius)           CMG_Ellipse( iXCenter, iYCenter,
                       iRadius, iRadius )
        void        CMG_Ellipse( cmg_Coord iXCenter, cmg_Coord iYCenter, cmg_Coord iRadiusX, cmg_Coord
                       iRadiusY );
        void        CMG_RoundedRectangle( cmg_Coord iX1, cmg_Coord iY1, cmg_Coord iX2, cmg_Coord iY2,
                       cmg_Coord iRadius );
    //-------------------------------------------------------------------------

    //-- Blitting -----------------------------------------------------------
        void        CMG_CreateImage( cmg_Image *pImage, cmg_u8 *pbyImageStartAddr, cmg_Coord iImageWidth,
                       cmg_Coord iImageHeight, cmg_mu16 uImageLineLengthBytes );
        void        CMG_Image( cmg_Coord iDestX, cmg_Coord iDestY, cmg_Image *pImage );
        void        CMG_ImageTruncated( cmg_Coord iDestX, cmg_Coord iDestY, cmg_Coord iSizeX, cmg_Coord
                       iSizeY, cmg_Image *pImage, cmg_Coord iSrcStartX, cmg_Coord iSrcStartY );
        void        CMG_ImageScaled( cmg_Coord iDestX, cmg_Coord iDestY, cmg_Coord iDestSizeX, cmg_Coord
                       iDestSizeY, cmg_Image *pImage );

        cmg_mu16    CMG_GetScreen( cmg_u8 *pbyDestStartAddr, cmg_mu16 uDestAvailableBytes, cmg_mu16
                       *puDestBytesPerLine, cmg_Coord iSrcX, cmg_Coord iSrcY, cmg_Coord iSizeX,
                       cmg_Coord iSizeY );
        void        CMG_Scroll( cmg_Coord iDestX, cmg_Coord iDestY, cmg_Coord iSrcX, cmg_Coord iSrcY,
                       cmg_Coord iSizeX, cmg_Coord iSizeY );
    //-------------------------------------------------------------------------
```

Listing 88 – CMG_DRAW.h, Teil 1

Die Schnittstelle teilt sich in Init/Exit, in Statusfunktionen und in Zeichenfunktionen auf.

Zusätzlich werden öffentliche Datentypen für Stifte, Pinsel und Pixelbilder definiert:

```
typedef struct _cmg_Pen     {
    cmg_u8     *pbyPenStartAddr;            // pen start addr
    cmg_mu16    uPenLengthBytes;            // number of pen bytes
    cmg_sfp1616 sfpPenScaleFactor;          // pen scale factor
    cmg_Coord   iPenSpoolPixels;            // pen spool pixels
} cmg_Pen;

typedef struct  _cmg_Brush  {
    cmg_u8     *pbyBrushStartAddr;          // brush start addr
    cmg_mu16    uBrushLineLengthBytes;      // number of brush bytes per line
    cmg_mu16    uBrushLines;                // number of brush line
    cmg_sfp1616 sfpBrushScaleFactorX;       // brush scale factor x
    cmg_sfp1616 sfpBrushScaleFactorY;       // brush scale factor y
    cmg_Coord   iBrushSpoolPixelsX;         // brush spool pixels x
    cmg_Coord   iBrushSpoolPixelsY;         // brush spool pixels y
} cmg_Brush;

typedef struct  _cmg_Image  {
    cmg_u8     *pbyImageStartAddr;          // image start addr
    cmg_Coord   iImageWidth;                // image x
    cmg_Coord   iImageHeight;               // image y
    cmg_mu16    uImageLineLengthBytes;      // number of image bytes per line
} cmg_Image;
```

Listing 89 – CMG_DRAW.h, Teil 2

9.2.2. DRAW-Interne Schnittstellen

Intern besitzt die DRAW-Schicht – bis jetzt – drei weitere Schnittstellen:

- **Die Schnittstelle für die gemeinsamen Funktionen:**
  Hier enthalten sind alle globalen Variablen und die Hilfsfunktionen aus den gemeinsamen Komponenten:

```
//--------------------------------------------------------------------------
// Globals
//--------------------------------------------------------------------------

// display
extern  cmg_Coord   g_DRAW_iScreenSizeX;            // logical screen x size
extern  cmg_Coord   g_DRAW_iScreenSizeY;            // logical screen y size

extern  cmg_mu8     g_DRAW_uDisplayOrientationMask; // mirrow/swap flags

// clipping
extern  cmg_Coord   g_DRAW_iClippingWindow_LX;      // clipping rect left x
extern  cmg_Coord   g_DRAW_iClippingWindow_RX;      // clipping rect right x
extern  cmg_Coord   g_DRAW_iClippingWindow_TY;      // clipping rect top y
extern  cmg_Coord   g_DRAW_iClippingWindow_BY;      // clipping rect bottom y

extern  cmg_bool    g_DRAW_bProhibitCoordAdjustment;// internal use to prohibit coord adjm.

// byROPMode
extern  cmg_mu8     g_DRAW_uROPMode;                // rop mode

// drawstyle
extern  cmg_mu8     g_DRAW_uDrawStyle;              // drawstyle (border, solid, both)


// color modes
    #define     DRV_COLORMODE_SOLIDCOLOR        0x00
    #define     DRV_COLORMODE_PEN_BRUSH         0x01
extern  cmg_mu8     g_DRAW_uColorMode;              // current color mode (solid - pen)
extern  cmg_mu8     g_DRAW_uFillColorMode;          // current fillcolor mode (solid - brush)


// foreground color
extern  cmg_Color   g_DRAW_Color;                   // foreground color

// background color
extern  cmg_Color   g_DRAW_FillColor;               // background color

// pen
extern  cmg_Pen    *g_DRAW_pPen;                    // current selected pen

// brush
extern  cmg_Brush  *g_DRAW_pBrush;                  // current selected brush
extern  cmg_sfp1616 g_DRAW_sfpBrushCurrentLine;     // current brush fp line
//--------------------------------------------------------------------------


//--------------------------------------------------------------------------
// PRIVATE DRAW INTERNALS
//--------------------------------------------------------------------------
    void _DRAW_AdjustXY( cmg_Coord *pX, cmg_Coord *pY );
```

```
    void _DRAW_ApplyPen( cmg_Coord iSpoolX );
    void _DRAW_ApplyBrush( cmg_Coord iSpoolY );
    void _DRAW_ApplyBrushNextLine( cmg_Coord iLineSpoolX );

    void _DRAW_ClippedPixel( cmg_Coord iX, cmg_Coord iY );
    void _DRAW_ClippedHLine( cmg_Coord iXLeft, cmg_Coord iXRight, cmg_Coord iY, cmg_Coord
                             iSpoolLeftmostX );
//------------------------------------------------------------------------------
```

Listing 90 – CMG_DRAW_Main.h

- **Die Schnittstelle für Linienfunktionen**
  Eine der schnellsten Methoden eine Linie zu zeichnen, ist der Bresenham-Algorithmus. CMG verwendet einen auf diesem Prinzip aufbauenden Algorithmus. Da diese Funktionalität aber nicht nur von der eigentlichen Linienfunktion genutzt wird, sondern zum Beispiel auch von der Dreieck-Funktion, ist die Schnittstelle innerhalb der DRAW-Schicht öffentlich.

```
typedef struct _cmg_DRAW_Line_Data {
  // public coordinates:
    cmg_Coord       iX;             // current x
    cmg_Coord       iY;             // current y

  // public count:
    cmg_Coord       iCount;         // counting variable (if > 0 still have to draw)

  // private internals:
    cmg_Coord       iDeltaLong;     // long dir delta
    cmg_Coord       iDeltaShort;    // short dir delta
    cmg_Coord       iIncLong;       // long dir increment
    cmg_Coord       iIncShort;      // short dir increment
    cmg_Coord       iDrift;         // current line error
    cmg_Coord       iLong;          // current long dir
    cmg_Coord       iShort;         // current short dir
    cmg_bool        bSwapXY;        // swaps x/y
} cmg_DRAW_Line_Data;


//------------------------------------------------------------------------------
// PRIVATE DRAW INTERNALS
//------------------------------------------------------------------------------
_PROTECTED  void _Line_Loop_Init( cmg_DRAW_Line_Data *pLineData, cmg_Coord iX1, cmg_Coord iY1,
                          cmg_Coord iX2, cmg_Coord iY2 );
_PROTECTED  void _Line_Loop( cmg_DRAW_Line_Data *pLineData );
//------------------------------------------------------------------------------
```

Listing 91 – CMG_DRAW_Line.h

Die Schnittstelle definiert einen benutzerspezifischen Typ, der alle notwendigen Informationen einer Linie beinhaltet. Für den Anwender sind nur die ersten drei Elemente von Bedeutung, die restlichen werden intern zur Berechnung verwendet.

Zum Initialisieren einer Linie muß die `Loop_Init`-Funktion mit den entsprechenden Parametern der Linie aufgerufen werden. Solange `iCount` noch nicht Null ist sind weitere Punkte auf der Linie vorhanden. In jedem `Loop`-Aufruf werden die Daten aktualisiert und der nächste Punkt berechnet.

- **Die Schnittstelle für Ellipsenfunktionen**
  Die in CMG verwendete Ellipsenfunktion baut ebenfalls auf den Grundlagen des Bresenham-Algorithmus' für Kreise und Ellipsen auf:

```
typedef struct _cmg_DRAW_Ellipse_Data {
  // public coordinates:
    cmg_Coord       iX_Left;            // current left x
    cmg_Coord       iX_Right;           // current right x
    cmg_Coord       iY_Top;             // current top y
    cmg_Coord       iY_Bottom;          // current bottom y

  // public count:
    cmg_Coord       iCount;             // counting variable (if > 0 still have to draw)

  // private internals:
    cmg_CoordLong   iDeltaX;            // internal, (constant, mut be one time inited)
    cmg_CoordLong   iDeltaY;            // internal, (constant, mut be one time inited)
    cmg_CoordLong   iRot;               // internal
    cmg_CoordLong   iRotX;              // internal
    cmg_CoordLong   iRotY;              // internal
} cmg_DRAW_Ellipse_Data;
```

```
//---------------------------------------------------------------------------
// PRIVATE DRAW INTERNALS
//---------------------------------------------------------------------------
_PROTECTED  void _Ellipse_OneTimeInit( cmg_DRAW_Ellipse_Data *pEllData, cmg_Coord iRadiusX,
                        cmg_Coord iRadiusY );
_PROTECTED  void _Ellipse_LoopInside_Init( cmg_DRAW_Ellipse_Data *pEllData, cmg_Coord iMidX,
                        cmg_Coord iMidY, cmg_Coord iRadiusX, cmg_Coord iRadiusY );
_PROTECTED  void _Ellipse_LoopOutside_Init( cmg_DRAW_Ellipse_Data *pEllData, cmg_Coord iMidX,
                        cmg_Coord iMidY, cmg_Coord iRadiusX, cmg_Coord iRadiusY );
_PROTECTED  void _Ellipse_LoopInside( cmg_DRAW_Ellipse_Data *pEllData );
_PROTECTED  void _Ellipse_LoopOutside( cmg_DRAW_Ellipse_Data *pEllData );
```

Listing 92 – CMG_DRAW_Ellipse.h

Es existieren zwei verschiedene Versionen, welche sich in den gewünschten Richtungen und Startpunkten unterscheiden. Die Details dazu werden im betreffenden Abschnitt näher erläutert.

9.3. KONFIGURATION

In der DRAW-Schicht können wieder ein paar Einstellungen konfiguriert werden. Es folgt die komplette Konfigurationsdatei:

```
//############################################################################
// (4) CMG_DRAW - CONFIG
//############################################################################
//
// CMG_DRAW_*
// ======================
// DRAW configuration.
//
//
// For internal correct compilation.
// *** DO NOT CHANGE! ***
#define     CMG_DRAW_CONFIGURED
//
//
//============================================================================
// User Area:
// ==========
// Read the sections carefully and change the values accordingly to
// your needs.
//
//
// Basic functions always included:
// --------------------------------
// * Blt, Line, Pixel, Rectangle
//
//
// Define extra functions that you need:
// -------------------------------------
#define     CMG_DRAW_ELLIPSE
#define     CMG_DRAW_GETSCREEN
#define     CMG_DRAW_LINELIST
#define     CMG_DRAW_LINEWIDTH
#define     CMG_DRAW_ROUNDEDRECTANGLE
#define     CMG_DRAW_SCROLL
#define     CMG_DRAW_TRIANGLE
//
//
//############################################################################
//############################################################################
```

Listing 93 – CMG_DRAW.config

Alle öfter benötigten Basisfunktionen, wie Linien, Pixel, Rechtecke und das Blitten, werden immer kompiliert. Der Anwender kann allerdings bestimmte, erweiterte – aber nicht benötigte – Zeichenfunktionen ausschließen und somit Speicherplatz sparen. Die Liste an ausblendbaren Funktionen steht direkt in der obigen Config-Datei.

9.4. Main, Gemeinsame Komponenten

9.4.1. Main

Zuerst definiert `Main` die globalen Variablen für die DRAW-Schicht. Eine Auflistung der Variablen finden Sie in Abschnitt 9.2.2.

Die `Init`-Funktion initialisiert alle Statusvariablen und setzt einen Standardmodus. Anschließend folgen die Funktionen `Exit(…)`, `PowerManagement(…)` und `ControllerSpecific(…)`, welche jedoch keine Besonderheiten aufweisen.

Es folgt eine Liste der Funktionen, mit welchen der Benutzer die Statusvariablen auslesen und setzen kann:

Funktion	Beschreibung
`GetScreenSize`	Die aktuelle Displayauflösung (X/Y) wird in den übergebenen Adressen abgelegt. Dieser Wert ist nicht immer gleich der physikalischen Auflösung des Displays, sondern kann sich je nach Display-Orientierung (Rotation) ändern.
`SetDisplayOrientation`	Setzt die übergebene Display-Orientiertung (Spiegelung oder Vertauschung der Achsen, Rotation, etc.).
`GetDisplayOrientation`	Liefert die aktuelle Display-Orientierung zurück.
`SetClippingWindow`	Setzt mit Hilfe der zwei übergebenen Punkte das Clipping-Fenster. Intern speichert die Schicht das Clipping-Fenster in LX, RX, TY und BY, wobei die Anfangsbuchstaben den jeweiligen Rand definieren (*engl. left, right, top, bottom*). Die Anpassung an die maximale Größe sowie die Zuordnung der Koordinaten zu den passenden Ecken übernimmt die Funktion.
`GetClippingWindow`	Diese Funktion liefert die aktuellen Grenzen des Clipping-Fensters zurück.
`SetROPMode`	Hier wird nur die eigentliche DRV-Funktion gekapselt und der Wert durchgereicht.

Tabelle 33 – CMG_DRAW_Main.c, Funktionen für die Statusvariablen

Der Benutzer kann die globalen Variablen nicht direkt setzen oder lesen, sondern muß diese Funktionen dafür benutzen. Der Vorteil an der Kapselung ist, daß die Eingabeparameter überprüft, korrigiert und teilweise sortiert werden können.

Eine sehr wichtige Funktion beinhaltet `Main` noch: Das Umwandeln der logischen Koordinaten anhand der eingestellten Display-Orientierung in physikalische Koordinaten:

```
// adjust x y accordingly to the display orientation
_PUBLIC
void _DRAW_AdjustXY( cmg_Coord *piX, cmg_Coord *piY )
{
    // if coord adjustment is prohibited return
    if ( g_DRAW_bProhibitCoordAdjustment )
        return;

    // swap xy
    if ( g_DRAW_uDisplayOrientationMask & DISPLAYORIENTATION_XYSWAP )
        SWAP_TYPE( *piX, *piY, cmg_Coord );

    // mirror x
    if ( g_DRAW_uDisplayOrientationMask & DISPLAYORIENTATION_XMIRROR )
        *piX = g_CTRL_iDisplayWidth - 1 - *piX;

    // mirror y
    if ( g_DRAW_uDisplayOrientationMask & DISPLAYORIENTATION_YMIRROR )
        *piY = g_CTRL_iDisplayHeight - 1 - *piY;
}
```

Listing 94 – CMG_DRAW_Main.c, AdjustXY

Zuerst überprüft die Funktion, ob `bProhibitCoordAdjustment` gesetzt ist und kehrt in diesem Fall ohne Änderung sofort zurück. Diese globale Variable ist normalerweise immer `false`. Ruft jedoch eine Draw-Funktion eine **andere** Draw-Funktion mit bereits umgewandelten Koordinaten auf, so dürfen diese Koordinaten nicht erneut umgewandelt werden, sondern entsprechen bereits den physikalischen Koordinaten. Innerhalb von Draw-Funktionen müssen also weitere Draw-Aufrufe mit `bProhibitCoordAdjustment` gleich `true` aufgerufen werden.

Anschließend nimmt die Funktion die entsprechenden Änderungen vor: Zuerst das Vertauschen der Achsen, danach das Spiegeln der X- als auch der Y-Achse. Zu beachten gilt hier noch, daß die Reihenfolge von Vertauschen und Spiegeln sehr wohl einen Unterschied macht, und man sich die Wirkungsweise genau überlegen muß. Für die normalen Anwendungsfälle, also das Rotieren des Displays, sind schon vordefinierte Makros vorhanden, welche einem diese Überlegungen abnehmen:

Funktion	Beschreibung
`ROTATE90`	`XYSWAP | XMIRROR`
`ROTATE180`	`XMIRROR | YMIRROR`
`ROTATE270`	`XYSWAP | YMIRROR`

Tabelle 34 – Display-Orientierung, Rotation

9.4.2. MainDrawHelpers

Die DRAW-Schicht führt – wie schon vom Beginn des Abschnittes bekannt – zusätzlich zu den Stiften aus der DRV-Schicht die Pinsel ein, welche nicht nur ein-, sondern zweidimensional sind.

Diese Aufgabe kann aber in der DRV-Schicht nur von den Stiften übernommen werden, da sonst keine anderen Möglichkeiten vorhanden sind.

Die Pinsel können realisiert werden, indem man jede Zeile des Pinsels als einzelnen Stift betrachtet. Bei jeder neuen zu zeichnenden Zeile muß der aktuelle Stift auf die nächste Zeile des Pinsels gelegt werden.

Es gibt – wie auch schon bei den Stiften – die Möglichkeit einen einfachen, einfarbigen Pinsel zu verwenden. Dieser nennt sich dann `SolidFillColor`.

Die Daten des aktuellen Stifts, des Pinsels, der konstanten Stiftfarbe und die der konstanten Hintergrundfarbe werden von der DRAW-Schicht lokal zwischengespeichert und können, je nach Bedarf, für die darunterliegende DRV-Schicht konvertiert werden.

Damit sich nicht jede Zeichenfunktion der DRAW-Schicht selbst um die korrekte Umwandlung und Bereitstellung kümmern muß, bietet `MainDrawHelpers` dafür passende, gemeinsame Funktionen an.

Die eigentlichen Zeichenfunktionen müssen nun nur noch vor dem Zeichnen mitteilen, ob sie den aktuellen Stift oder den aktuellen Pinsel verwenden wollen. Der Stift oder der Pinsel kann in diesem Fall auch der mit der konstanten Farbe sein.

Zuerst die Funktion zum Verwenden des aktuellen Stifts:

```
_PROTECTED
void _DRAW_ApplyPen( cmg_Coord iSpoolX )
{
    if ( g_DRAW_uColorMode == DRV_COLORMODE_SOLIDCOLOR )
    {
        // DRV_COLORMODE_SOLIDCOLOR
        CMG_DRV_SetSolidColor( g_DRAW_Color );
    }
    else
    {
        // DRV_COLORMODE_PEN_BRUSH
        CMG_DRV_SetPen( g_DRAW_pPen->pbyPenStartAddr, g_DRAW_pPen->uPenLengthBytes, g_DRAW_pPen-
                        >sfpPenScaleFactor );
        // spool pen
        CMG_DRV_SpoolPen( (cmg_Coord)( g_DRAW_pPen->iPenSpoolPixels + iSpoolX ) );
    }
}
```

Listing 95 – CMG_DRAW_MainDrawHelpers.c, ApplyPen

Die Funktion prüft, ob eine konstante Farbe oder der eigentliche Stift verwendet werden soll und ruft die entsprechenden Funktionen aus DRV auf. Zu erwähnen ist noch der Parameter `iSpoolX`: Jeder Pinsel besitzt bereits intern einen Parameter zum Verschieben des Stiftanfangs. Dieser kann vom Benutzer frei gewählt werden. In der letzten Codezeile sieht man, daß die beiden Spooling-Parameter vor dem Setzen addiert werden.

Wofür ist nun aber das zusätzliche Offset nötig? Stellen Sie sich eine – mit einem Stift gezeichnete – Linie vor. Die Linienfunktion erlaubt das Unterdrücken des ersten Pixels. Ist das der Fall, wäre der Pinsel der restlichen Linie um eben genau diesen einen Punkt verschoben. Noch größere Verschiebungen ergeben sich beim Clipping. Liegt nur ein Teil der Linie im aktiven Fenster, so soll das Ergebnis darin exakt so aussehen, als hätte man die Funktion ohne beschränkten Bereich ausgeführt.

Das Setzen eines Pinsels ist nur wenig aufwendiger: Es beinhaltet zusätzlich die Handhabung mehrerer Zeilen und besteht aus zwei Funktionen: Mit `ApplyBrush(…)` wird die Verwendung eines Pinsels vorbereitet, mit `ApplyBrushNextLine(…)` die aktuelle Zeile gesetzt.

Im Gegensatz zum Stift, welcher sowohl unskalierte als auch skalierte Stifte separat unterstützt um bei den unskalierten einen maximalen Performancegewinn zu erzielen, verwendet die Pinselimplementierung nur die skalierten und berechnet damit auch die unskalierten. Dadurch ergibt sich ein kleiner Geschwindigkeitsverlust, der aber aufgrund der Seltenheit des Aufrufs kaum ins Gewicht fällt. Allerdings spart man sich auch eine Menge Code zum Unterscheiden und Berechnen der sonst unterschiedlichen Fälle.

Zuerst die Initialisierung des Pinsels:

```
_PROTECTED
void _DRAW_ApplyBrush( cmg_Coord iSpoolY )
{
    if ( g_DRAW_uFillColorMode == DRV_COLORMODE_SOLIDCOLOR )
    {
        // DRV_COLORMODE_SOLIDCOLOR
        CMG_DRV_SetSolidColor( g_DRAW_FillColor );
    }
    else
    {
        // DRV_COLORMODE_PEN_BRUSH
        cmg_sfp1616     sfpBrushLines;

        g_DRAW_sfpBrushCurrentLine = ( g_DRAW_pBrush->iBrushSpoolPixelsY + iSpoolY ) * g_DRAW_pBrush-
                        >sfpBrushScaleFactorY;

        // adjust g_DRAW_sfpBrushCurrentLine
        sfpBrushLines = INT_TO_SFP( g_DRAW_pBrush->uBrushLines );
        while ( g_DRAW_sfpBrushCurrentLine < 0 )
            g_DRAW_sfpBrushCurrentLine += sfpBrushLines;
        while ( g_DRAW_sfpBrushCurrentLine >= sfpBrushLines )
            g_DRAW_sfpBrushCurrentLine -= sfpBrushLines;
    }
}
```

Listing 96 – CMG_DRAW_MainDrawHelpers.c, ApplyBrush

Man erkennt wieder die Unterteilung in konstante Füllfarbe und Pinsel. Beim Setzen des Pinsels wird zuerst, unter Beachtung des Spoolings, die Startzeile berechnet und dann, entsprechend den Grenzen, normiert. Auch hier gilt das Prinzip, daß, wegen des Clippings, zusätzlich ein zweiter, interner Spooling-Parameter angegeben werden kann.

Um nun die aktuelle Pinselzeile setzen und verwenden zu können, bzw. zur nächsten weiterzuschalten, muß die Funktion `ApplyBrushNextLine(…)` aufgerufen werden:

```
_PROTECTED
void _DRAW_ApplyBrushNextLine( cmg_Coord iLineSpoolX )
{
    // nothing to do for DRV_COLORMODE_SOLIDCOLOR anymore

    if ( g_DRAW_uFillColorMode == DRV_COLORMODE_PEN_BRUSH )
    {
        // DRV_COLORMODE_PEN_BRUSH
        cmg_sfp1616     sfpBrushLines;

        // get current brush line and set it
        CMG_DRV_SetPen( g_DRAW_pBrush->pbyBrushStartAddr + ( ( (cmg_mu16) SFP_TO_INT(
                            g_DRAW_sfpBrushCurrentLine ) ) * g_DRAW_pBrush->uBrushLineLengthBytes ),
                            g_DRAW_pBrush->uBrushLineLengthBytes, g_DRAW_pBrush->sfpBrushScaleFactorX );

        // spool pen
        CMG_DRV_SpoolPen( (cmg_Coord)(g_DRAW_pBrush->iBrushSpoolPixelsX + iLineSpoolX) );

        // calc next line
        g_DRAW_sfpBrushCurrentLine += g_DRAW_pBrush->sfpBrushScaleFactorY;

        // adjust g_DRAW_sfpBrushCurrentLine
        sfpBrushLines = INT_TO_SFP( g_DRAW_pBrush->uBrushLines );
        while ( g_DRAW_sfpBrushCurrentLine < 0 )
            g_DRAW_sfpBrushCurrentLine += sfpBrushLines;
        while ( g_DRAW_sfpBrushCurrentLine >= sfpBrushLines )
            g_DRAW_sfpBrushCurrentLine -= sfpBrushLines;
    }
}
```

Listing 97 – CMG_DRAW_MainDrawHelpers.c, ApplyBrushNextLine

Bei einer konstanten Füllfarbe ist gar nichts mehr zu tun, nur bei einem Pinsel. Zuerst wird in der DRV-Schicht die aktuelle Zeile des Pinsels gesetzt, danach mit dem angegebenen X-Wert verschoben und anschließend die Werte für die nächste Zeile berechnet und normiert. Somit kann ein Pinsel in jeder Zeile einen anderen X-Spooling-Wert annehmen, was zum Beispiel für Kreise, Ellipsen, aber auch für Dreiecke der Fall ist.

Um die einzelnen DRAW-Zeichenfunktionen beim Clipping zu entlasten, gibt es zwei vordefinierte Zeichenfunktionen mit Clipping-Unterstützung: Einen Pixel setzen oder eine horizontale Linie zeichnen:

```
/*-------------------------------------------------------------------------
| _DRAW_ClippedPixel
|------------------------------------
| draw the next border pixel with clipping support
+-------------------------------------------------------------------------
| @Params:
|   * iX:           x point
|   * iY:           y point
| @Return Value:    none
+-------------------------------------------------------------------------*/
_PROTECTED
void _DRAW_ClippedPixel( cmg_Coord iX, cmg_Coord iY )
{
    // clipping
    if ( ( iX < g_DRAW_iClippingWindow_LX ) || ( iX > g_DRAW_iClippingWindow_RX ) ||
         ( iY < g_DRAW_iClippingWindow_TY ) || ( iY > g_DRAW_iClippingWindow_BY ) )
    {
        // if to use a pen and it is clipped, spool one forward
        if ( g_DRAW_uColorMode == DRV_COLORMODE_PEN_BRUSH )
            CMG_DRV_SpoolPenRelative( 1 );
        return;
    }

    // set pixel
    CMG_DRV_Pixel( iX, iY );
}
```

Listing 98 – CMG_DRAW_MainDrawHelpers.c, ClippedPixel

Hier wird der Pixel nur gezeichnet, wenn er innerhalb des Clipping-Bereichs liegt. Ist das nicht der Fall und wird ein Stift verwendet, so muß dieser jedoch trotzdem um eben diesen einen Pixel weiterverschoben werden, da sich sonst der Rest der Stiftzeile verschieben würde.

Das Zeichnen einer horizontalen Linie ist etwas umfangreicher:

```
/*-------------------------------------------------------------------------
| _DRAW_ClippedHLine
|-----------------------------------
| draw the next fill brush line with clipping support
+-------------------------------------------------------------------------
| @Params:
|   * iXLeft:           left line start
|   * iXRight:          right line stop
|   * iY:               y line
|   * iSpoolLeftmostX:  leftmost point to calc spooling
| @Return Value:    none
+-------------------------------------------------------------------------*/
_PROTECTED
void _DRAW_ClippedHLine( cmg_Coord iXLeft, cmg_Coord iXRight, cmg_Coord iY, cmg_Coord iSpoolLeftmostX )
{
    // update spool position according to x start position
    iSpoolLeftmostX = iXLeft - iSpoolLeftmostX;

    // clipping
    if ( ( iY < g_DRAW_iClippingWindow_TY ) || ( iY > g_DRAW_iClippingWindow_BY ) )
         goto _lAbort;

    // clipping
    if ( g_DRAW_iClippingWindow_LX > iXLeft )
    {
        // update spool (since we must clip)
        iSpoolLeftmostX += g_DRAW_iClippingWindow_LX - iXLeft;
        // set new x
        iXLeft = g_DRAW_iClippingWindow_LX;
    }
    iXRight = MIN( iXRight, g_DRAW_iClippingWindow_RX );

    // nothing to draw anymore?
    if ( ( iXRight - iXLeft ) < 0 )
        goto _lAbort;


    // set next brush line
    _DRAW_ApplyBrushNextLine( iSpoolLeftmostX );

    // draw the line
    CMG_DRV_HLine( iXLeft, iY, (cmg_Coord)( iXRight - iXLeft + 1 ) );
    return;


_lAbort:
    // whe we abort we do have to switch to the next line to prevent the brush alignment
    _DRAW_ApplyBrushNextLine( 0 );
}
```

Listing 99 – CMG_DRAW_MainDrawHelpers.c, ClippedHLine

Der Funktion werden als Parameter die linke und die rechte X-Koordinate sowie die Y-Koordinate der Linie übergeben; zusätzlich dazu noch ein X-Wert, welcher den äußersten, linken Punkt kennzeichnet. Dadurch können auch bei komplizierten Figuren ohne Aufwand richtig ausgerichtete Pinsel verwendet werden.

Zuerst berechnet die Funktion eben diesen X-Spooling-Wert für die aktuelle Zeile. Anschließend wird das Clipping durchgeführt: Wird die linke Seite angepaßt, muß zusätzlich noch der Spooling-Wert mit angepaßt werden.

Falls danach noch etwas zum Zeichnen übrig bleibt, wird die aktuelle Pinselzeile gesetzt und die Linie gezeichnet.

9.4.3. MainDrawstyleColorPenBrush

Diese Komponente besteht aus einer Vielzahl von kleinen Hilfsfunktionen:

Funktion	Beschreibung
`CMG_SetDrawStyle`	Setzt den Zeichenstil. Mögliche Werte sind: `DRAWSTYLE_BORDER` zum Zeichnen nur des Randes mit dem aktuellen Stift, `DRAWSTYLE_FILL` zum Füllen der gesamten Figur mit dem aktuellen Pinsel und die Kombination aus beiden `DRAWSTYLE_BOTH` zum Zeichnen des Randes und anschließendem Ausfüllen des inneren Inhalts. Alle festen Zeichenkörper halten sich an diese Vorgaben.
`CMG_SetColor`	Setzt die übergebene, konstante Farbe für Stifte und wechselt in den konstanten Farbmodus bei den Stiften.
`CMG_SetFillColor`	Pendant dazu für eine konstante Füllfarbe. Wechselt ebenfalls, für den Pinsel, in den konstanten Farbmodus.
`CMG_SetPen`	Setzt den übergebenen Stift. Der Datentyp ist der in `CMG_DRAW.h` definierte Typ `cmg_Pen`.
`CMG_SetBrush`	Setzt den übergebenen Pinsel. Der Datentyp ist der in `CMG_DRAW.h` definierte Typ `cmg_Brush`.
`CMG_CreatePen`	Die folgenden Funktionen erstellen aus den übergebenen Parametern den gewünschten, benutzerdefinierten DRAW-Typ. Bei `CreatePen` muß nur eine Startadresse und die Anzahl der Bytes für den Stift angegeben werden.
`CMG_CreatePenEx`	Bei der erweiterten Ex-Funktion muß zusätzlich noch der Spooling-Wert und der Skalierungsfaktor angegeben werden. Er ist diesmal jedoch keine Festkommazahl, sondern ein `float`, was die Berechnung im Programm für den Anwender erleichtert.
`CMG_CreateBrush`	Mit `CreateBrush` wird – wie der Name schon aussagt – ein Pinsel erstellt. Als Parameter benötigt die Funktion, zusätzlich zu der Startadresse und der Anzahl an Bytes pro Zeile, noch die Gesamtanzahl der vorhandenen Zeilen.
`CMG_CreateBrushEx`	Die erweiterte Version erwartet zusätzlich noch die Spooling-Werte in beiden Richtungen sowie die Skalierungsfaktoren, ebenfalls in beiden Richtungen.

Tabelle 35 – CMG_DRAW_MainDrawstyleColorPenBrush.c, Funktionen

9.5. Aufbau der Beispiele

In den letzten Kapiteln wurden nur theoretische Sachverhalte und Zusammenhänge beschrieben. Das soll sich im Rest des Kapitels, in welchem die einzelnen Zeichenfunktionen an der Reihe sind, ändern. Zu jeder Zeichenfunktion werden Sie kurze Beispiele finden, um das Verständnis des Funktionsablaufs sowie die Verwendung jener Funktionen einfacher zu gestalten.

Die Beispiele sind alle nach demselben Muster aufgebaut:

```
#include "CMG/CMG.h"

int main( void )
{
    cmg_Result  iResult;
    cmg_Coord   iScreenSizeX, iScreenSizeY;

    iResult = CMG_DRAW_Init();
    if ( iResult != CMG_OK )
    {
        printf( " *** ERROR 0x%02x ***\n\n", iResult );
        return -1;
    }
```

```
    CMG_GetScreenSize( &iScreenSizeX, &iScreenSizeY );

    [...]

    _getch();

    CMG_DRAW_Exit();
    return 0;
}
```

Listing 100 – DRAW Beispielaufbau

CMG wird initialisiert, der Rückgabewert überprüft und die logische Bildschirmauflösung ermittelt. Danach steht der Beispielcode aus den nächsten Beispielen und zum Ende wird nach einem Tastendruck CMG wieder beendet.

Um Stifte, Pinsel und Bilder benutzen zu können, sind ein paar Beispiele vordefiniert. In den Abbildungen zum Grafikobjekt befindet sich oben immer das einzelne Objekt und darunter die normale Benutzung mit größeren Längen oder Flächen.

- **Stifte:**
  Ein Stift, bestehend aus 16 Pixeln, also 2 Bytes groß:

```
// pen dash dot dot - 16 (2 bytes)
cmg_u8  dataPenDashDotDot[] = { 0xfc, 0xcc };
```

Listing 101 – DRAW Beispielstift

Abbildung 52 – DRAW Beispielstift

- **Pinsel:**
  Es sind gleich drei Pinsel definiert:

```
// brush - 50% - 8x2 (1 byte)
cmg_u8  dataBrush50[] = { 0xaa, 0x55 };

// brush2 - light gray- 24x6 (3 bytes)
cmg_u8  dataBrushLight[] = { 0x92, 0x49, 0x24, 0x00, 0x00, 0x00, 0x24, 0x92, 0x49, 0x00, 0x00, 0x00,
                             0x49, 0x24, 0x92, 0x00, 0x00, 0x00 };

// HelloWorld - 41x6 (6 bytes)
cmg_u8  dataHelloWorld[] = { 0xae, 0x88, 0x42, 0x93, 0xa3, 0x00, 0xa8, 0x88, 0xa2, 0xaa, 0xa2, 0x80,
                             0xee, 0x88, 0xa3, 0xab, 0x22, 0x80, 0xa8, 0x88, 0xa3, 0xaa, 0xa2, 0x80,
                             0xae, 0xee, 0x42, 0x92, 0xbb, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
```

Listing 102 – DRAW Beispielpinsel

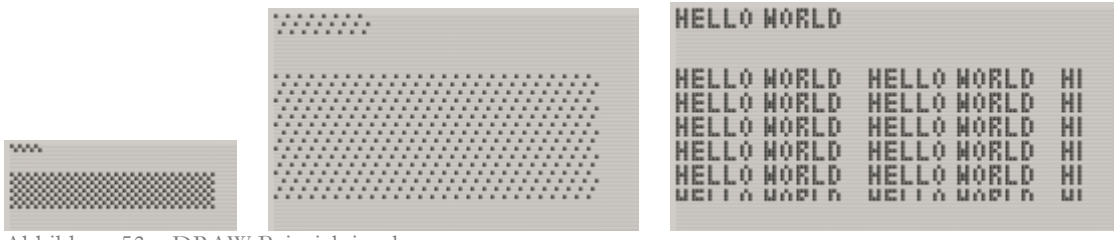


Abbildung 53 – DRAW Beispielpinsel

Diese Elemente werden wie folgt initialisiert und stehen damit den Beispielen zur Verfügung:

```
    cmg_Pen     PenDashDotDot;
    cmg_Brush   Brush50;
    cmg_Brush   BrushLight;
    cmg_Brush   BrushHelloWorld;

    CMG_CreatePen( &PenDashDotDot, dataPenDashDotDot, 2 );
    CMG_CreateBrush( &Brush50, dataBrush50, 1, 2 );
    CMG_CreateBrush( &BrushLight, dataBrushLight, 3, 6 );
    CMG_CreateBrush( &BrushHelloWorld, dataHelloWorld, 6, 6 );
```

Listing 103 – DRAW Beispielinitialisierung

9.6. PIXEL

Das Zeichnen eines Pixels ist die einfachste Grafikfunktion der DRAW-Schicht:

```
/*****************************************************************************
| CMG_DRV_SetPixel
|-----------------------------------
| Set a pixel at x/y
+----------------------------------------------------------------------------
| @Params:
|    * iX:           x coord
|    * iY:           y coord
| @Return Value:     none
+*****************************************************************************/
_PUBLIC
void CMG_Pixel( cmg_Coord iX, cmg_Coord iY )
{
    // display orientation
    _DRAW_AdjustXY( &iX, &iY );

    // clipping
    if ( ( iX < g_DRAW_iClippingWindow_LX ) || ( iX > g_DRAW_iClippingWindow_RX ) ||
         ( iY < g_DRAW_iClippingWindow_TY ) || ( iY > g_DRAW_iClippingWindow_BY ) )
         return;

    _DRAW_ApplyPen( 0 );

    CMG_DRV_Pixel( iX, iY );
}
```

Listing 104 – CMG_DRAW_Pixel.c

Zuerst führt die Funktion das immer notwendige Konvertieren der logischen Koordinaten in die physikalischen aus. Anschließend wird überprüft, ob der Pixel außerhalb des Clipping-Fensters liegt. Ist das der Fall, ist die Funktion auch schon beendet.

Ansonsten wird der aktuelle Stift gesetzt und der Pixel gezeichnet.

Beispiel:
Ein sehr einfaches Beispiel: Eine konstante Farbe und ein einzelner Pixel:

```
    CMG_SetColor( 1 );
    CMG_Pixel( 10, 10 );
```

Listing 105 – DRAW, Pixel, Beispiel

Das Ergebnis ist eher unspektakulär:

Abbildung 54 – DRAW, Pixel, Beispiel

9.7. LINIEN

Linien sind eine der wichtigsten und meistgenutzten grafischen Elemente. Neben den Ellipsen und Dreiecken zählen sie aber auch zu den komplexesten.

9.7.1. NORMALE LINIE

Es geht darum, einen möglichst performanten Algorithmus zu verwenden. Es gibt schon viel Literatur und noch mehr Überlegungen über die verschiedenen Arten eine Linie zu zeichnen und deren Vor- und Nachteile. Ich habe einige Methoden ausführlich getestet, war aber mit keiner ganz zufrieden. Entweder ließ die Implementierung starke Zweifel aufkommen, war schlichtweg falsch oder die Performance war eher in der Region miserabel. Ich habe dann die Entscheidung getroffen, den Algorithmus komplett neu zu implementieren – auf Basis des Algorithmus' von Bresenham [Bres1, Bres2], welcher mir am besten und schnellsten erscheint.

In meiner Version benötigt der Algorithmus zur Initialisierung nur Additionen, Subtraktionen, einfache Vergleichsoperationen und eine Shift-Operation. Die innere Schleife benötigt sogar nur drei bis fünf arithmetische Operationen (Addition oder Subtraktion) und nur einen Vergleich. Auch die Erweiterung vom ursprünglichen Oktanten von 0-45° auf beliebige Richtungen benötigt nur sehr wenig zusätzlichen Aufwand.

Eine weitere Möglichkeit zur Steigerung der Performance ist das Erkennen von horizontalen, sowie vertikalen Linen und das Zeichnen dieser mit den optimierten Versionen aus der DRV-Schicht.

CMG bietet zwei Funktionen zum Zeichnen einer einzelnen, einfachen Linie: `Line(…)` und `LineEx(…)`. Der Unterschied ist, daß bei der erweiterten Version zusätzlich zu den zwei Koordinaten noch ein Flag mit dem Namen `uSuppressPixelMask` übergeben wird. In diesem Flag steht, ob der erste bzw. letzte Pixel unterdrückt, also nicht gezeichnet werden soll. Diese Funktion ist zum Beispiel dann wichtig, wenn eigentlich durchgängige Linien bis zu einem Punkt gezeichnet werden und die nächste Linie wieder an genau diesem Punkt weitergeht. Dann wird der Pixel an dem Punkt zweimal gezeichnet, was sowohl bei verschiedenen ROP-Modes, als auch mit dem Spooling der Stifte Probleme erzeugen kann.

Die einfache Linie ist ein simples Makro, welches die Ex-Version mit dem Parameter `LINESUPPRESS_NONE` aufruft.

`LineEx(…)` hat es dafür in sich: Zu Beginn werden, wie immer, die logischen Koordinaten angepaßt und anschließend in `iClipSpool` die Anzahl der Pixel berechnet, um die der Stift verschoben werden muß. Danach teilt sich die Funktion – je nach Linienart – in drei große Blöcke auf: Entweder in eine horizontale oder in eine vertikale Line, dann werden die optimierten Versionen aus DRV verwendet – oder in keine der beiden Varianten, dann wird der Bresenham-Algorithmus verwendet, der allerdings in eine separate Funktion ausgelagert ist:

```
/*****************************************************************************
| CMG_Line
|-----------------------------------
| draw a line
+----------------------------------------------------------------------------
| @Params:
|    * iX1:          x1 coord
|    * iY1:          y1 coord
|    * iX2:          x2 coord
|    * iY2:          y2 coord
|    * bySuppressPixelMask:  hide first and/or last pixel
| @Return Value:    none
+*****************************************************************************/
```

```
_PUBLIC
void CMG_LineEx( cmg_Coord iX1, cmg_Coord iY1, cmg_Coord iX2, cmg_Coord iY2, cmg_u8 uSuppressPixelMask )
{
    cmg_Coord   idX, idY;
    cmg_Coord   iClipSpool;

    // display orientation
    _DRAW_AdjustXY( &iX1, &iY1 );
    _DRAW_AdjustXY( &iX2, &iY2 );


    // if to suppress first pixel we must spool one more
    iClipSpool = 0;
    if ( uSuppressPixelMask & LINESUPPRESS_FIRST )
        iClipSpool++;


    // get deltas
    idX = iX2 - iX1;
    idY = iY2 - iY1;


    // HLine ?
    if ( idY == 0 )
    {
        // clip y
        if ( ( iY1 < g_DRAW_iClippingWindow_TY ) || ( iY1 > g_DRAW_iClippingWindow_BY ) )
            return;

        // sort x values and suppress pixels
        if ( idX >= 0 )
        {
            // handle suppress pixels
            if ( uSuppressPixelMask & LINESUPPRESS_FIRST )
                iX1++;
            if ( uSuppressPixelMask & LINESUPPRESS_LAST )
                iX2--;
        }
        else
        {
            // x1 > x2 (wrong direction)
            // swap x1/x2 (y2 is temp)
            iY2 = iX2;
            iX2 = iX1;
            iX1 = iY2;

            // handle suppress pixels (must be inverted)
            if ( uSuppressPixelMask & LINESUPPRESS_FIRST )
                iX2--;
            if ( uSuppressPixelMask & LINESUPPRESS_LAST )
                iX1++;
        }

        // clip
        if ( g_DRAW_iClippingWindow_LX > iX1 )
        {
            // update spool (since we must clip)
            iClipSpool += g_DRAW_iClippingWindow_LX - iX1;
            // set new x
            iX1 = g_DRAW_iClippingWindow_LX;
        }
        iX2 = MIN( iX2, g_DRAW_iClippingWindow_RX );

        // recalc dx
        idX = iX2 - iX1;

        // nothing to draw anymore?
        if ( idX < 0 )
            return;

        // apply pen
        _DRAW_ApplyPen( iClipSpool );

        // only one pixel?
        if ( idX == 0 )
        {
            CMG_DRV_Pixel( iX1, iY1 );
            return;
        }

        // draw HLine (dx + 1 pixels)
        CMG_DRV_HLine( iX1, iY1, (cmg_Coord)( idX + 1 ) );

        return;
    }


    // VLine ?
    if ( idX == 0 )
    {
        // clip x
        if ( ( iX1 < g_DRAW_iClippingWindow_LX ) || ( iX1 > g_DRAW_iClippingWindow_RX ) )
            return;
```

```
        // sort y values and suppress pixels
        if ( idY >= 0 )
        {
            // handle suppress pixels
            if ( uSuppressPixelMask & LINESUPPRESS_FIRST )
                iY1++;
            if ( uSuppressPixelMask & LINESUPPRESS_LAST )
                iY2--;
        }
        else
        {
            // y1 > y2 (wrong direction)
            // swap y1/y2 (x2 is temp)
            iX2 = iY2;
            iY2 = iY1;
            iY1 = iX2;

            // handle suppress pixels (must be inverted)
            if ( uSuppressPixelMask & LINESUPPRESS_FIRST )
                iY2--;
            if ( uSuppressPixelMask & LINESUPPRESS_LAST )
                iY1++;
        }

        // clip
        if ( g_DRAW_iClippingWindow_TY > iY1 )
        {
            // update spool (since we must clip)
            iClipSpool += g_DRAW_iClippingWindow_TY - iY1;
            // set new x
            iY1 = g_DRAW_iClippingWindow_TY;
        }
        iY2 = MIN( iY2, g_DRAW_iClippingWindow_BY );

        // recalc dx
        idY = iY2 - iY1;

        // nothing to draw anymore?
        if ( idY < 0 )
            return;

        // apply pen
        _DRAW_ApplyPen( iClipSpool );

        // only one pixel?
        if ( idY == 0 )
        {
            CMG_DRV_Pixel( iX1, iY1 );
            return;
        }

        // draw HLine (dx + 1 pixels)
        CMG_DRV_VLine( iX1, iY1, (cmg_Coord)( idY + 1 ) );

        return;
    }


    // draw normal line
    _DRAW_BresenhamLine( iX1, iY1, iX2, iY2, uSuppressPixelMask, iClipSpool );
}
```

Listing 106 – CMG_DRAW_Line.c, LineEx

Hat die Funktion eine horizontale oder vertikale Linie festgestellt (`dx` oder `dy` gleich Null), so geschehen folgenden Schritte: Zuerst wird überprüft, ob die Linie überhaupt im aktuellen Clipping-Fenster liegt und gezeichnet werden muß. Ist das der Fall, werden die Koordinaten sortiert, so daß die linke X-Koordinate bzw. die obere Y-Koordinate einen kleineren Wert besitzt. Gleichzeitig werden die unterdrückten Start- oder Stoppunkte berechnet und das Spooling aktualisiert. Danach wird die lange Achse an dem Clipping-Bereich abgeschnitten und die neue Breite bzw. Höhe berechnet. Sollte diese kleiner als Null sein, so lag die Linie komplett außerhalb und die Funktion ist beendet. Ab hier ist sicher, daß mindestens ein Teilstück der Linie sichtbar ist und gezeichnet werden muß. Der aktuelle Stift wird mit der zuständigen Hilfsfunktion gesetzt. Ist die oben berechnete Breite bzw. Höhe der Linie genau Null, so ist die Linie nur noch einen Pixel lang und eben dieser wird gezeichnet. Ansonsten wird die Funktion `HLine(…)` bzw. `VLine(…)` der DRV-Schicht aufgerufen und die Linie endlich gezeichnet.

War die Linie weder horizontal noch vertikal, so kommt der Algorithmus von Bresenham zum Einsatz: Die Funktion ist mit `_FORCE_INLINE` eingebettet und nur deshalb ausgelagert, damit der Code übersichtlicher wird:

```
/*-------------------------------------------------------------------------
| _DRAW_BresenhamLine
|-----------------------------------
| optimized line algorithm (ideas from Bresenham's algorithm)
+-------------------------------------------------------------------------
| @Params:
|   * iX1           x1 point (start)
|   * iY1           y1 point (start)
|   * iX2           x2 point (stop)
|   * iY2           y2 point (stop)
|   * bySuppressPixelMask   mask with wich pixels should be suppressed
| @Return Value:    none
+-------------------------------------------------------------------------*/
/* ### UNIVERSAL LINE ALGORITHM ### */
_PRIVATE
_INLINE_FORCE
void _DRAW_BresenhamLine( cmg_Coord iX1, cmg_Coord iY1, cmg_Coord iX2, cmg_Coord iY2, cmg_u8
                          bySuppressPixelMask, cmg_Coord iClipSpool )
{
    cmg_DRAW_Line_Data  LineData;                       // local line data
    cmg_bool            bNeedClipping, bInVisibleWindow;  // clipping helpers

    // . . . . . . . . . . . . . . . . . . .
    // init line loop
    _Line_Loop_Init( &LineData, iX1, iY1, iX2, iY2 );

    // . . . . . . . . . . . . . . . . . . .
    // init clipping
    // first assume best case (no clipping and start pos in visible window)
    bNeedClipping = false;
    bInVisibleWindow = true;
    // do we need clipping ?
    if ( ( iX1 < g_DRAW_iClippingWindow_LX ) || ( iX1 > g_DRAW_iClippingWindow_RX ) || ( iY1 <
                          g_DRAW_iClippingWindow_TY ) || ( iY1 > g_DRAW_iClippingWindow_BY ) )
    {
        bNeedClipping = true;
        bInVisibleWindow = false;
    }
    if ( ( iX2 < g_DRAW_iClippingWindow_LX ) || ( iX2 > g_DRAW_iClippingWindow_RX ) || ( iY2 <
                          g_DRAW_iClippingWindow_TY ) || ( iY2 > g_DRAW_iClippingWindow_BY ) )
    {
        bNeedClipping = true;
    }

    // . . . . . . . . . . . . . . . . . . .
    // pixel suppressing
    // suppress last pixel ?
    if ( bySuppressPixelMask & LINESUPPRESS_LAST )
        if ( !LineData.iCount-- )
            return;

    // suppress first pixel ?
    if ( bySuppressPixelMask & LINESUPPRESS_FIRST )
    {
        _DRAW_ApplyPen( 1 );
        goto _SuppressFirstPixel;
    }

    // . . . . . . . . . . . . . . . . . . .
    // apply pen
    _DRAW_ApplyPen( iClipSpool );

    // . . . . . . . . . . . . . . . . . . .
    // main pixel loop
    while ( 1 )
    {
        // draw pixel
        if ( !bNeedClipping )
        {
            CMG_DRV_Pixel( LineData.iX, LineData.iY );
        }
        else
        {
            // if we don't found out before: check if we are in the visible window
            if ( !bInVisibleWindow )
            {
                // if we are still outside
                if ( ( LineData.iX < g_DRAW_iClippingWindow_LX ) || ( LineData.iX >
                          g_DRAW_iClippingWindow_RX ) || ( LineData.iY < g_DRAW_iClippingWindow_TY ) || (
                          LineData.iY > g_DRAW_iClippingWindow_BY ) )
                {
                    iClipSpool++;
                }
                // if we get inside now
                else
                {
                    // enable drawing
                    bInVisibleWindow = true;
                    // spool pen correctly
                    _DRAW_ApplyPen( iClipSpool );
                }
            }

            // if we are in the visible window
            if ( bInVisibleWindow )
            {
                // if we get outside the visible window quit
                if ( ( LineData.iX < g_DRAW_iClippingWindow_LX ) || ( LineData.iX >
                          g_DRAW_iClippingWindow_RX ) || ( LineData.iY < g_DRAW_iClippingWindow_TY ) || (
                          LineData.iY > g_DRAW_iClippingWindow_BY ) )
                    return;
```

```
                // we are inside our clipping window
                CMG_DRV_Pixel( LineData.iX, LineData.iY );
            }
        }

_SuppressFirstPixel:
        // line end condition
        if ( LineData.iCount <= 1 )
            break;

        // calc next line pixel
        _Line_Loop( &LineData );
    }
}
```

Listing 107 – CMG_DRAW_Line.c, BresenhamLine

Zuerst wird der Bresenham-Algorithmus mit `_Line_Loop_Init(…)` initialisiert.

Es existieren zwei lokale Boolean-Variablen: `bNeedClipping` und `bInVisibleWindow`. In `bNeedClipping` steht, ob mindestens einer der beiden Punkte außerhalb des Clipping-Bereichs liegt. Dann muß später in der Schleife das Clipping beachtet werden. In `bInVisibleWindow` steht, ob sich der aktuelle Punkt innerhalb des sichtbaren Bereichs befindet. Zu Beginn ist das der Startpunkt.

Anschließend behandelt die Funktion unterdrückte Endpixel und setzt den richtigen Stift.

Die innere Schleife besteht aus drei Teilen: Im Ersten wird der nächste Pixel behandelt und gegebenenfalls gezeichnet, im Zweiten die Anzahl der noch zu zeichnenden Pixel überprüft und heruntergezählt und im Dritten der Bresenham-Algorithmus zum Berechnen der neuen Koordinaten des nächsten Pixels mit `_Line_Loop(…)` aufgerufen.

Falls kein Clipping stattfinden muß, `bNeedClipping` also `false` ist, so ist das Zeichnen der einzelnen Pixel aus dem ersten Teil sehr einfach. Es wird nur die Pixel-Funktion mit der aktuellen Koordinate aufgerufen.

Muß jedoch Clipping stattfinden, gibt es noch eine zusätzliche Unterscheidung: Ist `bInVisibleWindow` noch nicht wahr, wird nur intern der Spool-Wert hochgezählt. Dies geschieht solange, bis die Linie entweder zu Ende ist – dann wird die Funktion beendet – oder die Linie in den sichtbaren Bereich hinein kommt – dann wird der berechnete Spool-Wert angewandt und `bInVisibleWindow` auf wahr gesetzt. Ist diese Variable endlich wahr, wird solange ein Pixel gezeichnet, bis die Koordinaten wieder außerhalb liegen oder die Linie zu Ende ist.

Um diesen schwierigen Stoff besser verstehen zu können, soll die Grafik nach der Erklärung des Bresenham-Algorithmus' beitragen.

Zuerst kurz zum eigentlichen Algorithmus – zu Beginn muß dieser mit den Koordinaten initialisiert werden:

```
/*-----------------------------------------------------------------------------
| _Line_Loop_Init
|-----------------------------------
| init start positions and internal variables
| must be done before EACH loop
+----------------------------------------------------------------------------
| @Params:
|    * pLineData:    local line data structure
|    * iX1           x1 point (start)
|    * iY1           y1 point (start)
|    * iX2           x2 point (stop)
|    * iY2           y2 point (stop)
| @Return Value:    none
+----------------------------------------------------------------------------*/
_PROTECTED
void _Line_Loop_Init( cmg_DRAW_Line_Data *pLineData, cmg_Coord iX1, cmg_Coord iY1, cmg_Coord iX2, cmg_Coord
                          iY2 )
{
```

```
    // . . . . . . . . . . . . . . . . . . . . .
    // set internal variables

    // swap x/y coords if dy > dx
    {
        // caluclate preliminary deltas (speedup only)
        cmg_Coord   iDeltaX = iX2 - iX1;
        cmg_Coord   iDeltaY = iY2 - iY1;

        // calculate: "bSwapXY = abs( iDeltaY ) > abs( iDeltaX );"  and swap it if its true
        if ( ( pLineData->bSwapXY = ( ( iDeltaY >= 0 ) ? iDeltaY : -iDeltaY ) > ( ( iDeltaX >= 0 ) ?
                            iDeltaX : -iDeltaX ) ) != 0 )
        {
            // swap x and y (use iDeltaX as temp)
            SWAP( iX1, iY1, iDeltaX );
            SWAP( iX2, iY2, iDeltaX );
        }

        // explanation:
        // if (bSwapXY == false):   Long->X, Short->Y
        //    (bSwapXY == true ):   Long->Y, Short->X
    }

    // caluclate deltas and increment
    // long dir
    if ( iX1 <= iX2 )
    {
        pLineData->iDeltaLong = iX2 - iX1;
        pLineData->iIncLong   = 1;
    }
    else
    {
        pLineData->iDeltaLong = iX1 - iX2;
        pLineData->iIncLong   = -1;
    }

    // short dir
    if ( iY1 <= iY2)
    {
        pLineData->iDeltaShort = iY2 - iY1;
        pLineData->iIncShort   = 1;
    }
    else
    {
        pLineData->iDeltaShort = iY1 - iY2;
        pLineData->iIncShort   = -1;
    }

    // calc start drift value
    pLineData->iDrift = pLineData->iDeltaLong >> 1;

    // calc number of pixels
    pLineData->iCount = pLineData->iDeltaLong + 1;

    // set positions
    pLineData->iLong  = iX1;
    pLineData->iShort = iY1;

    // . . . . . . . . . . . . . . . . . . . . .
    // set start positions
    if ( !pLineData->bSwapXY )
    {
        pLineData->iX = iX1;
        pLineData->iY = iY1;
    }
    else
    {
        pLineData->iX = iY1;
        pLineData->iY = iX1;
    }
}
```

Listing 108 – CMG_DRAW_Line.c, _Line_Loop_Init

Hier werden nur die einzelnen Felder für die späteren Schleifen initialisiert. Außerdem ist ein Großteil des Codes für die Unterscheidung der Richtung der Linie und die Anpassung der Variablen daran zuständig.

Die eigentliche Schleife sieht so aus:

```
/*---------------------------------------------------------------------------
| _Line_Loop
|----------------------------------
| inner line loop to update to the next pixel
| see example above to see usage
+--------------------------------------------------------------------------
| @Params:
|   * pLineData:    local line data structure
| @Return Value:    none
+-------------------------------------------------------------------------*/
_PROTECTED
void _Line_Loop( cmg_DRAW_Line_Data *pLineData )
{
    // inner line loop to update to the next pixel
```

```
    // long dir adjust (always)
    pLineData->iLong  += pLineData->iIncLong;

    // short dir adjust (only step if needed)
    pLineData->iDrift -= pLineData->iDeltaShort;
    if ( pLineData->iDrift < 0 )
    {
        pLineData->iShort += pLineData->iIncShort;
        pLineData->iDrift += pLineData->iDeltaLong;
    }

    // update positions depending on swap state
    if ( !pLineData->bSwapXY )
    {
        // x is long dir
        pLineData->iX = pLineData->iLong;
        pLineData->iY = pLineData->iShort;
    }
    else
    {
        // y is long dir
        pLineData->iX = pLineData->iShort;
        pLineData->iY = pLineData->iLong;
    }

    // dec count
    pLineData->iCount--;
}
```

Listing 109 – CMG_DRAW_Line.c, _Line_Loop

Hier werden die Felder angepaßt und die neuen Werte berechnet sowie die fertigen Koordinaten `iX` und `iY` richtig gesetzt.

Zum nochmaligen Nachvollziehen und besseren Verständnis des ganzen Linienalgorithmus' nachfolgend die komplette Übersicht. Die einzelnen Funktionsblöcke bei der Bresenham-Linie spiegeln nicht exakt den Codeaufbau wieder, sondern erklären die Funktion. Der Code ist in ein paar Punkten auf Geschwindigkeit und Codegröße optimiert und verwendet nur eine große Schleife, anstatt der in der Abbildung aufgeführten kleineren. Die Funktion ist aber exakt gleich:

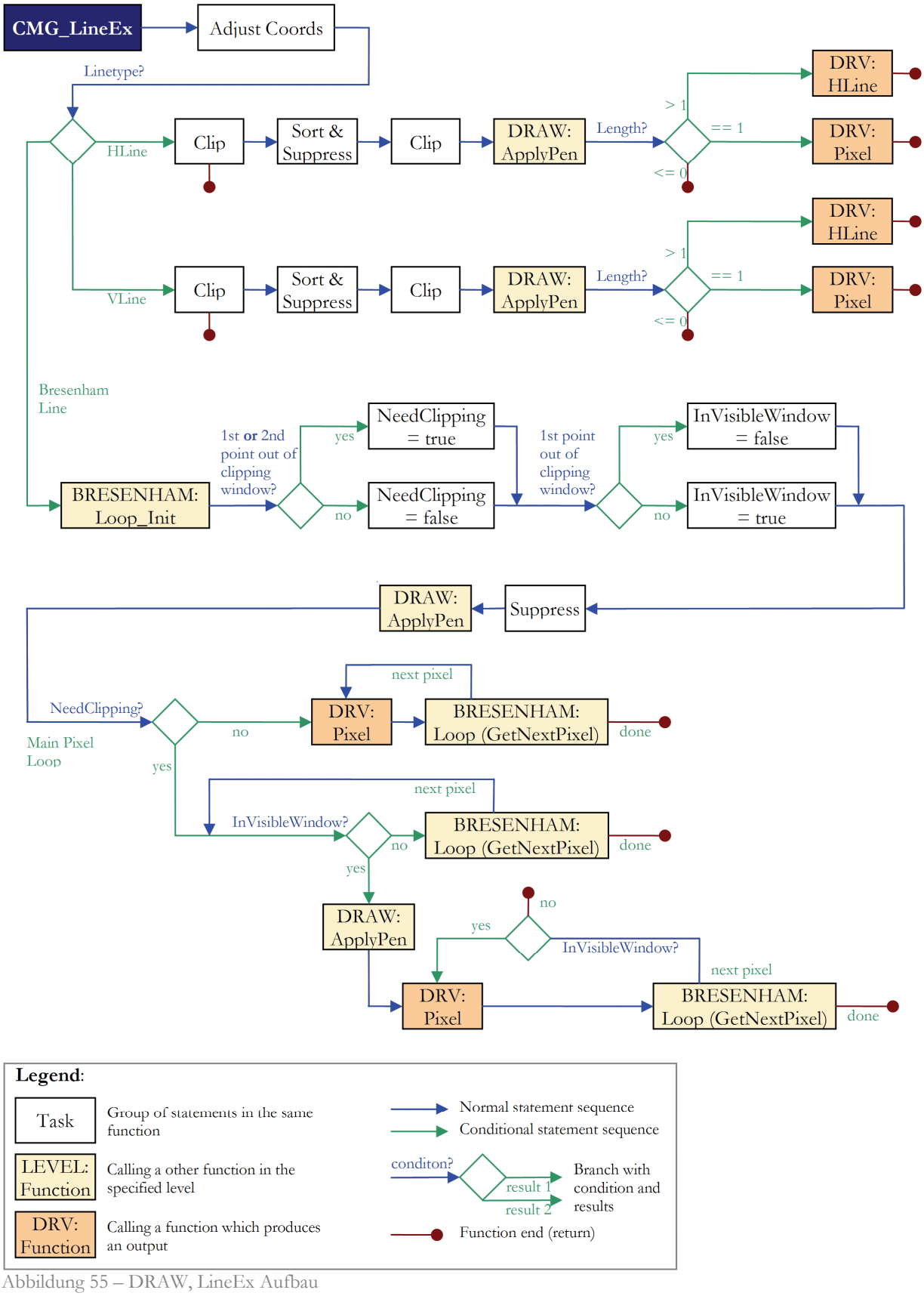


Abbildung 55 – DRAW, LineEx Aufbau

Beispiel:
Ein umfangreiches Beispiel mit Linien:

```
    CMG_SetColor( 1 );

    CMG_Line( 0, 10, 20, 10 );
    CMG_Line( 10, 0, 10, 20 );
    CMG_Line( 5, 5, 15, 15 );
    CMG_Line( 5, 15, 15, 5 );

    CMG_Line( 30, 0, 100, 0 );
    CMG_Line( 30, 0, 100, 10 );
    CMG_Line( 30, 0, 100, 20 );
    CMG_Line( 30, 0, 100, 30 );
    CMG_Line( 30, 0, 60, 30 );
    CMG_Line( 30, 0, 30, 30 );

    CMG_LineEx( 120, 0, 149, 0, LINESUPPRESS_NONE );
    CMG_LineEx( 120, 3, 149, 3, LINESUPPRESS_FIRST );
    CMG_LineEx( 120, 6, 149, 6, LINESUPPRESS_LAST );
    CMG_LineEx( 120, 9, 149, 9, LINESUPPRESS_FIRST | LINESUPPRESS_LAST );

    CMG_Line( 160, 30, 200, 0 );
    CMG_SetClippingWindow( 170, 10, 190, 20 );
    CMG_Line( 160, 32, 200, 2 );
    CMG_Line( 160, 34, 200, 4 );
```

Listing 110 – DRAW, Linien, Beispiel 1

In diesem Beispiel wird immer die konstante Farbe 1, also Schwarz, verwendet. Der erste Block zeichnet alle Linientypen; eine horizontale sowie eine vertikale, optimierte Linie und zwei diagonale Linien mit dem Bresenham-Algorithmus. Der zweite Block zeigt Linien mit verschiedenen Steigungen. Der dritte Block demonstriert die Verwendung der Unterdrückungs-Flags. Achten Sie genau auf die Enden der Linien! Der vierte und letzte Teil zeichnet drei, jeweils exakt untereinander liegende, Linien. Nach der ersten Linie wird aber ein Clipping-Fenster gesetzt. Die beiden restlichen Linien werden deshalb an den Grenzen abgeschnitten:

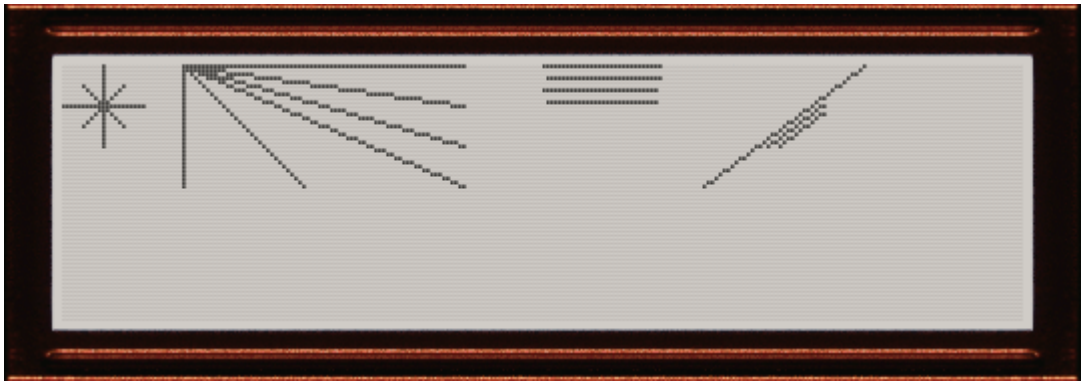


Abbildung 56 – DRAW, Linien, Beispiel 1

Nimmt man nun, anstelle der konstanten Farbe, den zuvor definierten Stift `PenDashDotDot` mit dem Aufruf `CMG_SetPen( &PenDashDotDot )`, so entsteht diese Ausgabe:

Abbildung 57 – DRAW, Linien, Beispiel 2

Ein letztes Beispiel soll die Verwendung des XOR-ROP-Modus zeigen: Vor dem ursprünglichen Beispielcode wird noch folgende Funktion ausgeführt:

```
    CMG_SetROPMode( ROPMODE_XOR );
```

Listing 111 – DRAW, Linien, Beispiel 2

Im Ergebnis sieht man, daß die Pixel, welche öfter gezeichnet werden, ihren Farbwert bei jedem Zeichenvorgang invertieren. In der Mitte des Sterns ist wieder ein weißer Pixel, denn er wurde insgesamt vier Mal gezeichnet. Auch im zweiten Block werden links oben teilweise Pixel mehrfach gezeichnet:

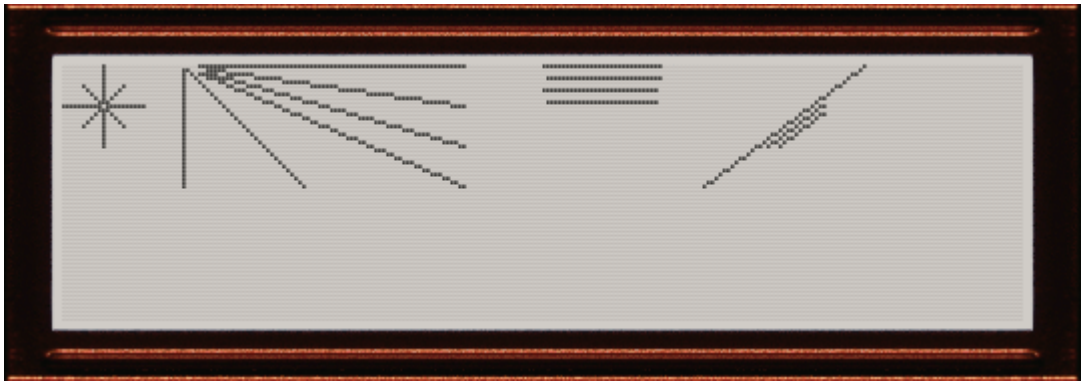


Abbildung 58 – DRAW, Linien, Beispiel 3

9.7.2. LINIENLISTE

Mit einer LineList läßt sich eine verbundene Linie mit beliebig vielen Zwischenpunkten in einem Aufruf zeichnen. Optional kann die Linie auch geschlossen werden, der letzte Punkt also zusätzlich mit dem ersten verbunden werden:

```
/*****************************************************************************
| CMG_LineList
|-----------------------------------
| draw a continous line from a list
+-----------------------------------------------------------------------------
| @Params:
|   * paiCoords:        pointer to an array with points. Each point is an
|                       array of cmg_Coord[2] - [0] = x and [1] = y. This
|                       means you can access it like paiCoords[pointnr][0/1]
|   * uNumberOfPoints:  number of points in array. 1 = one pixel,
|                       2 = one line, 3 = two lines (0-1, 1-2)
|   * bCloseList:       if to close the list. This means to draw the line
|                       from the last point to the first:
|                       -> line( point[ uNumberOfPoints - 1 ], point[ 0 ] )
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_LineList( cmg_Coord (*paiCoords)[2], cmg_mu16 uNumberOfPoints, cmg_bool bCloseList )
{
    cmg_mu16    uNr;

    // nothing to draw
    if ( uNumberOfPoints <= 0 )
        return;

    // only one point
    if ( uNumberOfPoints == 1 )
    {
        CMG_Pixel( paiCoords[0][0], paiCoords[0][1] );
        return;
    }

    // draw all mid parts
    for ( uNr = 1; uNr < ( uNumberOfPoints - 1 ); uNr++ )
        CMG_LineEx( paiCoords[ uNr - 1 ][0], paiCoords[ uNr - 1 ][1], paiCoords[ uNr ][0], paiCoords[ uNr
                            ][1], LINESUPPRESS_LAST );
```

```
    // close list (connect last point with first point)
    if ( bCloseList )
    {
        // draw last part in list
        CMG_LineEx( paiCoords[ uNr - 1 ][0], paiCoords[ uNr - 1 ][1], paiCoords[ uNr ][0], paiCoords[ uNr
                    ][1], LINESUPPRESS_LAST );
        // connect last and fist one
        CMG_LineEx( paiCoords[ uNr ][0], paiCoords[ uNr ][1], paiCoords[0][0], paiCoords[0][1],
                    LINESUPPRESS_LAST );
    }
    else
    {
        // draw last part and don't suppress last pixel!
        CMG_LineEx( paiCoords[ uNr - 1 ][0], paiCoords[ uNr - 1 ][1], paiCoords[ uNr ][0], paiCoords[ uNr
                    ][1], LINESUPPRESS_NONE );
    }
}
```

Listing 112 – CMG_DRAW_LineList.c

Der erste Parameter ist ein beliebig großes Array aus Punkten, also Koordinatenpaaren. Der zweite Parameter gibt die Anzahl der Punkte im Array an und der letzte, ob die Linie geschlossen werden soll.

Wenn keine Punkte vorhanden sind kehrt die Funktion sofort zurück – bei nur einem wird der Pixel gesetzt. Bei zwei oder mehr Punkten wird die Linienliste gezeichnet; zuerst alle Liniensegmente – wobei bei jeder Linie der letzte Pixel unterdrückt wird. Nur beim letzten Liniensegment wird unterschieden: Soll die Linie geschlossen werden, so wird auch hier der letzte Punkt unterdrückt und auch die Linie vom letzten zum ersten Punkt gezeichnet. Ist die Linie nicht geschlossen, wird das letzte Segment komplett gezeichnet.

Die Anzahl der zu zeichnenden Linien:

Modus	Linienanzahl
Offene Linienliste	Anzahl der Punkte – 1
Geschlossene Linienliste	Anzahl der Punkte

Tabelle 36 – DRAW, LineList, Anzahl der Linen

Beispiel:
Eine kleine Liste mit Koordinaten wird definiert und mit unterschiedlichen Konfigurationen gezeichnet:

```
    cmg_Coord   aaPoints[][2] = { { 10, 30 }, { 20, 0 }, { 30, 10 }, { 40, 0 }, { 50, 30 } };

    CMG_LineList( aaPoints, 5, false );
```

Listing 113 – DRAW, LineList, Beispiel 1

Abbildung 59– DRAW, LineList, Beispiel 1

Soll die Linienliste geschlossen werden, ist folgender Aufruf zu verwenden:

```
    CMG_LineList( aaPoints, 5, true );
```

Listing 114 – DRAW, LineList, Beispiel 2

Abbildung 60– DRAW, LineList, Beispiel 2

9.7.3. LINIE MIT BREITE

Eine entscheidende Eigenschaft haben alle bisherigen Linienfunktionen: Sie haben eine feste Stärke bzw. Breite von genau einem Pixel. Mit `LineWidth` lassen sich jedoch beliebig breite Linien zeichnen.

Diese Funktion verwendet einen sehr einfachen Algorithmus und eignet sich gut für nur wenige Pixel dicke Linien (etwa 2-5 Pixel). Es lassen sich jedoch auch beliebig breite Linien zeichnen.

Der Algorithmus funktioniert wie folgt: Die Funktion berechnet für die X- und Y-Richtung die absolute Länge der Linie. Diese Linie wird nun um die Achse mit der kleineren Länge parallel verschoben und jeweils gezeichnet.

Zuerst jedoch der Quellcode:

```
/*****************************************************************************
| CMG_Line
|-----------------------------------
| draw a line
+----------------------------------------------------------------------------
| @Params:
|    * iX1:          x1 coord
|    * iY1:          y1 coord
|    * iX2:          x2 coord
|    * iY2:          y2 coord
|    * iWidth:       width of the line in pixels
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_LineWidth( cmg_Coord iX1, cmg_Coord iY1, cmg_Coord iX2, cmg_Coord iY2, cmg_Coord iWidth )
{
    cmg_Coord   iAbsDeltaX, iAbsDeltaY, iDeltaStart;

    // param check
    if ( iWidth < 1 )
        return;

    // calc deltas
    iAbsDeltaX = iX2 - iX1;
    if ( iAbsDeltaX < 0 )   iAbsDeltaX = -iAbsDeltaX;
    iAbsDeltaY = iY2 - iY1;
    if ( iAbsDeltaY < 0 )   iAbsDeltaY = -iAbsDeltaY;

    // calc delta
    iDeltaStart = ( iWidth - 1 ) >> 1;

    // get width direction
    if ( iAbsDeltaX >= iAbsDeltaY )
    {
        // add width in y direction

        // adjust y
        iY1 -= iDeltaStart;
        iY2 -= iDeltaStart;

        // draw lines
        while ( iWidth-- )
            CMG_Line( iX1, iY1++, iX2, iY2++ );
    }
    else
```

```
    {
        // add width in x direction
        // adjust x
        iX1 -= iDeltaStart;
        iX2 -= iDeltaStart;

        // draw lines
        while ( iWidth-- )
            CMG_Line( iX1++, iY1, iX2++, iY2 );
    }
}
```

Listing 115 – CMG_DRAW_LineWidth.c

Dieser Algorithmus ist recht schnell, er ließe sich aber noch dahingehend optimierten, daß nicht für jede Linienbreite eine neue Line gezeichnet werden muß, sondern es insgesamt nur einen Durchgang gibt. Nachdem die Funktion aber eher selten verwendet wird, habe ich mich zugunsten der Codegröße für diese Variante entschieden.

Ein weiterer Nachteil ist, daß die Linienbreite nicht – wie es sein sollte – im rechten Winkel zur Linienrichtung verschoben wird, sondern nur zur passenden Achse. Dieses Phänomen zeigt später auch das letzte Beispiel noch einmal deutlich.

Zur Abhilfe könnte man eine beliebig breite, korrekt gezeichnete Linie mit Hilfe von Polygonen zeichnen, also aus zwei Dreiecken zusammensetzen. Allerdings können dann keine Stifte mehr verwendet werden, sondern nur Pinsel, die sich jedoch (noch) nicht rotieren lassen. Eine recht komplexe und verzwickte Angelegenheit also. Für einfache Anwendungen reicht diese Funktion jedoch durchaus aus.

Beispiel:
Verschieden breite Linien mit einer konstanten Farbe:

```
    CMG_SetColor( 1 );

    CMG_LineWidth( 5, 5, 50, 5, 1 );
    CMG_LineWidth( 5, 15, 50, 15, 2 );
    CMG_LineWidth( 5, 25, 50, 25, 3 );
    CMG_LineWidth( 5, 35, 50, 35, 4 );
    CMG_LineWidth( 5, 45, 50, 45, 5 );

    CMG_LineWidth( 60, 5, 110, 5, 2 );
    CMG_LineWidth( 60, 10, 110, 35, 2 );
    CMG_LineWidth( 60, 20, 70, 50, 2 );

    CMG_LineWidth( 120, 5, 170, 5, 3 );
    CMG_LineWidth( 120, 10, 170, 35, 3 );
    CMG_LineWidth( 120, 20, 130, 50, 3 );

    CMG_LineWidth( 180, 5, 230, 5, 4 );
    CMG_LineWidth( 180, 10, 230, 35, 4 );
    CMG_LineWidth( 180, 20, 190, 50, 4 );
```

Listing 116 – DRAW, LineWidth, Beispiel 1

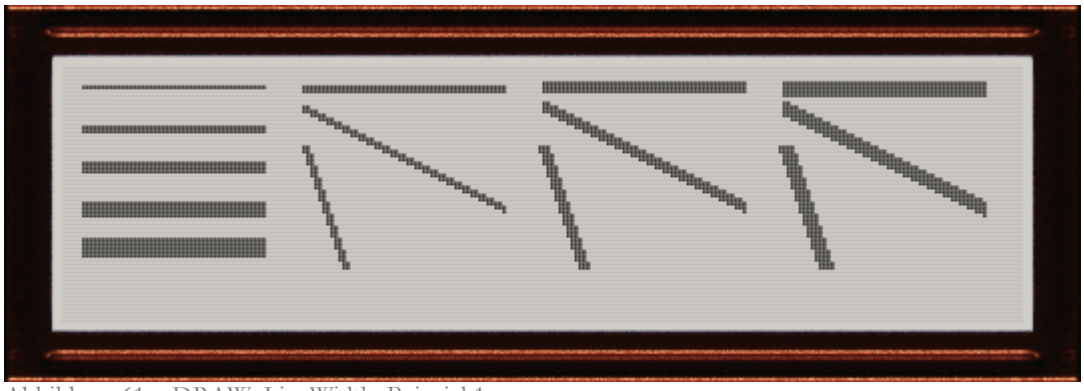


Abbildung 61 – DRAW, LineWidth, Beispiel 1

Wird anstelle der konstanten Farbe ein Stift mit `CMG_SetPen( &PenDashDotDot );` verwendet:

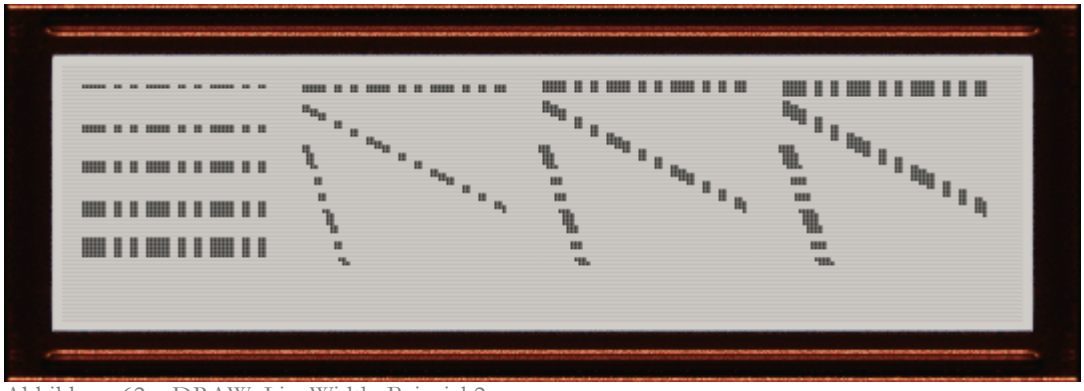


Abbildung 62 – DRAW, LineWidth, Beispiel 2

Bei Linien mit Breite, welche mit Stiften gezeichnet werden, sehen bis etwa Breite drei vernünftig aus. Ab einer Breite von vier läßt sich die oben gezeigte Schwachstelle schon deutlich wahrnehmen.

Besonders extrem wird es bei sehr breiten Linien, die einen großen Winkelunterschied zu den Achsen besitzen, also nahezu diagonal sind. Noch dazu ändert sich die Richtung schlagartig beim Überschreiten der 45°-Marke von der einen Achse zu der anderen:

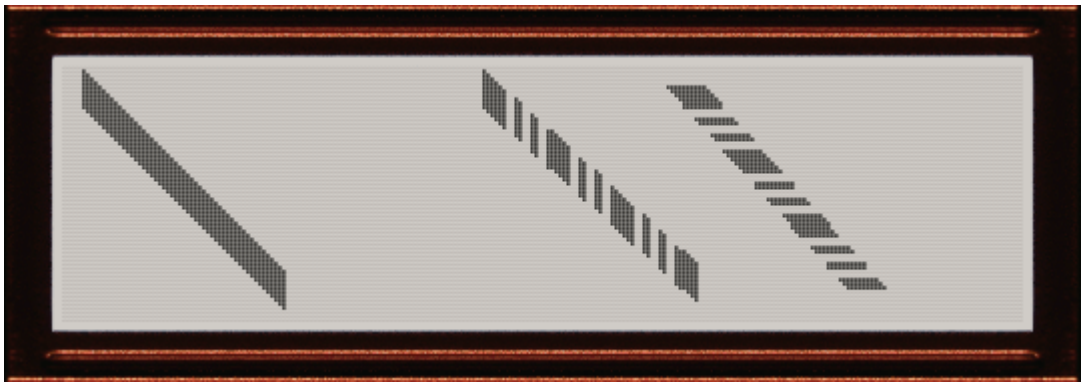


Abbildung 63 – DRAW, LineWidth, Beispiel 3

Diese Fälle sollten bei der Verwendung dieser Funktion möglichst vermieden werden.

9.8. Rechtecke

Rechtecke sind wohl – neben den Linien – eine der meistgenutzten Grafikobjekte. CMG bietet zwei verschiedene Typen an: Normale Rechtecke und abgerundete Rechtecke.

Bisher wurden nur nicht füllbare Grafikobjekte behandelt - alle folgenden in diesem Kapitel sind nun aber auch füllbar. Das bedeutet, sie bestehen aus einem Rand und einem geschlossenen Inneren. Der Rand wird mit einem Stift gezeichnet, das Innere mit einem Pinsel.

Es gibt nun drei verschiedene Zeichenmodi:

Zeichenmodi (Drawstyle)	Beschreibung
`BORDER`	Nur der Rand wird mit dem aktuellen Stift gezeichnet.
`FILL`	Nur das Innere – inklusive der Pixel des Randes – wird mit dem aktuellen Pinsel gezeichnet.
`BOTH`	Sowohl der Rand als auch das Innere wird gezeichnet.

Tabelle 37 – Zeichenmodi

9.8.1. Normale Rechtecke

Das Zeichnen eines Rechtecks ist recht einfach.

Für den Rand existieren nur jeweils zwei horizontale und vertikale Linien. Im Gegensatz zu nicht achsenparallelen Linien, die aus einzelnen Pixeln gezeichnet werden, bietet die DRV-Schicht hierfür die optimierten Funktionen `HLine` und `VLine` an.

Den Füllbereich zu zeichnen ist genauso einfach: Für jede Pixelzeile im Rechteck wird eine horizontale Linie von der linken bis zur rechten Grenze gezeichnet.

Hier der vollständige Code der Rechteckfunktion:

```
_PUBLIC
void CMG_Rectangle( cmg_Coord iX1, cmg_Coord iY1, cmg_Coord iX2, cmg_Coord iY2 )
{
    // display orientation
    _DRAW_AdjustXY( &iX1, &iY1 );
    _DRAW_AdjustXY( &iX2, &iY2 );

    // adjust coords
    {
        cmg_Coord   iTemp;
        if ( iX1 > iX2 )
            SWAP( iX1, iX2, iTemp );
        if ( iY1 > iY2 )
            SWAP( iY1, iY2, iTemp );
    }

    // draw border?
    if ( g_DRAW_uDrawStyle & DRAWSTYLE_BORDER )
    {
        // turn off adjustment because we have done this already
        g_DRAW_bProhibitCoordAdjustment = true;
        CMG_LineEx( iX1, iY1, iX2, iY1, LINESUPPRESS_LAST );
        CMG_LineEx( iX2, iY1, iX2, iY2, LINESUPPRESS_LAST );
        CMG_LineEx( iX2, iY2, iX1, iY2, LINESUPPRESS_LAST );
        CMG_LineEx( iX1, iY2, iX1, iY1, LINESUPPRESS_LAST );
        g_DRAW_bProhibitCoordAdjustment = false;
    }

    // fill rect?
    if ( g_DRAW_uDrawStyle & DRAWSTYLE_FILL )
    {
        cmg_Coord   iSizeX, iSizeY, iSpoolX, iSpoolY;

        // init spool
        iSpoolX = iSpoolY = 0;

        // border was drawn?
        if ( g_DRAW_uDrawStyle & DRAWSTYLE_BORDER )
        {
            // since we drawed a border we must adjust the inner fill area
            iX1++;
            iX2--;
            iY1++;
            iY2--;
        }
```

```
        // clipping
        if ( g_DRAW_iClippingWindow_LX > iX1 )
        {
            // update spool (since we must clip)
            iSpoolX = g_DRAW_iClippingWindow_LX - iX1;
            // set new x
            iX1 = g_DRAW_iClippingWindow_LX;
        }
        if ( g_DRAW_iClippingWindow_TY > iY1 )
        {
            // update spool (since we must clip)
            iSpoolY = g_DRAW_iClippingWindow_TY - iY1;
            // set new x
            iY1 = g_DRAW_iClippingWindow_TY;
        }
        // clip bottom right - no spool update here
        iX2 = MIN( iX2, g_DRAW_iClippingWindow_RX );
        iY2 = MIN( iY2, g_DRAW_iClippingWindow_BY );

        // nothing to draw anymore?
        if ( ( ( iX2 - iX1 ) < 0 ) || ( ( iY2 - iY1 ) < 0 ) )
            return;

        // calc size
        iSizeX = iX2 - iX1 + 1;
        iSizeY = iY2 - iY1 + 1;

        // init brush
        _DRAW_ApplyBrush( iSpoolY );

        // for each line
        while ( iSizeY-- )
        {
            // set brush line
            _DRAW_ApplyBrushNextLine( iSpoolX );
            // draw line
            CMG_DRV_HLine( iX1, iY1, iSizeX );
            // next line
            iY1++;
        }
    }
}
```

Listing 117 – CMG_DRAW_Rectangle.c

Wie immer werden zuerst die übergebenen Koordinaten in physikalische umgewandelt.

Anschließend prüft die Funktion die Reihenfolge der Koordinaten und korrigiert diese entsprechend. Es ist notwendig, daß `X1` kleiner ist als `X2`, `X1` also dem linken Punkt entspricht. Die gleiche Aussage gilt ebenfalls entsprechend für `Y`.

Danach muß, wenn ausgewählt, der Rand gezeichnet werden. Dazu verwendet der Algorithmus die `LineEx`-Funktion aus der gleichen Schicht, die dann zwischen den passenden, optimierten Linienarten die richtigen auswählt. Das Rechteck wird am Stück gezeichnet, jeweils mit Unterdrückung des letzten Pixels jeder Linie. Um das Clipping kümmert sich die `LineEx`-Funktion, so daß wir hier keine weiteren Arbeiten zu erledigen haben.

Soll der innere Bereich gefüllt werden, so können wir auf keine vorhandene Funktion zugreifen, sondern müssen die Funktionalität selbst implementieren.

Falls der Rand gezeichnet wurde paßt die Funktion zuerst die Koordinaten an. Dabei wird jede Seite um einen Pixel zur Mitte hin verschoben, damit später der Rand nicht wieder überschrieben wird.

Danach paßt die Funktion das Rechteck an die Clipping-Grenzen an und aktualisiert entsprechend die Spooling-Werte. Falls nicht mehr gezeichnet werden muß, kehrt die Funktion zurück.

Zuletzt berechnet der Algorithmus die endgültige Breite und Höhe und zeichnet dann die innere Fläche mit dem aktuellen Pinsel in einer Schleife.

Beispiel:
Einige Rechtecke mit verschiedenen Zeicheneinstellungen: Links mit konstanten Farben und rechts mit Stift und Pinsel:

```
    CMG_SetColor( 1 );
    CMG_SetFillColor( 1 );

    CMG_SetDrawStyle( DRAWSTYLE_BORDER );
    CMG_Rectangle( 5, 5, 30, 30 );
    CMG_SetDrawStyle( DRAWSTYLE_FILL );
    CMG_Rectangle( 35, 5, 60, 30 );
    CMG_SetDrawStyle( DRAWSTYLE_BOTH );
    CMG_Rectangle( 65, 5, 90, 30 );

    CMG_SetPen( &PenDashDotDot );
    CMG_SetBrush( &BrushLight );
    CMG_SetDrawStyle( DRAWSTYLE_BORDER );
    CMG_Rectangle( 125, 5, 150, 30 );
    CMG_SetDrawStyle( DRAWSTYLE_FILL );
    CMG_Rectangle( 155, 5, 180, 30 );
    CMG_SetDrawStyle( DRAWSTYLE_BOTH );
    CMG_Rectangle( 185, 5, 210, 30 );
```

Listing 118 – DRAW, Rectangle, Beispiel 1

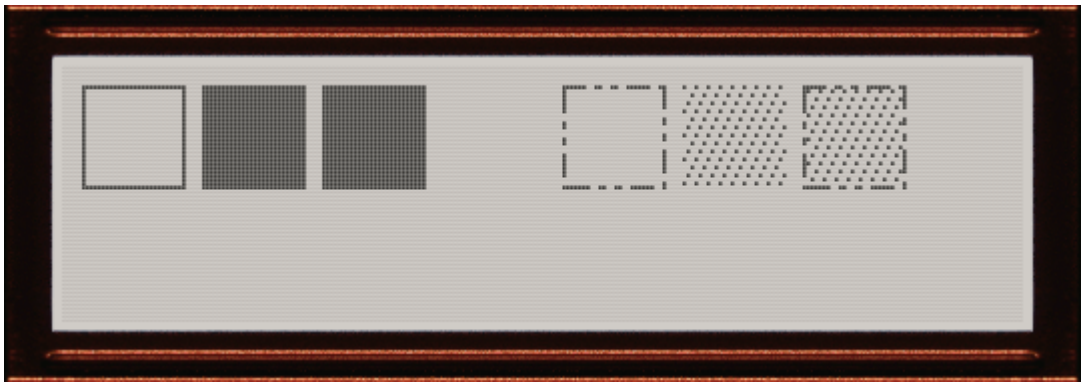


Abbildung 64 – DRAW, Rectangle, Beispiel 1

Hier noch mit dem XOR-ROP-Modus:

```
    CMG_SetColor( 1 );
    CMG_SetFillColor( 1 );
    CMG_SetROPMode( ROPMODE_XOR );

    CMG_SetDrawStyle( DRAWSTYLE_FILL );
    CMG_Rectangle( 10, 5, 50, 30 );
    CMG_Rectangle( 40, 20, 70, 50 );

    CMG_SetDrawStyle( DRAWSTYLE_FILL );
    CMG_Rectangle( 140, 20, 220, 50 );
    CMG_SetDrawStyle( DRAWSTYLE_BORDER );
    CMG_Rectangle( 120, 30, 150, 60 );
    CMG_Rectangle( 160, 25, 170, 35 );
    CMG_SetBrush( &BrushHelloWorld );
    CMG_SetDrawStyle( DRAWSTYLE_BOTH );
    CMG_Rectangle( 190, 5, 235, 35 );
```

Listing 119 – DRAW, Rectangle, Beispiel 2

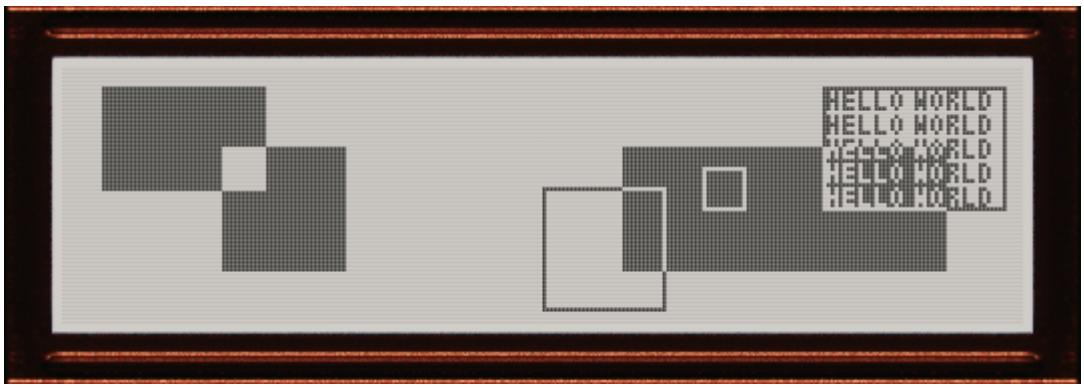


Abbildung 65 – DRAW, Rectangle, Beispiel 2

Als letztes Beispiel der Unterschied zwischen dem kompletten und dem transparenten Zeichnen mit Hilfe des ROP-Modes:

```
    CMG_SetDrawStyle( DRAWSTYLE_BOTH );

    CMG_SetROPMode( ROPMODE_COPY );
    CMG_SetBrush( &BrushHelloWorld );
    CMG_Rectangle( 0, 5, 65, 40 );
    CMG_SetBrush( &BrushLight );
    CMG_Rectangle( 40, 20, 90, 50 );
    CMG_SetBrush( &Brush50 );
    CMG_Rectangle( 70, 30, 110, 60 );

    CMG_SetROPMode( ROPMODE_OR );
    CMG_SetBrush( &BrushHelloWorld );
    CMG_Rectangle( 120, 5, 185, 40 );
    CMG_SetBrush( &BrushLight );
    CMG_Rectangle( 160, 20, 210, 50 );
    CMG_SetBrush( &Brush50 );
    CMG_Rectangle( 190, 30, 230, 60 );
```

Listing 120 – DRAW, Rectangle, Beispiel 3

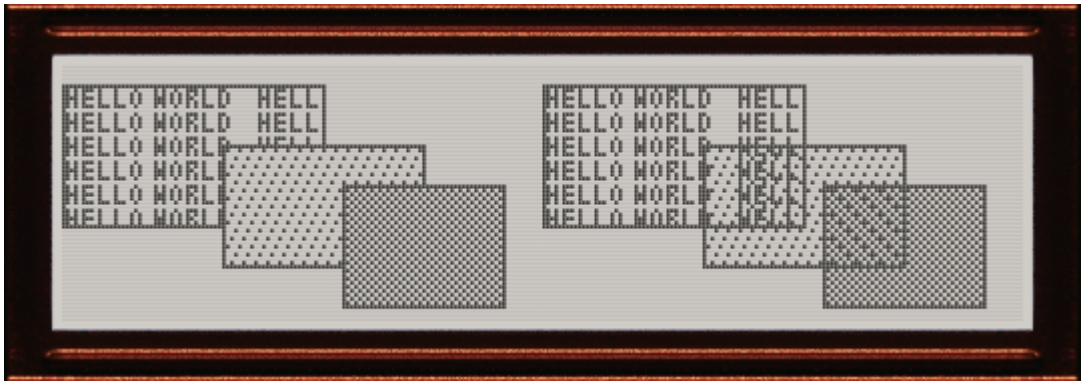


Abbildung 66 – DRAW, Rectangle, Beispiel 3

9.8.2. Abgerundete Rechtecke

Designmäßig geht der Trend schon seit Jahren weg vom Eckigen hin zum Abgerundeten. In der Anfangszeit waren runde Objekte aus Performancegründen oft nicht realisierbar, zumindest war das oft die Erklärung. Da CMG jedoch ohnehin Ellipsen und damit auch Kreise zeichnet, sind abgerundete Rechtecke nur eine logische Erweiterung. Und mal ganz ehrlich: Runde Knöpfe sind doch wirklich ansprechender als langweilig eckige.

Aus Sicht der Performance sind abgerundete Rechtecke sicherlich nicht mit normalen Rechtecken vergleichbar. Allerdings hält sich der Aufwand in Grenzen und entspricht etwa dem Zeichnen des Rechtecks und zusätzlich einem Kreis.

Der Code erscheint auf den ersten Blick recht umfangreich und wird im folgenden grob erklärt. Dafür wird die Funktion in drei einzelne Teile zerlegt:

```
void CMG_RoundedRectangle( cmg_Coord iX1, cmg_Coord iY1, cmg_Coord iX2, cmg_Coord iY2, cmg_Coord iRadius )
_PUBLIC
{
    cmg_DRAW_Ellipse_Data       EllData;

    // display orientation
    _DRAW_AdjustXY( &iX1, &iY1 );
    _DRAW_AdjustXY( &iX2, &iY2 );

    // adjust coords
    {
        cmg_Coord   iTemp;
        if ( iX1 > iX2 )
            SWAP( iX1, iX2, iTemp );
        if ( iY1 > iY2 )
            SWAP( iY1, iY2, iTemp );
    }
```

```
    // . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
    // param check
    // invalid radius?
    if ( iRadius < 0 )
        return;

    // simple pixel to draw
    if ( iRadius == 0 )
    {
        CMG_Rectangle( iX1, iY1, iX2, iY2 );
        return;
    }

    iRadius = MIN( iRadius, ( iX2 - iX1 ) >> 1 );
    iRadius = MIN( iRadius, ( iY2 - iY1 ) >> 1 );

    // . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
    // one time data structure init
    _Ellipse_OneTimeInit( &EllData, iRadius, iRadius );

[...]
```

Listing 121 – CMG_DRAW_RoundedRectangle.c, Teil 1

Zuerst wieder die obligatorische Umwandlung, danach das vom normalen Rechteck schon bekannte Sortieren der Koordinaten.

Anschließend wird der Radius-Parameter überprüft und der einfachste Sonderfall mit einem Radius von Null – also einem normalen Rechteck ohne Abrundung – behandelt.

Zuletzt muß noch der Radius an die Größe des Rechtecks angepaßt werden. Der Radius darf nicht größer als die Hälfte der Seitenlänge des Rechtecks werden.

Danach wird – diesmal zuerst – der innere Bereich gezeichnet:

```
[...]
    // . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
    // fill inside
    if ( g_DRAW_uDrawStyle & DRAWSTYLE_FILL )
    {
        cmg_Coord       iSpoolLeftmostX;
        cmg_Coord       iXInc;
        // last loop values
        cmg_Coord       iXLastLeft, iXLastRight, iYLast;

        // set brush/solid
        _DRAW_ApplyBrush( 0 );

        // init spool start value
        iSpoolLeftmostX = iX1;
        // when to draw a border we have to shift the brush one inside
        if ( g_DRAW_uDrawStyle & DRAWSTYLE_BORDER )
            iSpoolLeftmostX++;

        // calc vars
        iXInc = iX2 - iX1 - ( iRadius << 1 );


        // . . . . . . . . . . . . . . . . . . .
        // fill upper rounded part
        // init ellipse loop
        _Ellipse_LoopInside_Init( &EllData, (cmg_Coord)( iX1 + iRadius ), (cmg_Coord)( iY1 + iRadius ),
                                  iRadius, iRadius );
        EllData.iX_Right += iXInc;
        // init last loop values
        iXLastLeft  = EllData.iX_Left;
        iXLastRight = EllData.iX_Right;
        iYLast      = EllData.iY_Top;

        // if we have to draw a borter too we only fill the inside without border but use a different
                              technique
        // requiring one line less
        if ( g_DRAW_uDrawStyle == DRAWSTYLE_BOTH )
            EllData.iCount--;

        // for all pixels
        while ( EllData.iCount > 0 )
        {
            // update positions
            _Ellipse_LoopInside( &EllData );
```

```
            // if line has changed
            if ( iYLast != EllData.iY_Top )
            {
                // if we also draw the border only fill the inside, else the complete ellipse
                if ( g_DRAW_uDrawStyle & DRAWSTYLE_BORDER )
                    _DRAW_ClippedHLine( (cmg_Coord)( EllData.iX_Left + 1 ), (cmg_Coord)( EllData.iX_Right -
                            1 ), EllData.iY_Top, iSpoolLeftmostX );
                else
                    _DRAW_ClippedHLine( iXLastLeft, iXLastRight, iYLast, iSpoolLeftmostX );

                // store old line
                iYLast      = EllData.iY_Top;
            }

            // store old x coords
            iXLastLeft  = EllData.iX_Left;
            iXLastRight = EllData.iX_Right;
        }


        // . . . . . .
        // fill straight part
        if ( g_DRAW_uDrawStyle & DRAWSTYLE_BORDER )
            iYLast++;

        // use iYLast for temp y counter
        iXLastLeft = iY2 - iY1 - ( iRadius << 1 );
        while ( iXLastLeft-- )
        {
            if ( g_DRAW_uDrawStyle & DRAWSTYLE_BORDER )
                _DRAW_ClippedHLine( (cmg_Coord)( iX1 + 1 ), (cmg_Coord)( iX2 - 1 ), iYLast++,
                        iSpoolLeftmostX );
            else
                _DRAW_ClippedHLine( iX1, iX2, iYLast++, iSpoolLeftmostX );
        }


        // . . . . . .
        // fill lower rounded part
        // init ellipse loop
        _Ellipse_LoopOutside_Init( &EllData, (cmg_Coord)( iX1 + iRadius ), (cmg_Coord)( iY2 - iRadius ),
                        iRadius, iRadius );
        EllData.iX_Right += iXInc;
        // init last loop values
        iXLastLeft  = EllData.iX_Left;
        iXLastRight = EllData.iX_Right;
        iYLast      = EllData.iY_Top;

        // if we _DON'T_ have to draw a borter too we do have to draw an extra line due the useage of a
                        different technique
        if ( g_DRAW_uDrawStyle != DRAWSTYLE_BOTH )
            _DRAW_ClippedHLine( EllData.iX_Left, iXLastRight, EllData.iY_Bottom, iSpoolLeftmostX );


        // for all pixels
        while ( EllData.iCount > 0 )
        {
            // update positions
            _Ellipse_LoopOutside( &EllData );

            // if line has changed
            if ( iYLast != EllData.iY_Bottom )
            {
                // if we also draw the border only fill the inside, else the complete ellipse
                if ( g_DRAW_uDrawStyle & DRAWSTYLE_BORDER )
                    _DRAW_ClippedHLine( (cmg_Coord)( iXLastLeft + 1 ), (cmg_Coord)( iXLastRight - 1 ),
                            iYLast, iSpoolLeftmostX );
                else
                    _DRAW_ClippedHLine( EllData.iX_Left, EllData.iX_Right, EllData.iY_Bottom,
                            iSpoolLeftmostX );

                // store old line
                iYLast      = EllData.iY_Bottom;
            }

            // store old x coords
            iXLastLeft  = EllData.iX_Left;
            iXLastRight = EllData.iX_Right;
        }
    }
```

Listing 122 – CMG_DRAW_RoundedRectangle.c, Teil 2

Dieser Bereich teilt sich wiederum in drei Unterblöcke auf, die zeilenweise das komplette Rechteck zeichnen:

- **1. Block – Oberer Bereich:**
  Der Bereich von der obersten Linie bis zum Ende des abgerundeten Randes.

  Die erste Zeile ist die kürzeste. Die Länge ist die Gesamtbreite, abzüglich jeweils des vollen Radius' an beiden Seiten. Zeilenweise wird die Länge der Linie, bis hin zur eigentlichen Breite des Rechtecks, mit der Ellipsenfunktion verlängert.

- **2. Block – Mittlerer Bereich:**
  Der Bereich zwischen den abgerundeten Teilen darüber und darunter.

  Dieser Bereich ist besonders einfach und beinhaltet nur Linien der vollen Breite.

- **3. Block – Unterer Bereich:**
  Der Bereich unterhalb bis zum Ende des Rechtecks.

  Hier wiederholt sich – in umgekehrter Reihenfolge – das Vorgehen aus dem ersten Block, bis hin zum Ende des Rechtecks.

Zuletzt zeichnet die Funktion bei Bedarf noch den Rand:

```
    // . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
    // draw border
    if ( g_DRAW_uDrawStyle & DRAWSTYLE_BORDER )
    {
        // set pen/solid
        _DRAW_ApplyPen( 0 );

        // top tile
        CMG_DRV_HLine( (cmg_Coord)( iX1 + iRadius ), iY1, (cmg_Coord)( iX2 - iX1 - ( iRadius << 1 ) ) );

        // top right rounded edge
        _Ellipse_LoopInside_Init( &EllData, (cmg_Coord)( iX2 - iRadius ), (cmg_Coord)( iY1 + iRadius ),
                        iRadius, iRadius );
        while ( EllData.iCount > 0 )
        {
            _DRAW_ClippedPixel( EllData.iX_Right, EllData.iY_Top );
            _Ellipse_LoopInside( &EllData );
        }

        // right tile
        CMG_DRV_VLine( iX2, (cmg_Coord)( iY1 + iRadius ), (cmg_Coord)( iY2 - iY1 - ( iRadius << 1 ) ) );

        // bottom right rounded edge
        _Ellipse_LoopOutside_Init( &EllData, (cmg_Coord)( iX2 - iRadius ), (cmg_Coord)( iY2 - iRadius ),
                        iRadius, iRadius );
        while ( EllData.iCount > 0 )
        {
            _DRAW_ClippedPixel( EllData.iX_Right, EllData.iY_Bottom );
            _Ellipse_LoopOutside( &EllData );
        }

        // bottom tile
        CMG_DRV_HLine( (cmg_Coord)( iX1 + iRadius + 1 ), iY2, (cmg_Coord)( iX2 - iX1 - ( iRadius << 1 ) )
                        );

        // bottom left rounded edge
        _Ellipse_LoopInside_Init( &EllData, (cmg_Coord)( iX1 + iRadius ), (cmg_Coord)( iY2 - iRadius ),
                        iRadius, iRadius );
        while ( EllData.iCount > 0 )
        {
            _DRAW_ClippedPixel( EllData.iX_Left, EllData.iY_Bottom );
            _Ellipse_LoopInside( &EllData );
        }

        // left tile
        CMG_DRV_VLine( iX1, (cmg_Coord)( iY1 + iRadius + 1 ), (cmg_Coord)( iY2 - iY1 - ( iRadius << 1 ) )
                        );

        // top left rounded edge
        _Ellipse_LoopOutside_Init( &EllData, (cmg_Coord)( iX1 + iRadius ), (cmg_Coord)( iY1 + iRadius ),
                        iRadius, iRadius );
        while ( EllData.iCount > 0 )
        {
            _DRAW_ClippedPixel( EllData.iX_Left, EllData.iY_Top );
            _Ellipse_LoopOutside( &EllData );
        }
    }
}
```

Listing 123 – CMG_DRAW_RoundedRectangle.c, Teil 3

Dieser Teil besteht aus acht Bereichen und zeichnet das Rechteck am Stück ohne Unterbrechung. Der Startpunkt ist links oben mit der geraden Linie bis zum rechten Anfang der rechten, oberen Abrundung. Anschließend wird diese mit einem Viertelkreis gezeichnet; danach die rechte Linie nach unten bis zum Anfang der nächsten Abrundung. Dies setzt sich solange fort, bis zuletzt die Rundung links oben gezeichnet wurde.

Hier könnten zusätzlich noch einige Optimierungen stattfinden, da die Zeichenoperationen punktsymmetrisch bzw. achsensymmetrisch sind. Das innere Füllen könnte beim Zeichnen des oberen Bereichs gleichzeitig den unteren mitzeichnen und beim Zeichnen des Randes müßten nicht insgesamt vier Viertelkreise nacheinander berechnet werden, sondern nur einer und damit könnten alle vier Abrundungen gleichzeitig gezeichnet werden. Diese Optimierungen funktionieren allerdings nur mit konstanten Farben, da bei Stiften und Pinseln die Zeichenreihenfolge wichtig ist. Deshalb habe ich mich zugunsten der Codegröße auf keine zusätzliche Optimierungen festgelegt.

Beispiel:
Ein paar abgerundete Rechtecke mit verschiedenen Zeicheneinstellungen:

```
    CMG_SetDrawStyle( DRAWSTYLE_BORDER );
    CMG_RoundedRectangle( 5, 5, 65, 15, 1 );
    CMG_RoundedRectangle( 5, 20, 65, 30, 2 );
    CMG_RoundedRectangle( 5, 35, 65, 45, 4 );
    CMG_RoundedRectangle( 5, 50, 65, 60, 5 );

    CMG_SetDrawStyle( DRAWSTYLE_BOTH );
    CMG_SetBrush( &BrushLight );
    CMG_RoundedRectangle( 120, 25, 185, 50, 6 );
    CMG_SetBrush( &BrushHelloWorld );
    CMG_RoundedRectangle( 100, 5, 140, 35, 10 );
    CMG_SetBrush( &Brush50 );
    CMG_RoundedRectangle( 165, 40, 200, 60, 8 );
```

Listing 124 – DRAW, RoundedRectangle, Beispiel

Abbildung 67 – DRAW, RoundedRectangle, Beispiel

9.9. Kreise und Ellipsen

Der Algorithmus für das Zeichnen von Kreisen wurde schon von abgerundeten Rechtecken für das Berechnen der Randpunkte benutzt. Er selbst baut auf Bresenhams Ellipsenalgorithmus [Bres1, Bres2] auf und wurde noch zusätzlich an die Gegebenheiten von CMG optimiert.

Zuerst einmal stellt sich die Frage, wie man Kreise bzw. Ellipsen am schnellsten zeichnet. Da Ellipsen punktsymmetrisch zum Mittelpunkt sind, würde die Berechnung eines Viertels der Ellipse ausreichen, bei Kreisen sogar nur das eines Achtels. Die restlichen Punkte könnten nun einfach durch Spiegelung an den Achsen des Kreises bzw. an den der Ellipse berechnet werden.

Zugunsten der Codegröße verwendet CMG keinen eigenen Algorithmus für Kreise. Einerseits würde dieser nur minimal schneller sein als der optimierte für Ellipsen, und andererseits würde das bei Stiften und Pinseln nichts bringen, ganz im Gegenteil. Aber dazu später mehr. Um Kreise zu zeichnen, verwendet CMG deshalb die Ellipsenfunktion und übergibt dieser für beide Radienangaben den gleichen Radius des zu zeichnenden Kreises.

Doch zurück zum Problem: Das Ellipsen-Zeichnen. Der Algorithmus gibt folgende Punkte zurück:

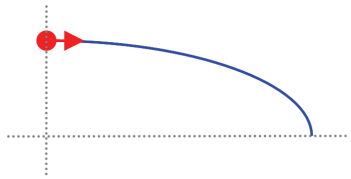


Abbildung 68 – Ellipse, Teil 1

Der Startpunkt ist oben mittig. Nach jedem Aufruf gibt die Ellipsenfunktion den nächsten Punkt im Uhrzeigersinn zurück. Der letzte Punkt ist rechts mittig.

Um nun eine halbe Ellipse zu erhalten, wird nicht nur oben mittig, sondern gleichzeitig auch unten mittig gestartet. Die X-Werte bleiben gleich und die Y-Werte werden invertiert:

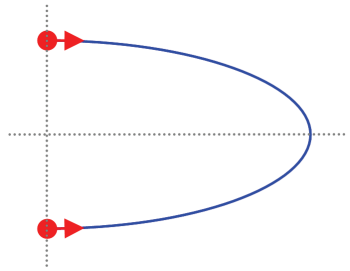


Abbildung 69 – Ellipse, Teil 2

Werden zusätzlich noch die Punkte an der Y-Achse der Ellipse gespiegelt, also alle X-Werte invertiert, so ergibt sich eine komplette Ellipse:

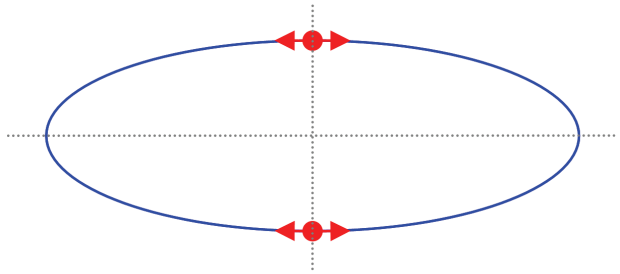


Abbildung 70 – Ellipse, Teil 3

Soll eine Ellipse mit einer konstanten Farbe gezeichnet werden, so werden die Punkte so berechnet. Wird jedoch ein Stift verwendet, so ergibt sich ein Problem: Im zweiten und vierten Quadranten ist die Laufrichtung verkehrt, der Stift würde also verkehrt herum gezeichnet werden. CMG bietet deswegen nicht nur eine Ellipsenfunktion an, sondern zwei. Die zweite benutzt genau denselben Algorithmus, vertauscht allerdings die Operationen in der passenden Weise und übernimmt damit bei Bedarf den zweiten Quadranten:

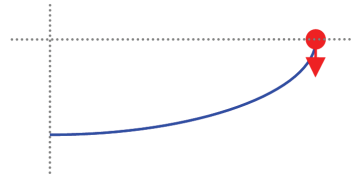


Abbildung 71 – Ellipse, Teil 4

Durch Spiegelung kann nun auch der letzte noch ausstehende Quadrant in der richtigen Richtung gezeichnet werden:

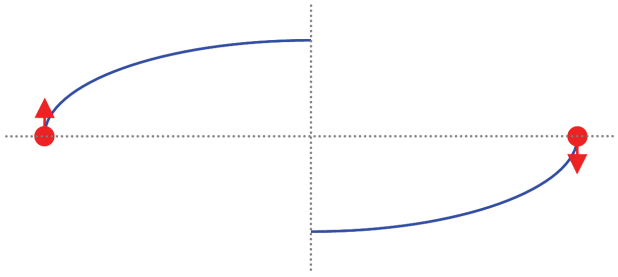


Abbildung 72 – Ellipse, Teil 5

Setzt man alle Teile zusammen, so kann mit vier Durchläufen eine komplette Ellipse mit einem Stift gezeichnet werden:

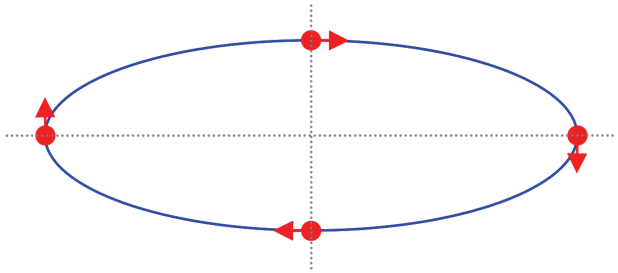


Abbildung 73 – Ellipse, Teil 6

Die komplette Quelldatei für das Ellipsenzeichnen und den Algorithmus ist die größte in der DRAW-Schicht. Die genaue Erklärung würde den Rahmen der Arbeit sprengen. Deshalb sollen im folgenden nur markante Punkte behandelt werden. Für die genaue Funktionsweise ziehen Sie bitte den Quellcode, mit seinen Kommentaren, zu Rate.

Die Ellipsenfunktion beginnt mit dem Funktionskopf:

```
/*****************************************************************************
| CMG_Ellipse
|-----------------------------------
| ellipse algorithm (bresenham)
+----------------------------------------------------------------------------
| @Params:
|    * iMidX:         x midpoint
|    * iMidY:         y midpoint
|    * iRadiusX:      x radius
|    * iRadiusY:      y radius
| @Return Value:     none
+*****************************************************************************/
_PUBLIC
void CMG_Ellipse( cmg_Coord iMidX, cmg_Coord iMidY, cmg_Coord iRadiusX, cmg_Coord iRadiusY )
{
[...]
```

Listing 125 – CMG_DRAW_Ellipse.c, Teil 1

Zuerst werden, nach der Konvertierung der Koordinaten, die Sonderfälle behandelt: Ist ein Radius kleiner als Null, so wird nichts gezeichnet. Sind beide genau Null, so ergibt sich nur ein einzelner Pixel. Ist nur einer von beiden Null, so zeichnet die Funktion die passende Linie.

Anschließend wird – bei Bedarf – das Innere mit dem aktuellen Pinsel gefüllt. Für die obere Hälfte verwendet die Funktion den ersten Ellipsenalgorithmus, für die untere Hälfte den zweiten. Eine weitere Schwierigkeit ist, daß nur der innere Bereich ohne Rand gezeichnet werden darf, falls der Rand anschließend noch extra gezeichnet werden soll.

Das oben gerade beschriebene Problem behandelt der nächste Codeausschnitt der Ellipsenfunktion – das Zeichnen des Randes:

```
    // . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
    // draw border
    if ( g_DRAW_uDrawStyle & DRAWSTYLE_BORDER )
    {
        cmg_mu8                     uLoop;

        // set pen/solid
        _DRAW_ApplyPen( 0 );

        // . . . . . . . . . . . . . . . . . . . . . . .
        // only solid color
        // (can parallely draw pixels)
        if ( g_DRAW_uColorMode == DRV_COLORMODE_SOLIDCOLOR )
        {
            // init ellipse loop
            _Ellipse_LoopInside_Init( &EllData, iMidX, iMidY, iRadiusX, iRadiusY );

            // for all pixels
            while ( EllData.iCount > 0 )
            {
                // draw the first two quadrants simultaneously
                _DRAW_ClippedPixel( EllData.iX_Right, EllData.iY_Top );
                _DRAW_ClippedPixel( EllData.iX_Left, EllData.iY_Bottom );

                // update loop
                _Ellipse_LoopInside( &EllData );

                // draw the last two quadrants simultaneously
                _DRAW_ClippedPixel( EllData.iX_Right, EllData.iY_Bottom );
                _DRAW_ClippedPixel( EllData.iX_Left, EllData.iY_Top );
            }
        }


        // . . . . . . . . . . . . . . . . . . . . . . .
        // with pen
        // (must draw pixels in order to prevent pen)
        else
        {
            // we need two passes to draw all quadrants in order
            for ( uLoop = 0; uLoop < 2; uLoop++ )
            {
                // draw: (uLoop==0): Q1  /  (uLoop==1): Q3
                // init ellipse loop
                _Ellipse_LoopInside_Init( &EllData, iMidX, iMidY, iRadiusX, iRadiusY );
                // for all pixels
                while ( EllData.iCount > 0 )
                {
                    if ( uLoop == 0 )   // Q1
                        _DRAW_ClippedPixel( EllData.iX_Right, EllData.iY_Top );
                    else                // Q3
                        _DRAW_ClippedPixel( EllData.iX_Left, EllData.iY_Bottom );

                    _Ellipse_LoopInside( &EllData );
                }

                // draw: (uLoop==0): Q2  /  (uLoop==1): Q4
                // init ellipse loop
                _Ellipse_LoopOutside_Init( &EllData, iMidX, iMidY, iRadiusX, iRadiusY );
                // for all pixels
                while ( EllData.iCount > 0 )
                {
                    if ( uLoop == 0 )   // Q2
                        _DRAW_ClippedPixel( EllData.iX_Right, EllData.iY_Bottom );
                    else                // Q4
                        _DRAW_ClippedPixel( EllData.iX_Left, EllData.iY_Top );
                    _Ellipse_LoopOutside( &EllData );
                }
            }
        }
    }
```

Listing 126 – CMG_DRAW_Ellipse.c, Teil 2

Im oberen Block wird die komplette Ellipse mit nur einem Durchgang gezeichnet, wenn eine konstante Farbe verwendet werden soll. Im unteren Bereich – mit Stift – werden die zuvor besprochenen vier Durchgänge benötigt.

Beispiel:
Es folgen ein paar Ellipsen mit verschiedenen Füllmodi:

```
    CMG_SetDrawStyle( DRAWSTYLE_BORDER );
    CMG_SetColor( 1 );
    CMG_Ellipse( 20, 10, 15, 7 );
    CMG_SetDrawStyle( DRAWSTYLE_FILL );
    CMG_SetFillColor( 1 );
    CMG_Ellipse( 20, 40, 10, 20 );
```

```
    CMG_SetDrawStyle( DRAWSTYLE_BORDER );
    CMG_SetPen( &PenDashDotDot );
    CMG_Ellipse( 60, 10, 15, 7 );
    CMG_SetDrawStyle( DRAWSTYLE_FILL );
    CMG_SetBrush( &Brush50 );
    CMG_Ellipse( 60, 40, 10, 20 );

    CMG_SetDrawStyle( DRAWSTYLE_BOTH );
    CMG_SetPen( &PenDashDotDot );
    CMG_SetBrush( &BrushLight );
    CMG_Ellipse( 150, 10, 30, 7 );
    CMG_SetColor( 1 );
    CMG_SetBrush( &BrushHelloWorld );
    CMG_Ellipse( 150, 40, 20, 20 );
```

Listing 127 – DRAW, Ellipse, Beispiel

Abbildung 74 – DRAW, Ellipse, Beispiel

9.10. DREIECKE

Nicht nur die gesamte 3D-Grafik baut darauf auf, sondern auch im 2-dimensionalen Raum lassen sich alle Formen in Dreiecke zerlegen; egal wie kompliziert die Formen auch sein mögen. Schon allein diese Tatsache spiegelt die Wichtigkeit der Dreiecksfunktion wider. Die Dreiecke sind deshalb die letzten soliden Körper der DRAW-Schicht.

Beginnen wir gleich mit dem ersten Codeabschnitt – dem Initialisieren und dem Zeichnen des Randes:

```
_PUBLIC
void CMG_Triangle(cmg_Coord iX1, cmg_Coord iY1, cmg_Coord iX2, cmg_Coord iY2, cmg_Coord iX3, cmg_Coord iY3)
{
    // display orientation
    _DRAW_AdjustXY( &iX1, &iY1 );
    _DRAW_AdjustXY( &iX2, &iY2 );
    _DRAW_AdjustXY( &iX3, &iY3 );

    // . . . . . .
    // border
    // if to draw the border its very easy
    if ( g_DRAW_uDrawStyle & DRAWSTYLE_BORDER )
    {
        // switch off coord adjustments
        g_DRAW_bProhibitCoordAdjustment = true;

        // just connect the three points
        CMG_LineEx( iX1, iY1, iX2, iY2, LINESUPPRESS_LAST );
        CMG_LineEx( iX2, iY2, iX3, iY3, LINESUPPRESS_LAST );
        CMG_LineEx( iX3, iY3, iX1, iY1, LINESUPPRESS_LAST );

        // switch on coord adjustments
        g_DRAW_bProhibitCoordAdjustment = false;
    }
[...]
```

Listing 128 – CMG_DRAW_Triangle.c, Teil 1

Dies ist der ausgesprochen einfache Teil. Wie schon beim Rechteck werden einfach alle Seiten nacheinander gezeichnet – jeweils ohne den letzten Punkt.

Das Füllen eines Dreiecks ist, vom Aufwand her, das genaue Gegenteil. Dazu betrachten wir zunächst ein Dreieck:

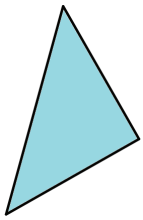


Abbildung 75 – DRAW, Triangle, ein Dreieck

Die einfachste, effektivste und nebenbei mit CMG auch einzig mögliche Art, ist das zeilenweise Füllen des Objekts. Dazu teilt man das Dreieck in zwei Teile: Den oberen und den unteren Teil. Die Trennstelle ist der mittlere Punkt:

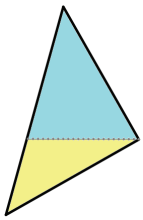


Abbildung 76 – DRAW, Triangle, Aufteilung

Zuerst müssen also die Eingabepunkte nach der Y-Koordinate sortiert werden. Der erste Punkt liegt am weitesten oben, der letzte ganz unten. Liegen zufällig zwei Punkte auf der gleichen Zeile, so entscheidet der X-Wert: Zuerst kommen die kleineren Werte, also die Punkte weiter links.

```
[...]
    // . . . . . . . . . . . . . . . . . . . . .
    // fill
    // if to draw the filled polygon
    if ( g_DRAW_uDrawStyle & DRAWSTYLE_FILL )
    {
        cmg_DRAW_Line_Data      LineData1, LineData2;
        cmg_Coord               iSpoolStartX;
        cmg_Coord               iLineY;
        cmg_Coord               iXLast1, iXLast2;

        // set brush/solid
        _DRAW_ApplyBrush( 0 );

        // . . . . . . . . . . . . . . . . . . . . .
        // sort points
        // sort points according their y value (and if equal their x value) (iLineY as temp)
        POINT_YX_SORT( iX1, iY1, iX2, iY2, iLineY );
        POINT_YX_SORT( iX2, iY2, iX3, iY3, iLineY );
        POINT_YX_SORT( iX1, iY1, iX2, iY2, iLineY );

        // . . . . . . . . . . . . . . . . . . . . .
        // setup start values
        // calc spool
        iSpoolStartX = MIN( iX1, MIN( iX2, iX3 ) );
        if ( g_DRAW_uDrawStyle & DRAWSTYLE_BORDER )
            iSpoolStartX++;
[...]
```

Listing 129 – CMG_DRAW_Triangle.c, Teil 2

Die Ausrichtung des Pinsels beginnt mit dem niedrigsten X-Wert. Wurde zusätzlich der Rand gezeichnet, muß dieser Wert noch um eins erhöht werden.

Zum eigentlichen Zeichnen verwendet der Algorithmus nun zwei Linien. Die Erste verbindet den obersten Punkt mit dem zweiten Punkt. Die Zweite verbindet den obersten mit dem dritten Punkt:

```
[...]
        // init lines: 1-2 and 1-3
        _Line_Loop_Init( &LineData1, iX1, iY1, iX2, iY2 );
        _Line_Loop_Init( &LineData2, iX1, iY1, iX3, iY3 );

        // start line
        iLineY = iY1;
[...]
```

Listing 130 – CMG_DRAW_Triangle.c, Teil 3

Jetzt wird für jede Zeile der Reihe nach folgendes vorgenommen:

- Solange die Y-Werte der ersten Linie noch auf dem alten Wert bleiben, wird die Linienfunktion aufgerufen, um neue Punkte auf der ersten Linie zu bekommen. Danach hat man einen Punkt des Dreiecks, der in der nächsten Zeile liegt.

  Sollte während dieser Schleife der Endpunkt der Linie erreicht werden – man also in Punkt zwei ankommt – wird die Linie neu initialisiert, diesmal von diesem zweiten Punkt hin zum letzten und dritten Punkt.

- Die gleiche Prozedur wie für die erste Linie wird auch für die zweite ausgeführt. Jedoch muß hier nicht auf den Endpunkt kontrolliert werden, da die zweite Linie immer bis zum untersten Punkt geht (Sortierung).

- Danach wird – je nach Füllstil – der komplette oder, wenn der Rahmen mitgezeichnet wurde, nur der innere Bereich gefüllt. Da jedoch die Reihenfolge der beiden Punkte nicht feststeht muß diese zuerst bestimmt werden, da die Funktion `_DRAW_ClippedHLine(…)` den linken Wert zuerst erwartet.

- Abgebrochen wird die Schleife, sobald die aktuelle Zeile die Endzeile erreicht oder – je nach Füllmodus – darüber hinausreicht.

Hier der komplette Code der Schleife und damit auch das Ende der Funktion:

```
[...]
        // . . . . . . . . . . . . . . . . . . . . . .
        // draw
        // for each line
        while ( 1 )
        {
            // step line1 to next y line point
            while ( LineData1.iY <= iLineY )
            {
                // save last x point
                iXLast1 = LineData1.iX;

                // if still points are on the line, get next point, else init line 2-3
                if ( LineData1.iCount > 0 )
                    _Line_Loop( &LineData1 );
                else
                    _Line_Loop_Init( &LineData1, iX2, iY2, iX3, iY3 );
            }

            // step line2 to next y line point
            while ( LineData2.iY <= iLineY )
            {
                // save last x point
                iXLast2 = LineData2.iX;
                _Line_Loop( &LineData2 );
            }

            // next line
            iLineY++;
```

```
            // did we draw the border?
            if ( g_DRAW_uDrawStyle & DRAWSTYLE_BORDER )
            {
                // only fill the inside of the triangle (prevent border)
                if ( LineData1.iX <= LineData2.iX )
                    _DRAW_ClippedHLine( (cmg_Coord)( LineData1.iX + 1 ), (cmg_Coord)( LineData2.iX - 1 ),
                            iLineY, iSpoolStartX );
                else
                    _DRAW_ClippedHLine( (cmg_Coord)( LineData2.iX + 1 ), (cmg_Coord)( LineData1.iX - 1 ),
                            iLineY, iSpoolStartX );

                // triangle end check
                if ( iLineY >= iY3 )
                    break;
            }
            else
            {
                // completely fill the triangle

                // sort x values and draw complete line
                if ( iXLast1 <= iXLast2 )
                    _DRAW_ClippedHLine( iXLast1, iXLast2, (cmg_Coord)( iLineY - 1 ), iSpoolStartX );
                else
                    _DRAW_ClippedHLine( iXLast2, iXLast1, (cmg_Coord)( iLineY - 1 ), iSpoolStartX );

                // triangle end check
                // we need an extra step since we are drawing (iLineY-1)
                if ( iLineY > iY3 )
                    break;
            }
        }
    }
}
```

Listing 131 – CMG_DRAW_Triangle.c, Teil 4

Mit diesem Algorithmus lassen sich auch die Sonderformen von Dreiecken korrekt zeichnen. Sonderformen liegen vor, wenn sich zum Beispiel alle Punkte auf einer Zeile, einer Spalte oder generell einer Linie befinden.

Andere Sonderformen entstehen, wenn der erste und zweite bzw. der zweite und der dritte Punkt in derselben Zeile liegen. Dann kann das Dreieck nämlich nicht in beide Hälften aufgeteilt werden, sondern besteht nur aus einer:

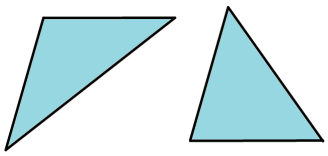


Abbildung 77 – DRAW, Triangle, Sonderformen

Der oben abgebildete Algorithmus zeichnet diese jedoch trotzdem korrekt, da er zeilenweise funktioniert und sich in der ersten bzw. letzten Hälfte nur die X-Werte ändern, also keine Ausgabe stattfindet.

Beispiel:
Nachfolgend ein paar Dreiecke inklusive der Sonderformen:

```
    CMG_SetDrawStyle( DRAWSTYLE_FILL );
    CMG_SetFillColor( 1 );
    CMG_Triangle( 2, 2, 20, 2, 5, 30 );
    CMG_Triangle( 23, 25, 20, 55, 10, 55 );

    CMG_Triangle( 60, 5, 35, 35, 70, 55 );
    CMG_Triangle( 80, 5, 140, 25, 100, 60 );

    CMG_SetDrawStyle( DRAWSTYLE_BOTH );
    CMG_SetColor( 1 );
    CMG_SetBrush( &Brush50 );
    CMG_Triangle( 130, 5, 180, 25, 170, 60 );

    CMG_SetPen( &PenDashDotDot );
    CMG_SetBrush( &BrushHelloWorld );
    CMG_Triangle( 239, 5, 180, 15, 200, 60 );
```

Listing 132 – DRAW, Triangle, Beispiel

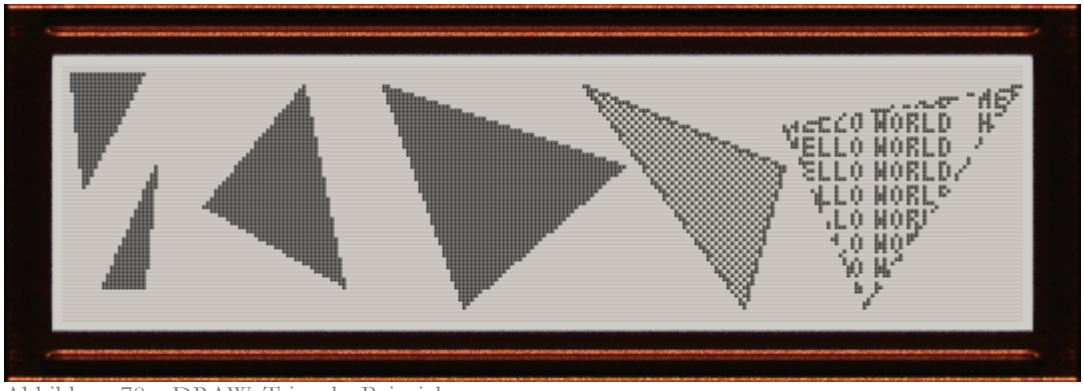


Abbildung 78 – DRAW, Triangle, Beispiel

9.11. Bildausschnitte, Images

Die letzte, noch grundlegende Ausgabeform ist das Zeichnen von Pixelbildern. Somit lassen sich beliebige Grafiken – von kleinen Symbolen bis hin zu großen Fotos – darstellen.

CMG unterstützt drei Zeichenfunktionen für Pixelbilder:

```
void CMG_Image( cmg_Coord iDestX, cmg_Coord iDestY, cmg_Image *pImage );

void CMG_ImageTruncated( cmg_Coord iDestX, cmg_Coord iDestY, cmg_Coord iSizeX, cmg_Coord iSizeY, cmg_Image
                         *pImage, cmg_Coord iSrcStartX, cmg_Coord iSrcStartY );

void CMG_ImageScaled( cmg_Coord iDestX, cmg_Coord iDestY, cmg_Coord iDestSizeX, cmg_Coord iDestSizeY,
                         cmg_Image *pImage );
```

Listing 133 – CMG_DRAW_Image.c, Deklarationen

- `CMG_Image(…)`: Zeichnet das übergebene Bild `pImage` in Originalgröße an der Koordinate (`iDestX`, `iDestY`).

- `CMG_ImageTruncated(…)`: Zeichnet einen Teilausschnitt aus dem übergebenen Bild `pImage` – ebenfalls in Originalgröße – an der Koordinate (`iDestX`, `iDestY`). Der Teilausschnitt beginnt im Quellbild bei der Koordinate (`iSrcStartX`, `iSrcStartY`) und hat eine Größe von (`iSizeX`, `iSizeY`).

- `CMG_ImageScaled(…)`: Zeichnet das übergebene Bild `pImage` in skalierter Form an der Koordinate (`iDestX`, `iDestY`). Das Bild wird an die Zielgröße von (`iDestSizeX`, `iDestSizeY`) angepaßt.

Allen Funktionen gemeinsam ist der Parameter `pImage`. Dieser repräsentiert das eigentliche Pixelbild. Um so ein Bild zu erstellen, muß diese Funktion verwendet werden:

```
_PUBLIC
void CMG_CreateImage( cmg_Image *pImage, cmg_u8 *pbyImageStartAddr, cmg_Coord iImageWidth, cmg_Coord
                         iImageHeight, cmg_mu16 uImageLineLengthBytes )
{
    pImage->pbyImageStartAddr       = pbyImageStartAddr;
    pImage->iImageWidth             = iImageWidth;
    pImage->iImageHeight            = iImageHeight;
    pImage->uImageLineLengthBytes   = uImageLineLengthBytes;
}
```

Listing 134 – CMG_DRAW_Image.c, Teil 1

Sie liefert die fertig initialisierte Image-Datenstruktur in `pImage` zurück und erwartet als Parameter die Startadresse der Pixeldaten im Speicher, die Breite sowie die Höhe in Pixeln und den Zeilenabstand in Bytes.

Das Zeichnen geschieht nun wie folgt:

```
/*****************************************************************************
| CMG_Image
|----------------------------------
| bit blitting
+----------------------------------------------------------------------------
| @Params:
|    * iDestX:               destination x point
|    * iDestY:               destination y point
|    * pImage:               image
| @Return Value:    none
+*****************************************************************************/
_PUBLIC
void CMG_Image( cmg_Coord iDestX, cmg_Coord iDestY, cmg_Image *pImage )
{
    cmg_DRAW_Image_StateBackup  StateBackup;
    cmg_Brush                   ImageBrush;

    // . . . . . . . . . . . . . . . . .
    // save state
    _Image_StateBackup_Save( &StateBackup );

    // . . . . . . . . . . . . . . . . .
    // brush and style
    // create image brush
    CMG_CreateBrush( &ImageBrush, pImage->pbyImageStartAddr, pImage->uImageLineLengthBytes, pImage-
                        >iImageHeight );

    // set brush
    CMG_SetBrush( &ImageBrush );

    // set draw style
    CMG_SetDrawStyle( DRAWSTYLE_FILL );

    // . . . . . . . . . . . . . . . . .
    // draw
    CMG_Rectangle( iDestX, iDestY, (cmg_Coord)( iDestX + pImage->iImageWidth - 1 ), (cmg_Coord)( iDestY +
                        pImage->iImageHeight - 1 ) );

    // . . . . . . . . . . . . . . . . .
    // restore state
    _Image_StateBackup_Restore( &StateBackup );
}
```

Listing 135 – CMG_DRAW_Image.c, Teil 2

Was sofort auffällt ist, daß die Funktion von den Aufrufen `_Image_StateBackup_Save(…)` und `_Image_StateBackup_Restore(…)` umgeben ist. Diese speichern die innerhalb der Funktion veränderten Werte ab und setzen sie danach wieder zurück. Die gespeicherten Werte sind der aktuelle Pinsel, die Position in diesem und der aktuelle Zeichenstil.

Zum Zeichnen selbst bedienen sich die Image-Funktionen der Pinsel-Funktionalität. Zuerst wird ein Pinsel mit den in `pImage` gespeicherten Daten erstellt, anschließend gesetzt und der Zeichenstil auf nur Füllen festgelegt. Jetzt zeichnet die Funktion ein ausgefülltes Rechteck an der angegebenen Position.

Die beiden anderen Bild-Zeichenfunktionen sind genau gleich aufgebaut, nur daß zusätzlich die Spooling-Werte bei der abgeschnittenen Version und die Scale-Werte bei der skalierten Version gesetzt werden. Für Details sehen Sie bitte direkt in den Quellcode.

Beispiel:
Nachfolgend ein paar Bilder in allen drei Variationen:

```
    // some characters - 8x14 (1 byte)
    cmg_u8    dataChar_C[] = { 0x00, 0x00, 0x00, 0x3c, 0x66, 0xc2, 0xc0, 0xc0, 0xc0, 0xc2, 0x66, 0x3c,
                               0x00, 0x00 };
    cmg_u8    dataChar_M[] = { 0x00, 0x00, 0x00, 0xc6, 0xee, 0xfe, 0xd6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6,
                               0x00, 0x00 };
    cmg_u8    dataChar_G[] = { 0x00, 0x00, 0x00, 0x3c, 0x66, 0xc2, 0xc0, 0xc0, 0xde, 0xc6, 0x66, 0x3a,
                               0x00, 0x00 };

    cmg_Image imageChar_C, imageChar_M, imageChar_G;

    CMG_CreateImage( &imageChar_C, dataChar_C, 8, 14, 1 );
    CMG_CreateImage( &imageChar_M, dataChar_M, 8, 14, 1 );
    CMG_CreateImage( &imageChar_G, dataChar_G, 8, 14, 1 );
```

```
    CMG_Image( 0, 0, &imageChar_C );
    CMG_Image( 10, 0, &imageChar_M );
    CMG_Image( 20, 0, &imageChar_G );

    CMG_ImageTruncated( 50, 0, 6, 8, &imageChar_G, 2, 5 );

    CMG_ImageScaled( 80, 0, 3, 6, &imageChar_C );
    CMG_ImageScaled( 80, 15, 4, 8, &imageChar_C );
    CMG_ImageScaled( 80, 30, 5, 9, &imageChar_C );
    CMG_ImageScaled( 80, 45, 6, 11, &imageChar_C );
    CMG_ImageScaled( 100, 0, 7, 13, &imageChar_C );
    CMG_ImageScaled( 100, 15, 8, 14, &imageChar_C );
    CMG_ImageScaled( 100, 30, 9, 16, &imageChar_C );
    CMG_ImageScaled( 100, 45, 10, 18, &imageChar_C );
    CMG_ImageScaled( 130, 0, 11, 20, &imageChar_C );
    CMG_ImageScaled( 130, 30, 12, 22, &imageChar_C );
    CMG_ImageScaled( 160, 0, 13, 24, &imageChar_C );
    CMG_ImageScaled( 160, 30, 14, 26, &imageChar_C );
    CMG_ImageScaled( 200, 0, 30, 60, &imageChar_C );
```

Listing 136 – DRAW, Image, Beispiel

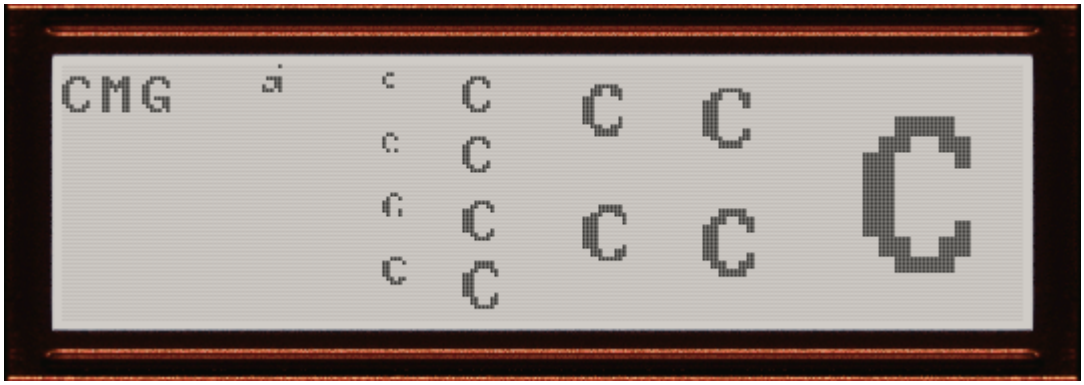


Abbildung 79 – DRAW, Image, Beispiel

9.12. BILDAUSSCHNITTE EINLESEN

Alle bisherigen Funktionen zeichneten Figuren oder Bilder und veränderten somit den Displayspeicher. Die hier behandelte Funktion `GetScreen(…)` macht jedoch genau das Gegenteil. Sie liest Daten aus dem Displayspeicher ein und speichert diese lokal, im Hauptspeicher, ab.

Sie benötigt folgende Parameter:

Parameter	Beschreibung
`pbyDestStartAddr`	Pointer auf den Start der Zieladresse in welcher der Abschnitt gespeichert wird.
`uDestAvailableBytes`	Maximale Größe des reservierten Speichers in `pbyDestStartAddr`. Die Funktion wird die Displaydaten nur dann ausgeben, wenn genug Speicherplatz vorhanden ist. Der Rückgabewert der Funktion ist die tatsächlich benötigte Anzahl an Bytes.
`puDestBytesPerLine`	Nur Ausgabeparameter. Wird kein NULL-Pointer übergeben, setzt die Funktion hier die Anzahl an Bytes pro Ausgabezeile.
`iSrcX`	X-Startwert im Display.
`iSrcY`	Y-Startwert im Display.
`iSizeX`	Breite des einzulesenden Abschnitts.
`iSizeY`	Höhe des einzulesenden Abschnitts.

Tabelle 38 – CMG_DRAW_GetScreen.c

Nach der Berechnung aller notwendigen Daten werden zeilenweise die Displaydaten mit `DRV_GetHLine(…)` eingelesen:

```
/*****************************************************************************
| CMG_GetScreen
|----------------------------------
| Get bitmap from screen
+-----------------------------------------------------------------------------
| @Params:
|   * pbyDestStartAddr:     destination start address (can be NULL, then the
|                           needed memory bytes will be returned in
|                           *puMemSizeBytes)
|   * uDestAvailableBytes:  available memory bytes in buffer
|   * puDestBytesPerLine:   linesize in bytes (out only, can be NULL)
|   * iSrcX:                x start
|   * iSrcY:                y start
|   * iSizeX:               x size to draw
|   * iSizeY:               y size to draw
| @Return Value:    actually needed memory bytes in buffer. If an error
|                   occures (insufficent mem or clipping error) the return
|                   value will be 0)
+*****************************************************************************/
_PUBLIC
cmg_mu16 CMG_GetScreen( cmg_u8 *pbyDestStartAddr, cmg_mu16 uDestAvailableBytes, cmg_mu16
                        *puDestBytesPerLine, cmg_Coord iSrcX, cmg_Coord iSrcY, cmg_Coord iSizeX,
                        cmg_Coord iSizeY )
{
    cmg_mu16    uNeededMem, uBytesPerLineSize;

    // display orientation
    _DRAW_AdjustXY( &iSrcX, &iSrcY );

    // adjust x size
    if ( iSizeX < 0 )
    {
        iSrcX -= iSizeX;
        iSizeX = -iSizeX;
    }
    // adjust y size
    if ( iSizeY < 0 )
    {
        iSrcY -= iSizeY;
        iSizeY = -iSizeY;
    }

    // clipping (regardless of current clipwindow, clip on whole screen)
    if ( ( iSrcX < 0 ) || ( iSrcY < 0 ) ||
        ( ( iSrcX + iSizeX ) >= g_DRAW_iScreenSizeX ) ||
        ( ( iSrcY + iSizeY ) >= g_DRAW_iScreenSizeY ) )
    {
        // return 0 for error
        return 0;
    }


    // calc linesize and needed bytes
    uBytesPerLineSize = CMG_DRV_FULL_BYTES_FROM_X( iSizeX );
    uNeededMem = uBytesPerLineSize * iSizeY;

    // if caller wants to know the linesize in bytes
    if ( puDestBytesPerLine )
        *puDestBytesPerLine = uBytesPerLineSize;

    // no destination addr given? just return needed size
    if ( !pbyDestStartAddr )
        return uNeededMem;

    // check for enough memsize; return 0 for error
    if ( uDestAvailableBytes < uNeededMem )
        return 0;


    // for every line
    while ( iSizeY-- )
    {
        // set pen
        CMG_DRV_GetHLine( pbyDestStartAddr, iSrcX, iSrcY++, iSizeX );

        // update destination start (next line)
        pbyDestStartAddr += uBytesPerLineSize;
    }

    // return needed mem
    return uNeededMem;
}
```

Listing 137 – CMG_DRAW_GetScreen.c

9.13. SCROLLING

Die letzte Funktion in der DRAW-Schicht ist das Scrolling. Sie vereint beide Bereiche: Das Einlesen und das Ausgeben des Displayspeichers.

Mit Hilfe der Funktion soll ein ganzer Bildschirmbereich an eine neue Position verschoben werden. Dies ist zum Beispiel zum Verschieben eines Fensters an eine neue Position nötig.

Die einfachste Lösung wäre, daß man den kompletten Bereich mit `GetScreen(…)` einliest und anschließend mit `Image(…)` wieder am neuen Ort zeichnet. Dafür ist aber – je nach Ausschnittgröße – relativ viel Speicher notwendig und speziell bei kleinen Mikroprozessoren nicht vorhanden.

Die Lösung ist einfach: Es wird nur zeilenweise verschoben. Hierbei muß allerdings die Verschieberichtung beachtet werden, damit die verschobenen Zeilen nicht noch später einzulesende Zeilen überschreiben. Die Regel ist einfach: Wenn der Y-Zielpunkt weiter unten liegt, also nach unten verschoben werden soll, dann muß zeilenweise von unten nach oben verschoben werden. Liegt der Y-Zielpunkt weiter oben, also soll nach oben verschoben werden, muß zeilenweise von oben nach unten verschoben werden. Somit ist garantiert, daß sich keine Zeilen überschreiben.

Hier der Code der Funktion. Der notwendige Speicher, für maximal eine Zeilenbreite, wird in der lokalen Variablen `abyLineBuffer` bereitgestellt:

```
/*************************************************************************
| CMG_Scroll
|------------------------------------
| moves a display range to another (used for scrolling windows)
+-------------------------------------------------------------------------
| @Params:
|   * iDestX:               destination x point
|   * iDestY:               destination y point
|   * iSrcX:                source x point
|   * iSrcY:                source y point
|   * iSizeX:               x size
|   * iSizeY:               y size
| @Return Value:    none
+*************************************************************************/
_PUBLIC
void CMG_Scroll( cmg_Coord iDestX, cmg_Coord iDestY, cmg_Coord iSrcX, cmg_Coord iSrcY, cmg_Coord iSizeX,
                 cmg_Coord iSizeY )
{
    // buffer to hold one line maximum
    cmg_u8      abyLineBuffer[ CMG_DRV_FULL_BYTES_FROM_X( CMG_CTRL_WIDTH ) ];
    cmg_Coord   iStepY;

    // display orientation
    _DRAW_AdjustXY( &iDestX, &iDestY );

    // param check
    if ( ( iSizeX <= 0 ) || ( iSizeY <= 0 ) )
        return;

    // set rop copy
    CMG_DRV_SetROPMode( ROPMODE_COPY );

    // get moving y direction
    if ( iDestY > iSrcY )
    {
        // move src downwards: start downmost and go upwards
        iSrcY  += iSizeY - 1;
        iDestY += iSizeY - 1;
        iStepY = -1;
    }
    else
    {
        // move src upwards: start uppermost (set already) and go downwards
        iStepY = 1;
    }

    // move each line
    while ( iSizeY-- )
    {
        // get src line
        CMG_DRV_GetHLine( abyLineBuffer, iSrcX, iSrcY, iSizeX );
```

```
        // draw dest line
        CMG_DRV_SetPen( abyLineBuffer, CMG_DRV_FULL_BYTES_FROM_X( CMG_CTRL_WIDTH ), DRV_SCALE_NONE );
        CMG_DRV_HLine( iDestX, iDestY, iSizeX );

        // adjust current line
        iSrcY  += iStepY;
        iDestY += iStepY;
    }

    // restore rop mode
    CMG_DRV_SetROPMode( g_DRAW_uROPMode  );
}
```

Listing 138 – CMG_DRAW_Scroll.c

9.14. SPEZIALEFFEKTE

Bisher wurden die erweiterten Möglichkeiten der Stifte und Pinsel noch nicht genauer betrachtet. Man kann bei Stiften und Pinseln den Startpunkt der Textur verschieben (*Spooling*) und ihre Größe selbst festlegen (*Scaling*). Damit lassen sich sehr interessante und umfangreiche Effekte verwirklichen.

9.14.1. SPOOLING

Stellen Sie sich einen Schaltplan der Elektronik, Pneumatik, Hydraulik oder einen Wasserkreislauf vor – einfach einen Schaltplan auf dem Teile fließen sollen.

Gegeben sind zum Beispiel vier Objekte, welche miteinander durch Leitungen verbunden sind. Zwischen zwei Objekten fließt nun irgend etwas in eine bestimmte Richtung. Diese Aufgabe soll grafisch dargestellt werden:

```
    // four objects
    CMG_SetDrawStyle( DRAWSTYLE_BORDER );
    CMG_SetColor( 1 );
    CMG_Rectangle( 0, 0, 10, 10 );
    CMG_Rectangle( 0, 20, 10, 30 );
    CMG_Rectangle( 60, 0, 70, 10 );
    CMG_Rectangle( 60, 20, 70, 30 );

    // draw static connection 1 & 2
    CMG_SetPen( &PenDashDotDot );
    CMG_Line( 11, 5, 59, 5 );
    CMG_Line( 11, 25, 59, 25 );

    CMG_SetROPMode( ROPMODE_XOR );

    while ( 1 )
    {
        // clear old connection
        CMG_Line( 11, 25, 59, 25 );

        // draw new connection
        PenDashDotDot.iPenSpoolPixels++;
        CMG_Line( 11, 25, 59, 25 );

        CMG_Sleep_ms( 100 );
    }
```

Listing 139 – Spooling Pen Demo 1

Mit Hilfe des XOR-ROP-Modes wird die alte Verbindung gelöscht und anschließend mit einem um eins erhöhten Spooling-Wert neu gezeichnet. Das bedeutet, daß sich alle 100ms der Inhalt der Linie um einen Pixel nach links bewegt.

Bei so einem einfachen Beispiel müßte der XOR-Mode nicht einmal verwendet werden. Es würde ausreichen die Linie nur komplett neu zu zeichnen:

```
    // four objects
    CMG_SetDrawStyle( DRAWSTYLE_BORDER );
    CMG_SetColor( 1 );
    CMG_Rectangle( 0, 0, 10, 10 );
    CMG_Rectangle( 0, 20, 10, 30 );
    CMG_Rectangle( 60, 0, 70, 10 );
    CMG_Rectangle( 60, 20, 70, 30 );

    // draw static connection 1 & 2
    CMG_SetPen( &PenDashDotDot );
    CMG_Line( 11, 5, 59, 5 );

    while ( 1 )
    {
        // draw new connection
        PenDashDotDot.iPenSpoolPixels++;
        CMG_Line( 11, 25, 59, 25 );

        CMG_Sleep_ms( 100 );
    }
```

Listing 140 – Spooling Pen Demo 2

Nachdem Animationen hier eher schlecht darstellbar sind, hoffe ich mit anschließender Bildfolge für Klarheit zu sorgen. Achten Sie auf die untere der beiden Verbindungslinien:

Abbildung 80 – Spooling Pen Demo

Das Gleiche was bei Stiften möglich ist, ist natürlich auch bei Pinseln möglich – hier allerdings in zwei Richtungen. Im Code können die Spooling-Werte der Pinsel jederzeit verändert werden.

Folgende Codezeilen verschieben den Text „*Hello World*“ bei jedem Aufruf um einen Pixel nach links unten.

```
    BrushHelloWorld.iBrushSpoolPixelsX++;
    BrushHelloWorld.iBrushSpoolPixelsY--;
```

Listing 141 – Spooling Brush Demo

9.14.2. SCALING

Die Stifte und Pinsel können auch skaliert werden. Dafür sind folgende Eigenschaften innerhalb der Objekte zuständig:

```
    Pen.sfpPenScaleFactor
    Brush.sfpBrushScaleFactorX
    Brush.sfpBrushScaleFactorY
```

Listing 142 – Scaling Eigenschaften

Diese Werte sind Festpunktwerte, welche den Skalierungsfaktor angeben. Sehen Sie sich dazu folgendes Beispiel an (um Platz zu sparen, stehen die Funktionen nebeneinander):

```
    CMG_SetPen( &PenDashDotDot );
    CMG_Line( 0, 0, 100, 0 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 0.9f );    CMG_Line( 0, 5, 100, 5 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 0.8f );    CMG_Line( 0, 10, 100, 10 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 0.7f );    CMG_Line( 0, 15, 100, 15 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 0.6f );    CMG_Line( 0, 20, 100, 20 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 0.5f );    CMG_Line( 0, 25, 100, 25 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 0.4f );    CMG_Line( 0, 30, 100, 30 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 0.3f );    CMG_Line( 0, 35, 100, 35 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 0.2f );    CMG_Line( 0, 40, 100, 40 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 0.1f );    CMG_Line( 0, 45, 100, 45 );


    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 1.1f );    CMG_Line( 130, 0, 230, 0 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 1.2f );    CMG_Line( 130, 5, 230, 5 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 1.3f );    CMG_Line( 130, 10, 230, 10 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 1.4f );    CMG_Line( 130, 15, 230, 15 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 1.5f );    CMG_Line( 130, 20, 230, 20 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 1.6f );    CMG_Line( 130, 25, 230, 25 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 1.7f );    CMG_Line( 130, 30, 230, 30 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 1.8f );    CMG_Line( 130, 35, 230, 35 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 1.9f );    CMG_Line( 130, 40, 230, 40 );
    PenDashDotDot.sfpPenScaleFactor = FLOAT_TO_SFP( 2.0f );    CMG_Line( 130, 45, 230, 45 );
```

Listing 143 – Scaling Pen Demo

Abbildung 81 – Scaling Pen Demo

Das Gleiche gilt natürlich auch für Pinsel – dort in X- und Y-Richtung separat einstellbar.

10. TEXT – TEXT RENDERING

10.1. AUFBAU

Die TEXT-Schicht bietet Unterstützung im Umgang mit der Textausgabe. Sie übernimmt die Aufgaben vom Erstellen und Laden der Schrift bis hin zum Rendern und Vermessen von Text.

Die folgende Abbildung listet die einzelnen Teile auf und zeigt deren Abhängigkeiten untereinander:

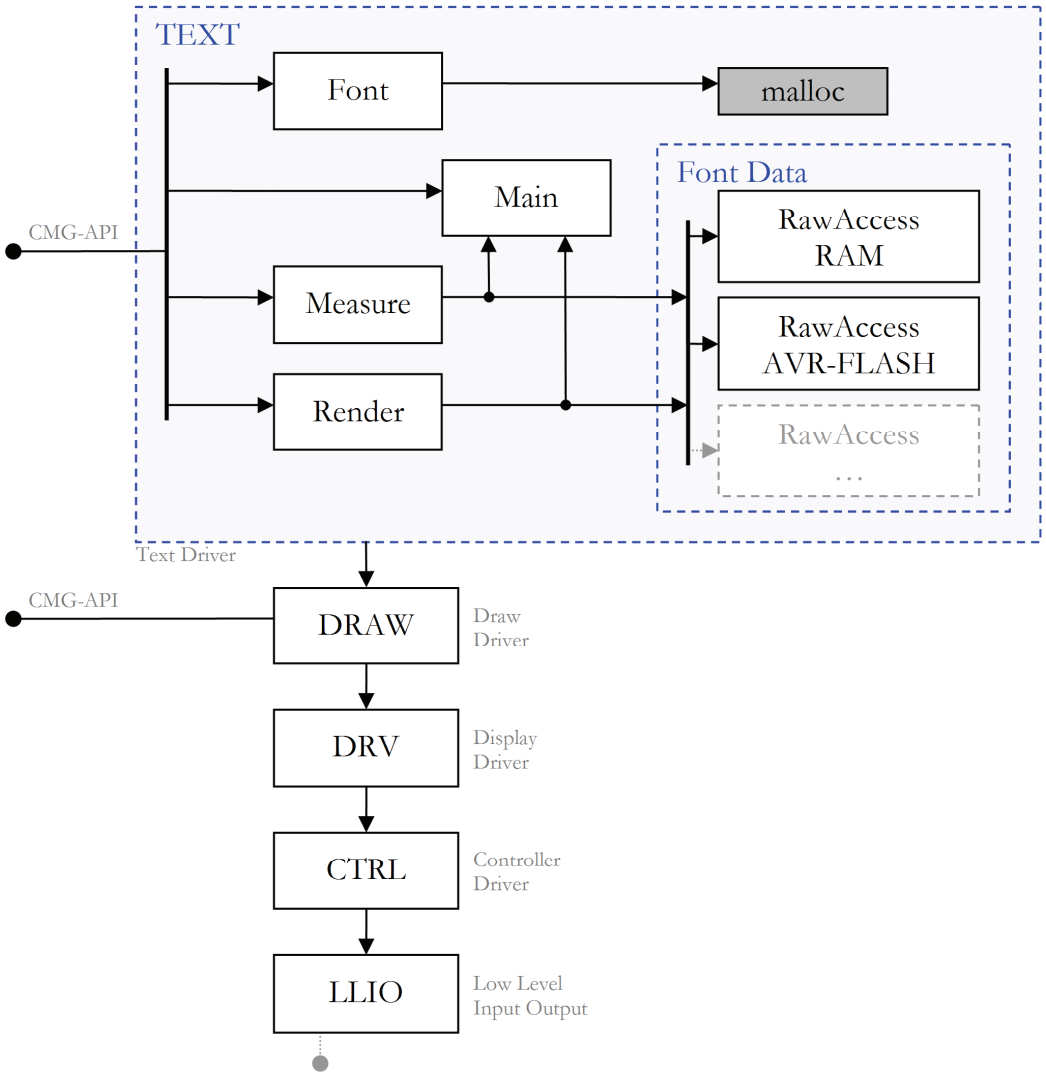


Abbildung 82 – TEXT Aufbau

Die einzelnen Teile besitzen folgende Aufgaben:

- **Main**
  Standard-Schicht-Funktionen wie das Initialisieren sowie allgemein gültige Funktionen, wie das Setzen der Farben oder Abprüfen bestimmter Zeichen auf Gültigkeit. Hier sind ebenfalls alle globalen Variablen definiert.

- **Font**
  Hier geschieht das Erstellen eines Font-Handles aus einer Schriftart sowie das Setzen dieses Handles, um die Schriftart zu verwenden.

- **Mem**
  Beim Setzen der Schriftart in dem eben beschriebenen Font-Teil muß zusätzlich ein Speicherbereich reserviert werden, der so groß ist, wie der größte vorkommende Buchstabe beansprucht. Da nicht alle Plattformen die gleiche malloc-Funktionalität besitzten, ist dieser Teil ausgelagert und kann bei Bedarf plattformabhängig ergänzt werden.

- **Measure**
  Mit Hilfe dieses Teils können Zeichen, Wörter oder ganze Texte vermessen werden. Man kann sowohl die Breite, als auch die Höhe der Textausgabe ermitteln und damit zum Beispiel den geeigneten Textumbruch bei mehrzeiligem Text bestimmen.

- **Render**
  In diesem Teil passiert die eigentliche Textausgabe – das Rendern von Zeichen und Text.

- **RawAccess**
  Jeder Zugriff auf die Rohdaten der Schriftart geschieht über die Funktionen dieses Teils. Der einfachste Fall ist, daß die gesamten Schriftdaten im RAM liegen. Gerade bei Mikrocontrollern mit geringem Hauptspeicher, ist dies aber meist nicht möglich. Hier wird die gesamte Schriftart im Code-Speicher – also im Flash – gehalten und nur bei Bedarf in einen Puffer kopiert.

10.2. SCHNITTSTELLEN

Die TEXT-Schicht besitzt ebenfalls wieder eine öffentliche und eine private, interne Schnittstelle.

10.2.1. ÖFFENTLICHE SCHNITTSTELLEN

Die öffentliche Schnittstelle beschreibt alle Funktionen und Datenstrukturen der Textausgabe.

Diesmal zuerst die Datenstrukturen der Schriftarten:

```
//---------------------------------------------------------------------------
// Font typedefs
//---------------------------------------------------------------------------
typedef     cmg_u8      cmg_FontRawData;

#define     CMG_FONTTYPE_UNCOMPRESSED       100
```

```
typedef struct _cmg_Font {
    cmg_u8              byFontType;                 // font compression type
    cmg_u8              byCharStart;                // first character index
    cmg_u8              byCharStop;                 // last character index
    cmg_u8              byLineSizeBytes;            // nr of bytes per char line
    cmg_u8              byHeight;                   // nr of lines per char
    cmg_mu16            uCharDataBytes;             // nr of data bytes per char
    cmg_mu16            uCharOffsetBytes;           // nr of complete bytes per char
    cmg_ms8             iCharSpacing;               // spacing pixels between each char
    cmg_FontRawData     *pFontRawCharDataStart;     // pointer to raw data of first char
} cmg_Font;
//---------------------------------------------------------------------------
```

Listing 144 – CMG_TEXT.h, Teil 1

Und hier sind die öffentlichen Funktionen aufgelistet:

```
  cmg_Result  CMG_TEXT_DrawAndTextInit( void );
  void        CMG_TEXT_Exit( void );

  void        CMG_TEXT_SetTextColor( cmg_Color TextColor );
  void        CMG_TEXT_SetBackColor( cmg_Color BackColor );
  cmg_bool    CMG_TEXT_IsValidChar( cmg_char cChar );

  void        CMG_TEXT_Font_Create( cmg_Font *pFont, cmg_FontRawData *pFontRawData, cmg_ms8 iCharSpacing );
  cmg_Result  CMG_TEXT_Font_Set( cmg_Font *pFont );

  cmg_Coord   CMG_TEXT_Measure_GetTextHeight( void );
  cmg_Coord   CMG_TEXT_Measure_GetTextWidth( cmg_char *strText );

  void        CMG_TEXT_Render( cmg_Coord iX, cmg_Coord iY, cmg_char *strText );
```

Listing 145 – CMG_TEXT.h, Teil 2

Hier sieht man deutlich zu welchen Teilen die Funktionen gehören: Main ist ohne ein Präfix, die anderen Teile haben ihren Namen als Präfix.

10.2.2. TEXT-INTERNE SCHNITTSTELLEN

Die internen Schnittstellen beschränken sich auf die Verwendung der RawAccess-, der Mem- und der Font-Funktionen sowie die Bekanntmachung der gemeinsamen, globalen Variablen:

```
//---------------------------------------------------------------------------
// globals
//---------------------------------------------------------------------------
extern cmg_Font    *g_TEXT_pFont;

extern cmg_Color   g_TEXT_TextColor;
extern cmg_Color   g_TEXT_TextBackColor;

extern cmg_Image   g_TEXT_ImageChar;
extern cmg_u8      *g_TEXT_pbyImageCharData;
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Declarations      (for details see below)
//---------------------------------------------------------------------------
    void        _TEXT_FontRawAccess_SetBaseAddress( cmg_FontRawData *pBaseAddress );
    void        _TEXT_FontRawAccess_SetOffset( cmg_mu16 uOffset );
    cmg_u8      _TEXT_FontRawAccess_GetByte( void );
    void        _TEXT_FontRawAccess_SaveState( void );
    void        _TEXT_FontRawAccess_RestoreState( void );

    cmg_Result  _TEXT_Mem_Alloc( cmg_u8 **ppbyData, cmg_mu16 uSize );
    void        _TEXT_Mem_Free( cmg_u8 **ppbyData );

    void        _TEXT_Font_Free( void );
//---------------------------------------------------------------------------
```

Listing 146 – CMG_TEXT_Internal.h, Internes Interface

10.3. KONFIGURATION

Die Konfiguration der TEXT-Schicht ist sehr einfach:

```
//#############################################################################
// (5) CMG_TEXT - CONFIG
//#############################################################################
// CMG_TEXT_*
// ======================
// TEXT configuration.
//
// For internal correct compilation.
// *** DO NOT CHANGE! ***
#define     CMG_TEXT_CONFIGURED
//
//=============================================================================
// User Area:
// ==========
// Read the sections carefully and change the values accordingly to
// your needs.
//
// Define to enable Text Rendering
// -------------------------------
#define   CMG_TEXT_ENABLED
//
// Select RawAccess mode for fonts
// -------------------------------
#define   CMG_TEXT_FONTRAWACCESS_RAM
// #define   CMG_TEXT_FONTRAWACCESS_AVRFLASH
//
//#############################################################################
```

Listing 147 – CMG_TEXT.config

Als erste Option steht `CMG_TEXT_ENABLE` zur Verfügung. Wird diese Zeile auskommentiert, wird keine der TEXT-Komponenten mehr kompiliert und somit – um Speicher zu sparen – die Textausgabe deaktiviert. Die zweite Option legt den Zugriffsmodus auf die Schriftdaten fest.

10.4. FONT-DATENSTRUKTUR

Um Schriftarten in CMG verwenden zu können, müssen diese genau nach der Font-Datenstruktur aufgebaut sein.

Bis jetzt ist nur die unkomprimierte Datenstruktur in CMG implementiert. Hier könnte man allerdings – einen geeigneten Kompressionsalgorithmus vorausgesetzt – noch einen Großteil der notwendigen Daten einsparen. Jedoch ergeben sich dann wieder neue Schwierigkeiten, wie z.B. das Zeichenoffset im Speicher. Bei der unkomprimierten Speicherung sind jetzt alle Zeichen gleich groß und somit läßt sich die Startadresse eines jeden Zeichens leicht durch eine Multiplikation ermitteln. Bei verschieden großen Zeichen muß aber ein anderer Weg gefunden werden, wie z.B. eine zusätzliche Tabelle mit den Startwerten, welche aber wiederum zusätzlichen Speicher benötigt. Zusammenfassend also keine leichte Angelegenheit.

10.4.1. AUFBAU

Eine Font-Datenstruktur ist vom Typ `cmg_FontRawData`. Diese ist in der TEXT-Schnittstellendefinition als einfaches Byte definiert. Somit ist die Datenstruktur eine Bytefolge.

Eine Schriftart ist wie folgt aufgebaut:

```
typedef struct _cmg_RawFont {
    cmg_RawFont_Header  FontHeader;                 // font header
    cmg_RawFont_Char    Char[];                     // chars
} cmg_RawFont;
```

```
typedef struct _cmg_RawFont_Header {
    cmg_u8              byFontType;             // font compression type
    cmg_u8              byCharStart;            // first character index
    cmg_u8              byCharStop;             // last character index
    cmg_u8              byLineSizeBytes;        // nr of bytes per char line
    cmg_u8              byHeight;               // nr of lines per char
} cmg_RawFont_Header;

typedef struct _cmg_RawFont_Char {
    cmg_u8              byCharWidth;            // char size in bytes
    cmg_u8              abyRawCharData[];       // char data
} cmg_RawFont_Char;
```

Listing 148 – TEXT, RawFont Schema

Hier noch einmal ausführlich in Tabellendarstellung:

Font-Datenstruktur:

Offset	Anzahl	Größe	Inhalt	Inhalt
0	1	5	`Header`	Font-Header
5	*Anzahl der Zeichen*	*Zeichenoffset*	`Char[]`	Array von Zeichen

Tabelle 39 – TEXT, RawFont Schema

Header-Datenstruktur:

Offset	Anzahl	Größe	Inhalt	Inhalt
0	1	1	`byFontType`	Kompressionstyp der Schriftart
1	1	1	`byCharStart`	Erste vorkommende Zeichen (ASCII-Wert)
2	1	1	`byCharStop`	Letzte vorkommende Zeichen (ASCII-Wert)
3	1	1	`byLineSizeBytes`	Anzahl der Bytes pro Zeichenzeile
4	1	1	`byHeight`	Anzahl der Zeilen pro Zeichen in Pixel

Tabelle 40 – TEXT, RawFont-Header Schema

Char-Datenstruktur:

Offset	Anzahl	Größe	Inhalt	Inhalt
0	1	1	`byCharWidth`	Breite des Zeichens in Pixel
1	*Anzahl der Datenbytes*	1	`abyCharData[]`	Array von Datenbytes

Tabelle 41 – TEXT, RawFont-Char Schema

In der Tabelle kommen bei Anzahl und Größe ein paar Werte vor, welche erst aus anderen berechnet werden müssen:

Anzahl der Zeichen = `byCharStop` – `byCharStart` + 1

Zeichenoffset = `sizeof( byCharWidth )` + *Anzahl der Datenbytes*

Anzahl der Datenbytes = `byLineSizeBytes` * `byHeight`

10.4.2. FONT-DATEIEN

Eine Font-Datei ist eine einfache C-Quellcode-Datei. Sie beinhaltet eine globale Variable für die Schriftart. Wo diese Daten abgespeichert sind (RAM, Flash, etc.) spielt keine Rolle, ebensowenig, wie diese eingelesen werden können. Darum kümmert sich die Komponente `RawAccess`.

Die Variable ist ein Array aus Bytes mit den eigentlichen Schriftdaten – aufgebaut nach dem oben beschriebenen Schema.

Nachfolgend eine gekürzte Fassung einer kleinen, von Windows importierten, Schriftart:

```
/*************************************************************\
| CMG Font: Tahoma 7
| Font created by CMGFontConvert (c) 2oo7 Christian Merkle
|-------------------------------------------------------------
| Font Type: uncompressed
| Start char: 32, stop char: 126, number of chars: 95
| Max char size: {Width=9, Height=11}
| Line size: 2 bytes
| Char size: 23 bytes
| Header size: 5 bytes
| Font size: 2190 bytes
\*************************************************************/

#include "CMG/CMG.h"

cmg_FontRawData    g_Font_Tahoma_7[] =
    {
        /* Font Header (type, start, stop, linesize, height): */
        0x64, 0x20, 0x7e, 0x02, 0x0b,
[...]
        /* Character Data "!" (width, 22 data bytes): */
        0x01, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x80, 0x00, 0x80, 0x00, 0x80, 0x00, 0x80, 0x00, 0x00,
                          0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00,
[...]
        /* Character Data "#" (width, 22 data bytes): */
        0x07, 0x00, 0x00, 0x00, 0x00, 0x24, 0x00, 0x24, 0x00, 0x7e, 0x00, 0x28, 0x00, 0xfc, 0x00, 0x48,
                          0x00, 0x48, 0x00, 0x00, 0x00, 0x00, 0x00,
[...]
        /* Character Data "A" (width, 22 data bytes): */
        0x06, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00, 0x30, 0x00, 0x48, 0x00, 0x48, 0x00, 0xfc, 0x00, 0x84,
                          0x00, 0x84, 0x00, 0x00, 0x00, 0x00, 0x00,

        /* Character Data "B" (width, 22 data bytes): */
        0x05, 0x00, 0x00, 0x00, 0x00, 0xe0, 0x00, 0x90, 0x00, 0x90, 0x00, 0xf0, 0x00, 0x88, 0x00, 0x88,
                          0x00, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x00,
[...]
        /* Character Data "W" (width, 22 data bytes): */
        0x07, 0x00, 0x00, 0x00, 0x00, 0x92, 0x00, 0x92, 0x00, 0xaa, 0x00, 0xaa, 0x00, 0xaa, 0x00, 0x44,
                          0x00, 0x44, 0x00, 0x00, 0x00, 0x00, 0x00,
[...]
        /* Character Data "Z" (width, 22 data bytes): */
        0x04, 0x00, 0x00, 0x00, 0x00, 0xf0, 0x00, 0x10, 0x00, 0x20, 0x00, 0x40, 0x00, 0x40, 0x00, 0x80,
                          0x00, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x00,
[...]
        /* Character Data "~" (width, 22 data bytes): */
        0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x64, 0x00, 0x98, 0x00, 0x00,
                          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
    };
```

Listing 149 – TEXT, RawFont-Schriftart, Beipiel

Beim Betrachten der Rohdaten wird schnell klar, daß hier noch ein großes Einsparpotential besteht – allerdings treten dann die oben schon beschriebenen Schwierigkeiten der Zeichengröße auf.

10.5. Komponenten der TEXT-Schicht

Die zu Beginn des Kapitels vorgestellten Komponenten der TEXT-Schicht sollen hier genauer beschrieben werden.

10.5.1. Main

In der Main-Komponente befinden sich die Funktionen `Init` und `Exit`, allerdings hat die Init-Funktion diesmal mit `CMG_TEXT_DrawAndTextInit` einen besonderen Namen. Nachdem die DRAW-Schicht schon eine Benutzer-API mit der Funktion `INIT` beinhaltet, ist diese Funktion in der TEXT-Schicht aussagekräftig benannt worden, um Mißverständnisse zu vermeiden.

Ruft der Benutzer also die Funktion `CMG_TEXT_DrawAndTextInit` auf, so wird sowohl DRAW mit allen untergeordneten Schichten initialisiert, als auch die eigentliche TEXT-Schicht.

Beim Initialisieren setzt die Funktion nur die Schrift und die Farbwerte zurück.

Beim Beenden der Schicht wird eine eventuell ausgewählte Schrift freigegeben.

Zusätzlich definiert Main noch zwei Funktionen zum Setzen der Text- sowie der Texthintergrundfarbe, welche die Werte zwar in den passenden, globalen Variablen abspeichern, aber diese Funktionalität jedoch noch nicht weiter implementieren.

Die letzte Funktion prüft, ob das übergebene Zeichen in der Schriftart definiert ist:

```
_PUBLIC
cmg_bool CMG_TEXT_IsValidChar( cmg_char cChar )
{
    if ( ( cChar >= g_TEXT_pFont->byCharStart ) && ( cChar <= g_TEXT_pFont->byCharStop ) )
        return true;

    return false;
}
```

Listing 150 – CMT_TEXT_Main.c

10.5.2. FONT

Die Font-Komponente implementiert die zum Laden, Setzen und Freigeben der Schriftart notwendigen Funktionen.

Die Funktion zum Laden einer Schriftart erwartet einen Pointer auf die Rohdaten der Schriftart sowie den Zeichenabstand zwischen den einzelnen Zeichen und gibt die geladene Schrift als `cmg_Font` zurück:

```
_PUBLIC
void CMG_TEXT_Font_Create( cmg_Font *pFont, cmg_FontRawData *pFontRawData, cmg_ms8 iCharSpacing )
{
    // save state
    _TEXT_FontRawAccess_SaveState();

    // set new font ptr
    _TEXT_FontRawAccess_SetBaseAddress( pFontRawData );

    // read font type
    pFont->byFontType = _TEXT_FontRawAccess_GetByte();
    if ( pFont->byFontType != CMG_FONTTYPE_UNCOMPRESSED )
        return;

    // read font header
    pFont->byCharStart = _TEXT_FontRawAccess_GetByte();
    pFont->byCharStop = _TEXT_FontRawAccess_GetByte();
    pFont->byLineSizeBytes = _TEXT_FontRawAccess_GetByte();
    pFont->byHeight = _TEXT_FontRawAccess_GetByte();

    // calculate other needed values
    pFont->uCharDataBytes = pFont->byLineSizeBytes * pFont->byHeight;
    pFont->uCharOffsetBytes = 1 + pFont->uCharDataBytes;
    pFont->pFontRawCharDataStart = pFontRawData + 5;

    // set spacing
    pFont->iCharSpacing = iCharSpacing;

    // restore state
    _TEXT_FontRawAccess_RestoreState();
}
```

Listing 151 – CMG_TEXT_Font.c, Teil 1

Die Funktion liest die Rohdaten des Headers aus der Schriftart ein und berechnet die restlichen, notwendigen Werte.

Ist eine Schriftart erst geladen, kann sie – wie auch Stifte und Pinsel – jederzeit mit `CMG_TEXT_Font_Set(…)` gesetzt werden. Diese Funktion gibt zuerst eine eventuell gesetzte, alte Schrift frei, setzt die globale Variable auf die neue Schrift und den Zugriffpointer auf den Buchstabenanfang. Anschließend muß noch der Speicher für das größte vorkommende Zeichen reserviert und ein Image als Hilfsmittel zum Zeichnen der einzelnen Zeichen erstellt werden:

```
_PUBLIC
cmg_Result CMG_TEXT_Font_Set( cmg_Font *pFont )
{
    cmg_Result  iResult;

    // first free any previously set font
    _TEXT_Font_Free();

    // set global var and char data start base ptr
    g_TEXT_pFont = pFont;
    _TEXT_FontRawAccess_SetBaseAddress( g_TEXT_pFont->pFontRawCharDataStart );

    // alloc image space for one char
    iResult = _TEXT_Mem_Alloc( &g_TEXT_pbyImageCharData, g_TEXT_pFont->byLineSizeBytes * g_TEXT_pFont-
                           >byHeight );
    CHECKRESULT();

    // create image for char drawing
    CMG_CreateImage( &g_TEXT_ImageChar, g_TEXT_pbyImageCharData, 1, g_TEXT_pFont->byHeight, g_TEXT_pFont-
                           >byLineSizeBytes );

    // return success
    return CMG_OK;

    //.............................
    // on error free everything
_Error:
    _TEXT_Font_Free();
    return iResult;
}
```

Listing 152 – CMG_TEXT_Font.c, Teil 2

Die letzte Funktion in Main wird nur intern aufgerufen und gibt eine gesetzte Schrift wieder frei. Dabei ist nur die Schrift auf Null zu setzen und der zuvor reservierte Speicherbereich wieder freizugeben:

```
_PROTECTED
void _TEXT_Font_Free( void )
{
    g_TEXT_pFont = NULL;
    _TEXT_Mem_Free( &g_TEXT_pbyImageCharData );
}
```

Listing 153 – CMG_TEXT_Font.c, Teil 3

10.5.3. MEM

Die Mem-Komponente kapselt die Funktionen zum Reservieren und Freigeben von Speicher auf dem Heap. Dies ist notwendig, da nicht alle Plattformen die gleichen malloc-Befehle unterstützen. Außerdem geben die Funktionen den Ergebnisstatus nicht im Pointer, sondern separat zurück.

Das Reservieren von Speicher sieht so aus:

```
_PROTECTED
cmg_Result _TEXT_Mem_Alloc( cmg_u8 **ppbyData, cmg_mu16 uSize )
{
    cmg_Result  iResult;
    cmg_u8      *pbyData;

    // param check
    CHECK( !ppbyData, CMG_ERROR_PARAM );

    // alloc data
    pbyData = (cmg_u8*) malloc( uSize );
    CHECK( !pbyData, CMG_ERROR_NOMEM );

    // set data ptr
    *ppbyData = pbyData;

    // return success
    return CMG_OK;

    //.............................
    // on error
_Error:
    return iResult;
}
```

Listing 154 – CMG_TEXT_Mem.c, Teil 1

Das Freigeben von Speicher geschieht wie folgt:

```
_PROTECTED
void _TEXT_Mem_Free( cmg_u8 **ppbyData )
{
    // just free data
    if ( ppbyData )
    {
        if ( *ppbyData )
            free( *ppbyData );

        *ppbyData = NULL;
    }
}
```

Listing 155 – CMG_TEXT_Mem.c, Teil 2

Hierbei wird der übergebene Pointer gleich genullt.

10.5.4. FontRawAccess

Der Zugriff auf die rohen Font-Daten wäre einfacher, wenn diese immer komplett im RAM liegen könnten. Kleine Mikrocontroller haben jedoch nicht ausreichend Platz dafür bzw. der begrenzte, vorhandene Platz soll sinnvoller benutzt werden.

Um jede denkbare Zugriffsart auf diese Daten zu ermöglichen, sind die Zugriffsfunktionen in die Komponente FontRawAccess ausgelagert.

Zur Zeit ist in CMG nur der direkte RAM-Zugriff implementiert, eine Anpassung an AVRs ist aber nicht schwer. Dabei sollen die Rohdaten nur im Flash – also im Befehlsspeicher – gespeichert und bei Bedarf daraus geladen werden.

Der Zugriff erfolgt über eine einfache Technik:

Es wird ein Basis-Pointer und ein Offset angegeben und gesetzt. Bei jedem Zugriff errechnet sich die aktuelle Speicherposition aus dem Basispointer + dem Offset. Nach dem Zugriff wird das Offset automatisch um eins erhöht. Es kann nur Byteweise auf den Speicher zugegriffen werden.

10.5.4.1. RAM

Der RAM-Zugriff ist sehr einfach, da die gewünschten Funktionen direkt abzubilden sind. Nachfolgend alle Funktionen in einem Listing zusammengefaßt:

```
//-----------------------------------------------------------------------------
// globals
//-----------------------------------------------------------------------------
cmg_FontRawData     *g_TEXT_FRA_pBaseAddress;
cmg_mu16            g_TEXT_FRA_uOffset;

cmg_FontRawData     *g_TEXT_FRA_pBaseAddress_Saved;
cmg_mu16            g_TEXT_FRA_uOffset_Saved;
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
// exportet functions
//-----------------------------------------------------------------------------
_PROTECTED
void _TEXT_FontRawAccess_SetBaseAddress( cmg_FontRawData *pBaseAddress )
{
    g_TEXT_FRA_pBaseAddress = pBaseAddress;
    g_TEXT_FRA_uOffset = 0;
}

_PROTECTED
void _TEXT_FontRawAccess_SetOffset( cmg_mu16 uOffset )
{
    g_TEXT_FRA_uOffset = uOffset;
}
```

```
_PROTECTED
cmg_u8 _TEXT_FontRawAccess_GetByte( void )
{
    return *( g_TEXT_FRA_pBaseAddress + g_TEXT_FRA_uOffset++ );
}

_PROTECTED
void _TEXT_FontRawAccess_SaveState( void )
{
    g_TEXT_FRA_pBaseAddress_Saved = g_TEXT_FRA_pBaseAddress;
    g_TEXT_FRA_uOffset_Saved = g_TEXT_FRA_uOffset;
}

_PROTECTED
void _TEXT_FontRawAccess_RestoreState( void )
{
    g_TEXT_FRA_pBaseAddress = g_TEXT_FRA_pBaseAddress_Saved;
    g_TEXT_FRA_uOffset = g_TEXT_FRA_uOffset_Saved;
}
//---------------------------------------------------------------------------
```

Listing 156 – CMG_TEXT_FontRawAccess_RAM.c

10.5.4.2. AVR-FLASH

Selbst die großen Mikrocontroller der AVR-Familie besitzen nur wenige Kilobytes an RAM, jedoch 32, 64 oder sogar 128kB Befehlsspeicher in Flash-Form. Ziel ist es, die Schriftdaten in diesem Speicher zu belassen und nur bei Bedarf in kleinen Blöcken einzulesen.

Für diese Aufgabe bieten die AVR-Bibliotheken komfortable Lösungen an. Bei der Implementierung muß allerdings beachtet werden, daß der Flashspeicher nur in 16-Bit-breiten Blöcken ausgelesen werden kann. Die Aufteilung in Basisadresse und Offset sollte jedoch keine zusätzlichen Schwierigkeiten bereiten.

10.5.5. MEASURE

Um zu berechnen, wie groß der zu zeichnende Text letztendlich wird, ist die Measure-Komponente zuständig. Dadurch kann z.B. der passende Zeilenumbruch in einem Text gefunden oder einfach auch nur die Abmessungen für das dahinterliegende Rechteck berechnet werden.

Die Komponente teilt sich in zwei Bereiche auf: Die Texthöhe und die Textbreite.

Die Berechnung der Texthöhe ist sehr einfach, denn diese Information liegt direkt in der Schriftart:

```
_PUBLIC
cmg_Coord CMG_TEXT_Measure_GetTextHeight( void )
{
    return (cmg_Coord) g_TEXT_pFont->byHeight;
}
```

Listing 157 – CMG_TEXT_Measure.c, Teil 1

Die Berechnung der Textbreite ist dafür etwas aufwendiger – die Funktion sieht wie folgt aus:

```
_PUBLIC
cmg_Coord CMG_TEXT_Measure_GetTextWidth( cmg_char *strText )
{
    cmg_Coord   iWidth = 0;
    cmg_char    c;

    // add each char width
    while ( ( c = *strText++ ) != NULL )
        iWidth += _TEXT_Measure_GetCharWidth( c ) + g_TEXT_pFont->iCharSpacing;

    return iWidth;
}
```

Listing 158 – CMG_TEXT_Measure.c, Teil 2

Sie ruft für jedes Zeichen im übergebenen String die private Hilfsfunktion `_TEXT_Measure_GetCharWidth(…)` auf und addiert den Wert für den Buchstabenabstand `iCharSpacing` dazu.

Die eigentliche Berechnung der Zeichenbreite übernimmt die Hilfsfunktion:

```
_PRIVATE
cmg_Coord _TEXT_Measure_GetCharWidth( cmg_char cChar )
{
    cmg_Coord   iWidth = 0;

    // if char is valid
    if ( CMG_TEXT_IsValidChar( cChar ) )
    {
        // set offset to char
        _TEXT_FontRawAccess_SetOffset( ( cChar - g_TEXT_pFont->byCharStart ) * g_TEXT_pFont-
                        >uCharOffsetBytes );
        // read char width in first byte
        iWidth = (cmg_Coord) _TEXT_FontRawAccess_GetByte();
    }

    return iWidth;
}
```

Listing 159 – CMG_TEXT_Measure.c, Teil 3

Diese prüft, ob das Zeichen überhaupt in der Schriftart definiert ist und setzt anschließend das Offset zum Zeichenanfang. Laut der Schriftartendefinition steht die Zeichenbreite im ersten Byte des Zeichens. Dieses wird eingelesen und zurückgeliefert.

10.5.6. RENDER

Kommen wir nun zum eigentlichen Hauptteil der TEXT-Schicht: Das Rendern von Text. Dies übernimmt die Funktion `CMT_TEXT_Render(…)`:

```
_PUBLIC
void CMG_TEXT_Render( cmg_Coord iX, cmg_Coord iY, cmg_char *strText )
{
    cmg_char    c;
    // render each char
    while ( ( c = *strText++ ) != NULL )
        iX += _TEXT_RenderChar( iX, iY, c ) + g_TEXT_pFont->iCharSpacing;
}
```

Listing 160 – CMG_TEXT_Render.c, Teil 1

Hier wird – wie schon bei der Berechnung der Textbreite – eine Hilfsfunktion für jedes Zeichen aufgerufen, welche das eigentliche Zeichnen übernimmt:

```
_PRIVATE
cmg_Coord _TEXT_RenderChar( cmg_Coord iX, cmg_Coord iY, cmg_char cChar )
{
    cmg_Coord   iWidth = 0;
    cmg_u8      *pbyData;
    cmg_mu16    uLength;

    // if char is valid
    if ( CMG_TEXT_IsValidChar( cChar ) )
    {
        // set offset to char data start
        _TEXT_FontRawAccess_SetOffset( ( cChar - g_TEXT_pFont->byCharStart ) * g_TEXT_pFont-
                        >uCharOffsetBytes );
        // and read char width
        iWidth = (cmg_Coord) _TEXT_FontRawAccess_GetByte();

        // prepare char image
        g_TEXT_ImageChar.iImageWidth = iWidth;
        pbyData = g_TEXT_ImageChar.pbyImageStartAddr;
        uLength = g_TEXT_pFont->uCharDataBytes;

        // read raw char data into image
        while ( uLength-- )
            *pbyData++ = _TEXT_FontRawAccess_GetByte();

        // draw image with char
        CMG_Image( iX, iY, &g_TEXT_ImageChar );
    }
    // return the char width
    return iWidth;
}
```

Listing 161 – CMG_TEXT_Render.c, Teil 2

Die Funktion prüft, ob das Zeichen in der Schriftart definiert ist und setzt das Offset auf das Zeichen. Anschließend wird die Breite eingelesen und für den Rückgabewert zwischengespeichert. Danach bereitet die Funktion das Image für das neue Zeichen vor, indem sie die Zielbreite setzt und die Schriftdaten direkt aus den Rohdaten kopiert.

Hier müßte die Behandlung der Text- und Texthintergrundfarbe sowie die der zu verwendenden Farbtiefe stattfinden. Die hier benutzte Variante funktioniert nur bei 1bpp mit schwarzer Schrift vor weißem Hintergrund.

Zuletzt muß das soeben gefüllte Image noch an der richtigen Position gezeichnet werden.

10.6. WINDOWS FONT-CONVERTER

Um ohne großen Aufwand vorhandene Schriftarten in CMG-Schriftarten umwandeln zu können, benötigt man einen Konverter. Für Windows-Schriftarten beinhaltet CMG den in diesem Kapitel vorgestellten Konverter.

Das Programm *CMGFontConvert* ist in C# für das .NET 2.0 Framework geschrieben und als Entwicklungsumgebung wurde Visual Studio 2005 verwendet.

CMGFontConvert ist primär für Windows entwickelt worden, sollte jedoch mit *mono*[34] auch auf Linux, Solaris, Unix oder dem Mac laufen [MONO]. *mono* ist eine .NET-kompatible Entwicklungs- und Laufzeitumgebung für plattformunabhängige Software, basierend auf dem Common Language Infrastructure-Standard (CLI)[35].

In *CMGFontConvert* kann der Benutzer eine installierte Schriftart – samt Größe und Stil – auswählen und in eine CMG-Schriftart konvertieren lassen. Als Ergebnis steht eine C-Quelldatei zur Verfügung.

Die genaue Beschreibung dieses Tools würde den Rahmen der Arbeit sprengen. Hier sollen nur die Grundlagen behandelt werden. Bei genauerem Interesse verweise ich Sie auf den Quelltext des Projekts.

10.6.1. AUFBAU UND FUNKTIONSWEISE

Was eignet sich zum Einstieg in ein neues Programm besser, als ein paar Bilder? Ich möchte Ihnen im folgenden Abschnitt kurz das Programm mit seinen Möglichkeiten vorstellen.

Nach dem Programmstart können Sie aus den installierten Schriftarten auswählen sowie die Größe und den zu verwendenden Stil festlegen. Im den Statistik-Feldern werden Ihnen dabei die Daten der Schriftart und die benötigte Speichergröße mitgeteilt:

---

[34] Mono Project, http://www.mono-project.com/Main_Page

[35] Beschreibung aus Wikipedia, Mono-Projekt, 02.09.2007, http://de.wikipedia.org/w/index.php?title=Mono-Projekt&oldid=36234399

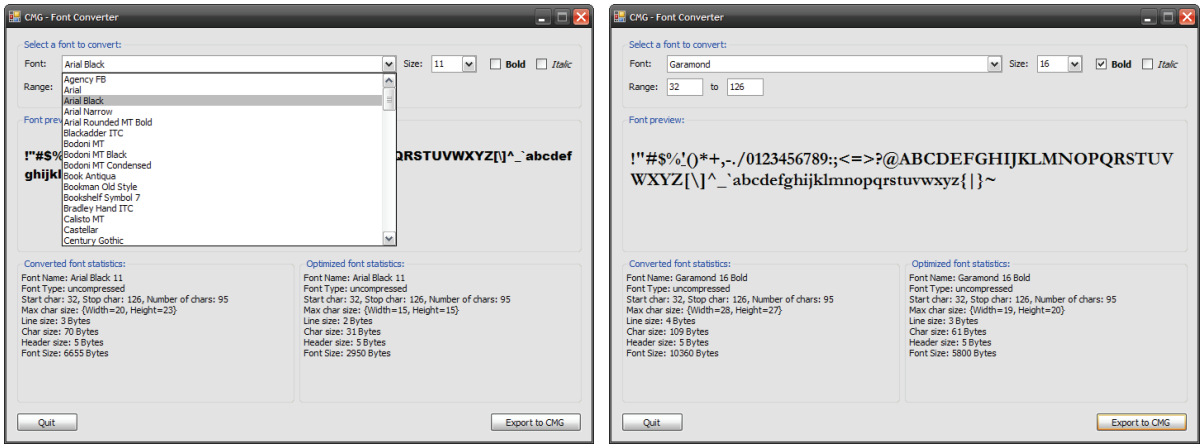


Abbildung 83 – CMGFontConvert, Screenshot 1

Nachdem Sie die passende Schrift ausgewählt und auf „Export to CMG“ geklickt haben, erstellt das Programm die CMG-Schriftart und zeigt das Ergebnis direkt als C-Quellcode an:

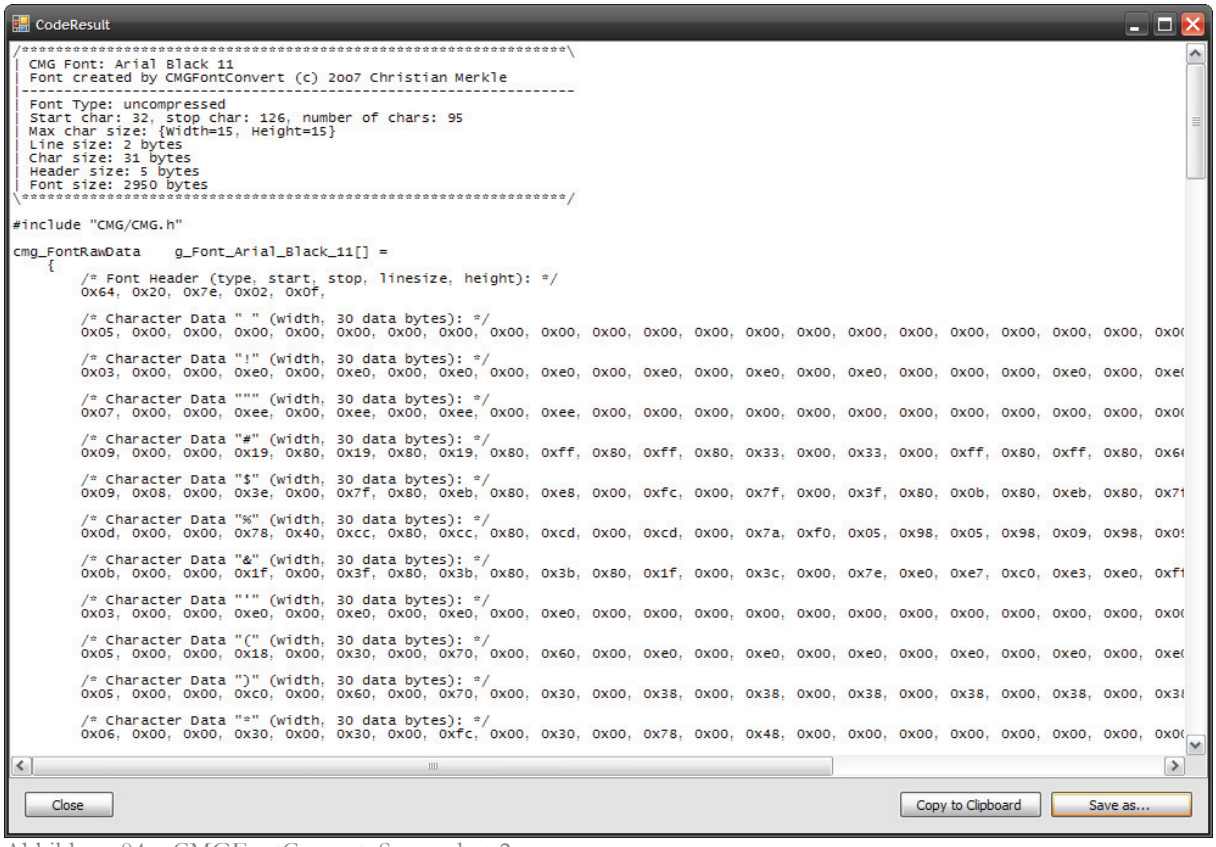


Abbildung 84 – CMGFontConvert, Screenshot 2

Sie können nun wahlweise das Ergebnis in der Zwischenablage oder in einer Zieldatei speichern.

10.6.2. DETAILS

Unter der Haube ist das Programm nach folgendem Klassendiagramm aufgebaut:

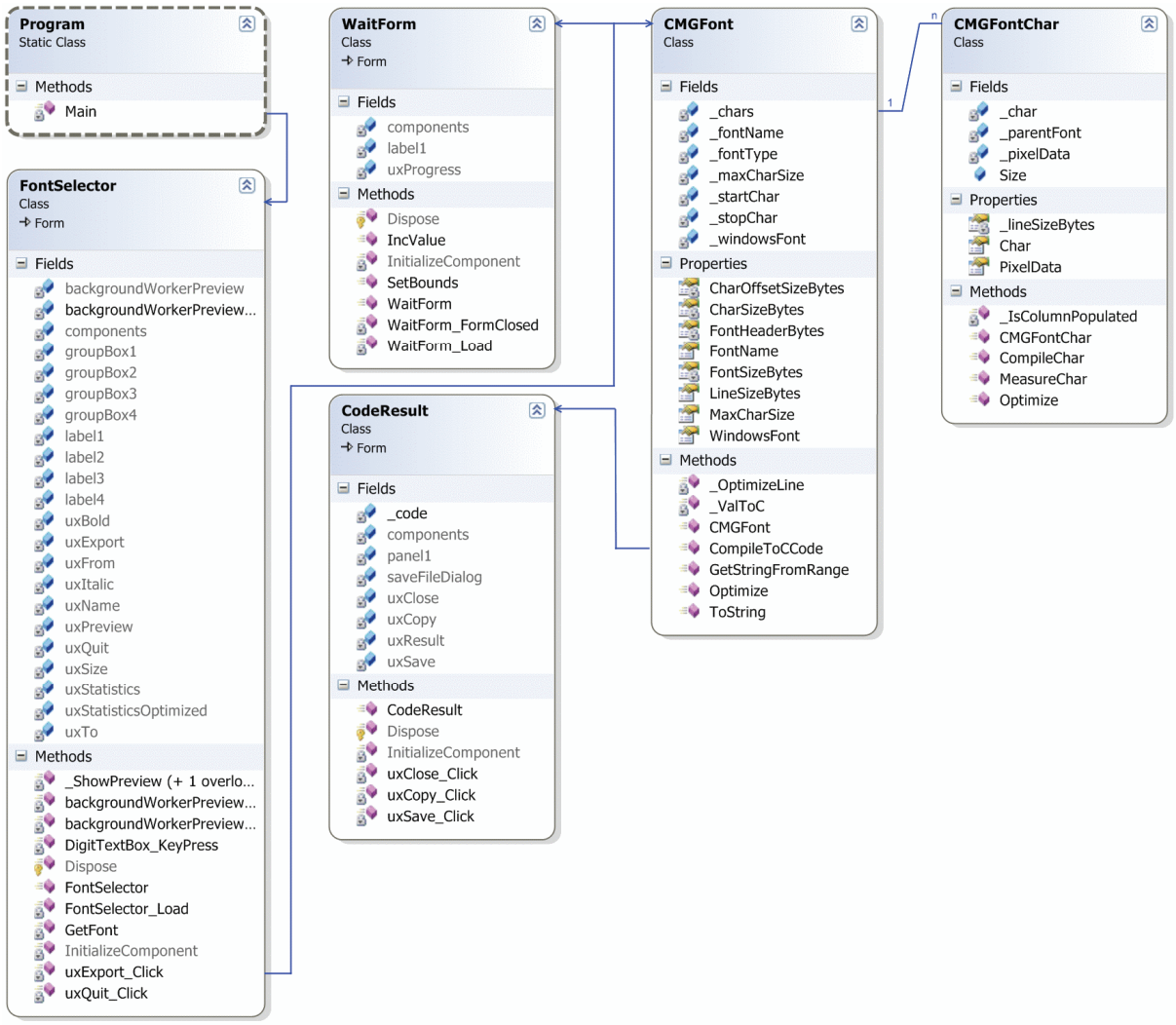


Abbildung 85 – CMGFontConvert, Klassendiagramm

Die Klasse `Program` dient nur dem Programmeinstieg und zeigt gleich die Form `FontSelector` an.

Die Klassen `FontSelector`, `WaitForm` und `CodeResult` sind von `Form` abgeleitet – also GUI-Fenster – und behandeln die Benutzereingaben und die Ergebnisdarstellung.

Die eigentliche Arbeit erledigen die Klassen `CMGFont` und `CMGFontChar`.

`CMGFont` stellt die komplette CMG-Schriftart dar und `CMGFontChar` ein einzelnes Zeichen daraus.

Windows ermittelt beim Ausgeben von Text nicht die genauen Abmessungen, sondern läßt nach allen Seiten hin etwas Abstand. Die beiden `Optimize`-Methoden verkleinern die abzuspeichernde Größe auf ein Minimum. In der `Char`-Klasse wird die Breite eines jeden einzelnen Zeichens optimiert, damit das erste Pixel in der ersten Spalte erscheint und die Gesamtbreite wirklich nur die benötigten Pixel beträgt. Die `Font`-Klasse reduziert die Höhe der Schriftart auf ein Minimum. Da die Höhe jedoch für alle Zeichen gleich ist, müssen auch alle Zeichen gleichzeitig in die Berechnung einfließen.

Zuletzt besitzen auch beide Klassen eine `Compile`-Methode: In der `Char`-Klasse liefert diese die für das Zeichen notwendigen Daten als Bytestrom zurück. Die `Compile`-Methode in der `Font`-Klasse übergibt als Ausgabe direkt den fertigen C-Quellcode, in welcher die einzelnen Zeichen eingebunden worden sind.

10.7. Beispiele

Das erste Beispiel zeigt die Verwendung mehrerer Schriftarten; insgesamt werden fünf verschiedenartige Schriftarten geladen:

```
    cmg_Font Font_Elephant_14;
    cmg_Font Font_Garamond_13;
    cmg_Font Font_Tahoma_7;
    cmg_Font Font_Rage_Italic_14;
    cmg_Font Font_Impact_14;

    cmg_Coord y = 1;

    CMG_TEXT_Font_Create( &Font_Elephant_14, g_Font_Elephant_14, 1 );
    CMG_TEXT_Font_Create( &Font_Garamond_13, g_Font_Garamond_13, 1 );
    CMG_TEXT_Font_Create( &Font_Tahoma_7, g_Font_Tahoma_7, 1 );
    CMG_TEXT_Font_Create( &Font_Rage_Italic_14, g_Font_Rage_Italic_14, 0 );
    CMG_TEXT_Font_Create( &Font_Impact_14, g_Font_Impact_14, 2 );
```

Listing 162 – TEXT, Beispiel 1, Teil 1

Beachten Sie die verschiedenen Spacing-Werte für die einzelnen Schriftarten. Je nach Schrifttyp und Schriftgröße sind andere Werte sinnvoll: Eine Blockschrift braucht einen größeren Abstand, kursive Schreibschrift eventuell gar keinen.

Nach diesem Codeblock können die Schriften nun verwendet werden. In der Variablen `y` steht die aktuelle Pixelzeile für die Textausgabe, welche nach jeder Ausgabe um die Texthöhe verschoben wird:

```
    CMG_TEXT_Font_Set( &Font_Elephant_14 );
    CMG_TEXT_Render( 2, y, "Hello World!" );
    y += CMG_TEXT_Measure_GetTextHeight();

    CMG_TEXT_Font_Set( &Font_Garamond_13 );
    CMG_TEXT_Render( 2, y, "That's the Garamond font!" );
    y += CMG_TEXT_Measure_GetTextHeight();

    CMG_TEXT_Font_Set( &Font_Tahoma_7 );
    CMG_TEXT_Render( 2, y, "Hello from the small Tahoma font." );
    y += CMG_TEXT_Measure_GetTextHeight();

    CMG_TEXT_Font_Set( &Font_Rage_Italic_14 );
    CMG_TEXT_Render( 2, y, "Sample of cursive handwriting." );
    y += CMG_TEXT_Measure_GetTextHeight();

    CMG_TEXT_Font_Set( &Font_Impact_14 );
    y += 10;
    CMG_TEXT_Render( 60, y, "~~~ CMG ~~~" );
    y += CMG_TEXT_Measure_GetTextHeight();
    CMG_TEXT_Render( 5, y, "Christian Merkle Graphics" );
    y += CMG_TEXT_Measure_GetTextHeight();
```

Listing 163 – TEXT, Beispiel 1, Teil 2

Das Ergebnis dieser Zeilen zeigt das Bild auf der nächsten Seite:

Abbildung 86 – TEXT, Beispiel 1

10.8. Optimierungsmöglichkeiten

Aufgrund der Zeitbeschränkung der Masterarbeit konnte ich noch nicht alle – von mir geplanten und während der Implementierung aufgetretenen – Aspekte der Schriftunterstützung vollständig verwirklichen. Die folgenden Abschnitte enthalten einige Ideen zur Erweiterung und Verbesserung der TEXT-Schicht:

10.8.1. Schriftarten

10.8.1.1. Speicherung und Kompression

Ein großes Problem der Schriften ist der hohe Speicherverbrauch. Im obigen Beispiel benötigt die kleinste Schriftart Tahoma schon 2kB, die größte sogar 6kB.

Auch wenn sich die Daten nur im Flash-Speicher befinden, ist das gerade bei kleinen Embedded Systems unnötig viel Speicherplatz, da die überwiegende Anzahl der abgespeicherten Werte überflüssig ist. Um diesen Umstand zu verstehen, müssen wir uns noch einmal vor Augen führen, wie die Schriftart abgespeichert wird:

Beim Optimiervorgang von CMGFontConvert werden die kleinstmögliche Höhe aller Schriftzeichen sowie die kleinste Breite eines jeden Zeichens berechnet. Die Höhe ist für alle Zeichen gleich, die Breite kann jedoch unterschiedlich sein. Allerdings benötigen nur sehr wenige Zeichen die volle Höhe – die meisten sogar nur einen Bruchteil davon. Und auch die Berechnung der minimalen Breite des Zeichens hilft dem Speicherverbrauch nichts, denn intern müssen im jetzigen Dateiaufbau alle Zeichen gleich breit sein – nämlich ein Vielfaches von acht Pixeln. Benötigt auch nur ein einziges Zeichen im Zeichensatz rechts einen Pixel mehr – nehmen wir 9 statt 8 Pixel Breite an – so müssen auch für alle anderen Zeichen pro Zeile ein Byte mehr gespeichert werden.

Eine Lösung des Problems ist, sich beim Einlesen eines jeden Zeichens von den Bytegrenzen zu lösen und dafür einen Bitstrom einzuführen: Für jedes Zeichen wird nicht – wie bisher – nur die Breite gespeichert, sondern auch welcher Bereich des Zeichens in dem Bitstrom abgelegt ist. Dafür sind vier Werte notwendig: Die X- und Y-Koordinate des Startpunkts sowie die Breite und Höhe des gespeicherten Zeichens. Auch diese Werte sollten nicht als Bytes gespeichert werden, da sie meistens – vor allem bei kleinen Schriftarten – nur einen Teil des Bytes belegen. Für einen Großteil der Schriftarten sollten 4-5 Bits ausreichen. Dieser Wert der Bitbreite könnte zusätzlich im Font-Header gespeichert werden.

Anschließend an die Beschreibung des Zeichens kommen die eigentlichen Zeichendaten – auch direkt als Bitstrom – ohne Berücksichtigung etwaiger Bytegrenzen.

Somit sollte sich der benötigte Speicherbedarf auf die Hälfte bis hin zu etwa einem Drittel verringern lassen. Und 700 Bytes bis 1kB für eine kleine Schriftart sind auch für speicherschwache Mikrocontroller zu verkraften.

Einen großen Nachteil hat diese Speicherart jedoch: Bis jetzt ist es einfach möglich, das Startoffset eines jeden einzelnen Zeichens schnell und ohne großen Aufwand zu berechen:

```
_TEXT_FontRawAccess_SetOffset( ( cChar - g_TEXT_pFont->byCharStart ) * g_TEXT_pFont->uCharOffsetBytes );
```

Listing 164 – TEXT, Berechnung des Zeichenoffsets

Man muß nur den Zeichenindex mit der Zeichengröße multiplizieren – die Zeichengröße ist ja für alle Zeichen konstant. Wird dieser Wert aber – wie im obigen Ansatz – variabel gemacht, so kann das Offset nicht mehr direkt berechnet werden.

Eine Lösung dafür ist die Einführung einer zusätzlichen Tabelle, welche entweder bereits in den Schriftdaten vorhanden ist oder erst beim Laden erstellt wird. Diese enthält für jedes Zeichen einen Pointer oder ein Offset auf den eigentlichen Zeichenanfang. Aber auch wenn man nur 128 Zeichen pro Zeichensatz annimmt, wäre diese Tabelle für einen Prozessor mit 16-Bit Speicherbus schon 256 Bytes, für einen 32-Bit Speicherbus sogar schon 512 Bytes groß. Diese Größe ist im RAM-Speicher bei kleinen Controllern undenkbar. Selbst wenn diese Tabelle in den Flash-Speicher ausgelagert werden würde, benötigt sie jedoch trotzdem schon einen großen Teil der Schriftdaten (z.B. 800 Bytes Schriftdaten + 512 Bytes Tabelle).

Die Lösung wird deshalb wohl irgendwo in der Mitte liegen: Es existiert eine verkleinerte Tabelle mit den Offsets im Flash-Speicher. Diese Tabelle beinhaltet nur die Offsets jedes z.B. achten Zeichens. Auch wird nicht die volle Speicherbreite ausgenutzt, sondern nur die tatsächlich benötigte. Dafür wird wieder ein Bitstrom eingesetzt; bei einer Schriftart von kleiner als 2kB also nur 11 Bits pro Eintrag. Somit reduziert sich die Größe – mit den Werten des obigen Beispiels – auf nur noch 22 Bytes. Das eigentliche Startoffset findet man, indem man zum nächstliegenden Zeichen springt und dann die restlichen – noch nötigen – Schritte manuell durchsucht. Hier müßte man durch Versuche die beste Kombination aus Performance und Speicherplatzverbrauch finden.

10.8.1.2. VIRTUELLER CURSOR

Um einen virtuellen Cursor einzuführen, sollten noch – zusätzlich zu der Breite – zwei vorzeichenbehaftete Werte je Zeichen mit angegeben werden. Somit kann man die Platzierung der Zeichen von ihrer eigentlichen Größe trennen. Wenn man sich ein großes und breites „T“ vorstellt, so könnte das nächste, kleine Zeichen schon vor dem rechten Ende des T's anfangen – nämlich schon nach dem Ende des vertikalen, mittleren Strichs. Jedes Zeichen gibt also im ersten Wert an, um wie viele Pixel der vertikale Wert des Einfügepunkts des Zeichens verschoben

werden soll. Der zweite Wert sagt aus, wie weit der virtuelle Cursor nach dem Zeichnen verschoben werden soll, also ab wann ein neues Zeichen eingefügt werden kann – unabhängig davon, wie breit das Zeichen insgesamt wirklich ist. Somit könnte man das Schriftbild – vor allem bei kursiven Schriftarten – optimieren.

10.8.2. TEXTAUSGABE

10.8.2.1. TEXT- UND HINTERGRUNDFARBE

Bis jetzt werden die zu zeichnenden Daten im Image 1:1 von den Quell-Schriftdaten übernommen. Das TEXT-Interface beinhaltet schon Funktionen zum Setzen der Text- sowie der Hintergrundsfarbe. Somit wäre auch weiße Schrift auf schwarzem Grund möglich bzw. in anderen Farbmodi Graustufen oder richtige Farben. Um dies möglichst einfach zu implementieren, sollte man bis nach der Erweiterung aus 10.8.1.1 – der Einführung eines Bitstroms – warten. Danach muß beim Laden eines Zeichens – bei der Erstellung des Images – jedes eingelesene Bit in die Text- bzw. die Hintergrundfarbe konvertiert werden. Somit wäre auch gleich die Unterstützung aller Farbtiefen gesichert.

10.8.2.2. BLOCK-AUSGABE

Mit der aktuellen TEXT-Schicht kann nur ein Startpunkt für die Textausgabe angegeben werden. Zusätzlich wäre eine Ausgabe sinnvoll, die ein Rechteck mit überliefert bekommt, in welche sie den Text anpaßt und einen automatischen Wörterumbruch vornimmt.

Dies sollte nicht schwer zu implementieren sein: Die benötigte Breite eines Textes kann leicht bestimmt werden. Die Funktion sucht nun die maximale Anzahl an Wörtern, welche gerade noch in das Rechteck hineinpassen und rendert dann diese Textzeile. Mit dem restlichen Text fängt sie anschließend in der nächsten Zeile wieder von vorne an.

10.8.2.3. SKALIERUNG

Wenn gleiche Schriftarten in verschiedenen Größen benutzt werden sollen, könnte die Ausgabe um einen Skalierungsfaktor erweitert werden. Somit ließen sich die Schriftzeichen stufenlos sowohl vergrößern als auch verkleinern oder aber auch Spezialeffekte wie eine Breitschrift oder eine sehr schmale, hohe Schrift erzielen. Allerdings wird diese Technik, vor allem bei kleinen und dünnen Schriftarten, nicht sehr ansprechend aussehen, da systembedingt entweder einzelne Zeilen fehlen oder doppelt erscheinen. Mögliche Lösungen hierfür wären Antialaising oder sogar die Unterstützung von Vektorschriftarten. Das sind allerdings sehr aufwendige, prozessor- und speicherhungrige Techniken, welche in einem so kompakten Framework wie CMG vermutlich fehl am Platze sind.

11. GUI – User Interface

Die GUI-Schicht ist noch nicht implementiert. Dieses Kapitel soll allerdings ein paar Ideen dazu geben.

Die wichtigste Eigenschaft der Schicht muß jedoch sein, daß bei minimaler Codegröße sowie minimalem Speicherverbrauch die damit maximal mögliche Funktionalität und Flexibilität erreicht wird.

11.1. Aufbau

Der grundlegende Aufbau sieht wie in folgender Abbildung aus:

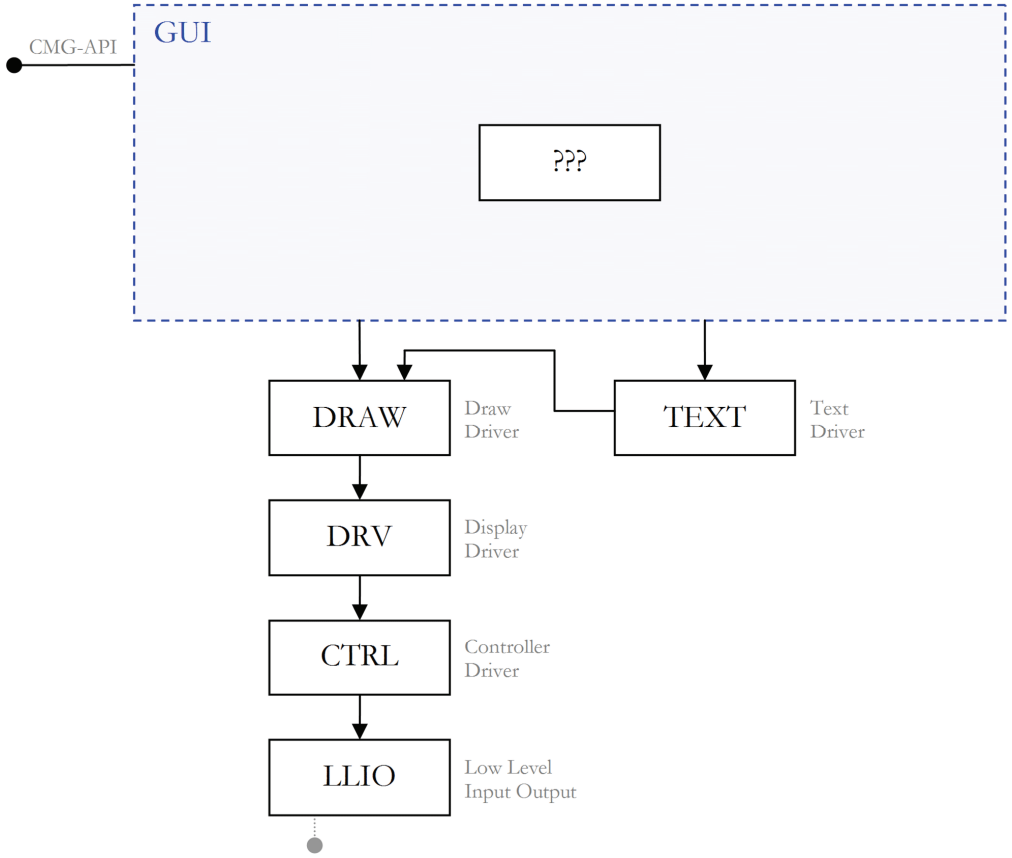


Abbildung 87 – GUI Aufbau

Die GUI-Schicht greift somit auf die DRAW- und die TEXT-Schicht zu. Die ursprünglichen Interfaces der beiden unteren DRAW- und TEXT-Schichten dürfen nicht mehr öffentlich sein – der User darf also nicht direkt darauf zugreifen. Vielmehr stellt die GUI-Schicht eine Abstraktion dar und implementiert zusätzliche Funktionalität.

Der Benutzer darf nun nicht mehr auf den ganzen Bildschirm schreiben, sondern arbeitet mit sogenannten Fenstern – mit ihren jeweils eigenen, von den anderen unabhängigen

Koordinatensystemen. Jedes Fenster ist frei auf dem zugehörigen Elternfenster plazierbar. Jedes Elternfenster kann beliebig viele Unterfenster besitzen. Der komplette Bildschirm ist das sogenannte Root- oder Desktopfenster. Im Prinzip sind auch die großen Frameworks nach dem gleichen Muster – wie unter anderen GTK+, Gnome oder GDI – aufgebaut.

Bei der Ausgabe in einem Fenster ist diese auf die Abmessungen des Fensters mit Hilfe des Clipping-Fensters beschränkt, ein Fenster kann also nicht außerhalb sich selbst zeichnen.

Die Oberfläche soll komplett nachrichtenbasiert aufgebaut sein. Um den einzelnen Fenstern Nachrichten zukommen zu lassen, besitzt jedes Fenster eine Callback-Funktion, welche beim Eintreffen einer Nachricht aufgerufen wird. Diese Funktion bekommt Parameter mit überliefert, welche den Nachrichtentyp und die dafür notwendigen Daten beinhalten.

Die Nachrichten können sowohl vom System kommende Anfragen – wie das Neuzeichnen des Fensters – sein, als auch die Benachrichtigung von Benutzereingaben – wie Tastendrücke und eventuelle Maus- oder Joystickaktionen, je nachdem welche Eingabegeräte vorhanden sind.

Um die Verwendung einfacher zu gestalten, existieren einige vordefinierte Fenstertypen – die grundlegenden Steuerelemente. Beispiele hierfür wären: Dialogfenster, Schaltflächen, Texteingabefelder, Auswahlknöpfe, aber auch Fenster zur Layouthilfe.

Beim Erstellen eines neuen Fensters wird angegeben, ob man sich selbst um die Behandlung der Nachrichten kümmern möchte oder ob dieses Fenster ein vordefinierter Typ ist – der bei Bedarf auch erweitert werden kann. Vordefinierte Fenster stellen nach außen hin eine Datenstruktur zur Verfügung, die alle wichtigen Eigenschaften beinhaltet. Bei Texteingabefelder, zum Beispiel den eingegebenen, dargestellten Text, bei Auswahlelementen, den aktuellen Zustand usw.

11.2. Beispiele

Mit den Zeichenmöglichkeiten von CMG lassen sich schon jetzt ansehnliche Oberflächen erstellen. In diesem Abschnitt werden alle Teile von CMG miteinander kombiniert und damit ein Dialog mit einigen Steuerelementen gezeichnet.

Der Dialog an sich wird – nachdem die GUI-Schicht noch nicht implementiert ist – keine Funktionalität besitzen. Er soll jedoch veranschaulichen, wie einfach das Erstellen und Zeichnen von Oberflächen mit CMG ist.

Zu Beginn wird der Desktophintergrund gezeichnet und anschließend die Funktion zum Zeichnen des Dialogs aufgerufen:

```
    // desktop background
    CMG_SetROPMode( ROPMODE_COPY );
    CMG_SetDrawStyle( DRAWSTYLE_FILL );
    CMG_SetBrush( &BrushLight );
    CMG_Rectangle( 0, 0, iScreenSizeX - 1, iScreenSizeY - 1 );

    Dialog( 10, 7, 225, 100, "CMG - Christian Merkle Graphics" );
```

Listing 165 – GUI, Beispiel, Teil 1

Zuerst einmal möchte ich den fertigen Dialog präsentieren:

Abbildung 88 – GUI, Beispiel 1

Der dafür notwendige Quelltext ist nicht sehr umfangreich: Der nächste Codeabschnitt zeigt das Zeichnen des Dialogs und besteht aus zwei Teilen. Im ersten wird der Dialog an sich gezeichnet und im zweiten die Steuerelemente darauf:

```
void Dialog( cmg_Coord x1, cmg_Coord y1, cmg_Coord x2, cmg_Coord y2, cmg_char *strTitle )
{
    CMG_SetROPMode( ROPMODE_COPY );

    // dialog window
    CMG_SetDrawStyle( DRAWSTYLE_BOTH );
    CMG_SetColor( 1 );
    CMG_SetFillColor( 0 );
    CMG_Rectangle( x1, y1 + 7, x2, y2 );

    // titlebar
    CMG_SetDrawStyle( DRAWSTYLE_FILL );
    CMG_SetFillColor( 1 );
    CMG_RoundedRectangle( x1, y1, x2, y1 + 14, 6 );

    // title text
    CMG_SetROPMode( ROPMODE_XOR );
    CMG_TEXT_Render( x1 + 10, y1 + 3, strTitle );
    CMG_SetROPMode( ROPMODE_COPY );

    // close Button
    CMG_SetFillColor( 0 );
    CMG_Circle( x2 - 7, y1 + 7, 5 );
    CMG_Line( x2 - 9, y1 + 5, x2 - 5, y1 + 9 );
    CMG_Line( x2 - 5, y1 + 5, x2 - 9, y1 + 9 );


    // text
    CMG_TEXT_Render( x1 + 10, y1 + 25, "Please enter some text:" );

    // textbox
    TextBox( x1 + 10, y1 + 40, x2 - x1 - 20, "What do you think about this one?" );

    // buttons
    Button( x1 + 10, y2 - 25, 60, "Cool" );
    Button( x2 - 90, y2 - 25, 80, "Very Cool!" );

}
```

Listing 166 – GUI, Beispiel, Teil 2

Eine Textbox wird so gezeichnet:

```
void TextBox( cmg_Coord x1, cmg_Coord y1, cmg_Coord iWidth, cmg_char *strText )
{
    cmg_Coord   iCusorPos;

    CMG_SetROPMode( ROPMODE_COPY );

    // element
    CMG_SetDrawStyle( DRAWSTYLE_BOTH );
    CMG_SetColor( 1 );
    CMG_SetFillColor( 0 );
    CMG_Rectangle( x1, y1, x1 + iWidth, y1 + 15 );

    // text
    CMG_TEXT_Render( x1 + 3, y1 + 4, strText );
```

```
    // cursor
    iCusorPos = CMG_TEXT_Measure_GetTextWidth( strText ) + 3;
    CMG_Line( x1 + iCusorPos, y1 + 3, x1 + iCusorPos, y1 + 12 );
}
```

Listing 167 – GUI, Beispiel, Teil 3

Der letzte Quelltext zeichnet eine Schaltfläche:

```
void Button( cmg_Coord x1, cmg_Coord y1, cmg_Coord iWidth, cmg_char *strText )
{
    cmg_Coord   iTextOffset;

    CMG_SetROPMode( ROPMODE_COPY );

    // element
    CMG_SetDrawStyle( DRAWSTYLE_BOTH );
    CMG_SetColor( 1 );
    CMG_SetFillColor( 0 );
    CMG_RoundedRectangle( x1, y1, x1 + iWidth, y1 + 14, 5 );

    // text
    iTextOffset = ( iWidth - CMG_TEXT_Measure_GetTextWidth( strText ) ) >> 1;
    CMG_TEXT_Render( x1 + iTextOffset, y1 + 3, strText );
}
```

Listing 168 – GUI, Beispiel, Teil 4

Erweitert man die eine Codezeile nach dem Zeichnen des Desktophintergrunds und zeichnet zwei Dialoge übereinander, läßt sich die Verwendung von mehreren Fenstern auch schon erahnen:

```
    Dialog( 5, 6, 220, 100, "CMG - Christian Merkle Graphics" );
    Dialog( 40, 31, 252, 125, "Hello from my second dialog" );
```

Listing 169 – GUI, Beispiel, Teil 5

Die in diesem Beispiel verwendete Displayauflösung ist nur 256 x 128:

Abbildung 89 – GUI, Beispiel 2

Somit ist auch eine grafische Oberfläche mit sehr geringen Auflösungen möglich – wobei bei Auflösungen wie 320 x 240 natürlich ganz andere Möglichkeiten zu Verfügung stehen.

12. Zusammenfassung

12.1. Fazit

Zuerst einmal Erleichterung: Alle von mir zu Beginn gesetzten Ziele sind erreicht und erfolgreich umgesetzt. Zur Erinnerung noch einmal der Abschnitt zu den Zielen aus der Einleitung:

Die Hauptziele von CMG sind:

- Einfache API und damit Ansteuerung
- Unabhängigkeit vom Display-Controller
- Unabhängigkeit von der Plattform
- Einfache Erweiterbarkeit um neue Controller und Plattformen
- Emulation und Entwicklung der Programme am PC ohne direkte Hardwareanbindung

Zusätzlich soll CMG optimiert sein für:

- Codegröße
- Ausgabegeschwindigkeit
- Modularität

Die Arbeit beinhaltete viele interessante Aspekte: Von der Recherche der vorhandenen Lösungen über die Planung des Projekts sowie den Aufbau der einzelnen Schichten, bis hin zur Implementierung und der Dokumentation. Es war anspornend und schön zu sehen, wenn eine weitere Schicht fertiggestellt wurde und diese mit den anderen wie geplant funktionierte.

Wie vermutlich bei jedem größeren Projekt gab es Hoch- und Tiefpunkte: Manche zu Beginn sehr schwer vermuteten Teile stellten sich bei der Implementierung doch leichter als erwartet heraus – andere, einfach geglaubte oder gar nicht beachtete Komponenten brachten dann doch teilweise hohe Hürden mit sich.

Zusammenfassend läßt sich jedoch festhalten, daß CMG dem Benutzer – auch schon auf dem jetzigen Stand – einen großen Teil der Arbeit bei der Grafikausgabe für eigene Projekte abnimmt und es sich nicht nur für kleine, leistungsschwache Mikrocontroller mit geringem Speicher hervorragend eignet.

12.2. Mögliche Verbesserungen und vorhandene Limitierungen

Ein solches Projekt wie CMG wird nie komplett fertiggestellt sein. Es wird immer Punkte geben, die verbessert oder erweitert werden können. Dieses Kapitel soll einige, während der Arbeit aufgetauchten Ideen zeigen und richtet sich vom Aufbau her nach den einzelnen Schichten.

Um einen möglichst großen Erfolg zu bekommen, sollten – zusätzlich zu den folgenden, konkreten Vorschlägen – die einzelnen Schichten an sich um weitere Komponenten erweitert werden. Je mehr Controller- und Displaytypen sowie Farbmodi unterstützt werden, um so besser.

12.2.1. Generelle Verbesserungen

- Bis jetzt sind die in `CMT_Types.h` definierten Minimum-Typen wie `cmg_mu8`, `cmg_ms16`, usw. noch nicht auf die jeweilige Hardware optimiert, sondern nur mit der minimalen Größe definiert. Die Auswahl sollte nach verwendeter Plattform automatisch in `#if`-Blöcken erfolgen.

- Es sollten Vorlagen zu jeder Schicht erstellt werden, damit sich Erweiterungen noch einfacher einpflegen lassen.

- Ein komfortables Programm zur Erstellung und Verwaltung der Config-Datei wäre wünschenswert. Dieses kopiert die einzelnen Config-Fragmente aus den vorhandenen Schichten zusammen und hilft beim Einstellen der Optionen.

- Es könnte im Code eine automatische Auswahl der Zielplattform – je nach verwendetem Compiler – erfolgen. Durch geschickte `#if`-Blöcke müssen die `CMG_PLATFORM_*`-Werte korrekt gesetzt werden.

12.2.2. LLIO / Emulatoren

- Bis jetzt müssen die `/RD`- und `/WR`-Wartezeiten in Taktzyklen angegeben werden. Es wäre jedoch komfortabler, diese in Zeiteinheiten – wie Nanosekunden – angeben zu können. Dazu müssen jedoch die Taktrate sowie eventuell andere mikrocontrollerspezifische Details beachtet werden, um anschließend wieder die korrekte Anzahl an Taktzyklen errechnen zu können.

- Die GKT+-Ausgabe der Emulatorschicht müßte – wie in GDI – um ein Fehlerausgabefenster erweitert werden, welches noch nicht unterstützte Controller-Befehle anzeigt. Zur Zeit sollten keine Fehler auftauchen, da alle benötigten Befehle implementiert sind; falls aber andere Controller-Emulatoren implementiert werden, ist dies nicht mehr gewährleistet.

- Die Emulator-Schicht sollte um eine Farbausgabe erweitert werden. Bis jetzt lassen sich nur Graustufen darstellen.

12.2.3. CTRL

- Die vorhandenen Komponenten dieser Schicht bedürfen eigentlich keiner Nachbesserung. Jedoch sollte die Schicht um weitere Treiber für andere Controller erweitert werden.

- Es gilt noch ein anderes, schwieriges Problem zu lösen: Während meiner Arbeit sind mir zwei Controllertypen untergekommen, welche intern einen komplett anderen Speicheraufbau besitzen, als alle anderen mir bekannten und verwendeten. Nahezu alle Controller ordnen die einzelnen Bits zeilenweise fortlaufend von links nach rechts an. Die acht Bits eines Bytes liegen also direkt horizontal nebeneinander. Das nachfolgende Byte

schließt sich direkt an das letzte rechte Bit an. Beim SED1565 ist das jedoch nicht der Fall. Die acht Bits eines Bytes liegen vertikal untereinander. Das nächste Byte wird nun aber horizontal, um einen Pixel nach rechts, verschoben. Dieser Modus ist leider absolut inkompatibel zu dem jetzigen, benötigten Speicheraufbau. Es gibt zwei Lösungsmöglichkeiten: Entweder es wird für diese Art Controller zusätzlich eine neue DRV-Schicht mit eben diesem Speichermodell eingeführt oder der Speicherzugriff auf ein Byte durch acht einzelne Zugriffe auf die jeweiligen Positionen ausgeführt und das eigentliche Byte damit zusammengesetzt.

12.2.4. DRV

- Es sollten auch hier weitere Treiber für andere Farbtiefen implementiert werden.

- Bis jetzt müssen sich wiederholende Stifte immer auf einer Bytegrenze enden – also eine Länge mit dem Vielfachen von Acht besitzen. Es wäre jedoch viel universeller die Stiftlänge nicht in Bytes, sondern in einzelnen Pixeln anzugeben.

- Außerdem wäre es bei Stiften sinnvoll, zusätzlich eine Funktion wie `GetReversedBits(…)` zu besitzen, um auch bei Rechtecken – also für optimierte horizontale sowie für optimierte vertikale Linien – einen durchgehenden Rand zeichnen zu können.

- Eine weitere Verbesserung wäre, zusätzlich ein Ausgabeziel festzulegen. Ist das normale Ziel das Display, dann arbeitet alles wie gehabt. Es kann alternativ dazu jedoch auch ein Speicherbereich definiert werden (z.B. ein Image), in welchem gezeichnet wird. Somit könnten – genügend Speicherplatz vorausgesetzt – Ausgaben im Hintergrund vorbereitet und anschließend beliebig ausgegeben werden.

12.2.5. DRAW

- Die grundlegenden Umwandlungen von den logischen zu den physikalischen Koordinaten sind bereits in CMG integriert, lauffähig und werden in jeder Funktion der DRAW-Schicht aufgerufen. Allerdings funktioniert die Ausgabe von Images oder Pinseln bei Rotationen noch nicht korrekt. Entweder müßte man dann auf die Hardwareunterstützung der gleichzeitigen Byteausgabe verzichten und alle Pixel einzeln zeichnen oder die Bilder zuvor im Speicher entsprechend rotieren.

- Die DRAW-Schicht sollte um weitere Zeichenobjekte erweitert werden. Hierzu zählen u.a. Kurven und Bögen, wie Beziers, Splines und Arcs.

12.2.6. TEXT

- Für diese Schicht müssen noch Flash-Laderoutinen für die ARM- und AVR-Plattformen implementiert werden. Bis jetzt kann nur das RAM verwendet werden.

- Alle weiteren Verbesserungs- und Erweiterungsmöglichkeiten zu der TEXT-Schicht finden Sie direkt im Abschnitt 10.8.

12.3. AUFWAND UND STATISTIKEN

Der Quellcode von CMG wurde wegen dem beschränkten Platz im Zielsystem bewußt sehr kompakt gehalten. Es war die Aufgabe, möglichst viel Funktionalität mit minimalen Mitteln umzusetzen. Ein weiterer Punkt ist, daß ich regen Gebrauch von Makros gemacht habe, da ich den Quelltext lesbarer sowie so wenig wie möglichst redundant schreiben wollte. Auch spielten hier Überlegungen zur Performance eine große Rolle.

Aus diesen Gründen halten sich die Quellcodezeilen in Grenzen. Das bekannte Tool *SLOCCount*[36] von David A. Wheeler erstellt beim Analysieren des Projekts folgende Ausgabe:

```
SLOC    Directory       SLOC-by-Language (Sorted)
5380    CMG             ansic=5380
1443    top_dir         ansic=1443
878     CMGFontConvert  cs=878
202     make            asm=162,ansic=40


Totals grouped by language (dominant language first):
ansic:         6863 (86.84%)
cs:             878 (11.11%)
asm:            162 (2.05%)


Total Physical Source Lines of Code (SLOC)                = 7,903
Development Effort Estimate, Person-Years (Person-Months) = 1.75 (21.03)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 0.66 (7.95)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 2.64
Total Estimated Cost to Develop                           = $ 236,768
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

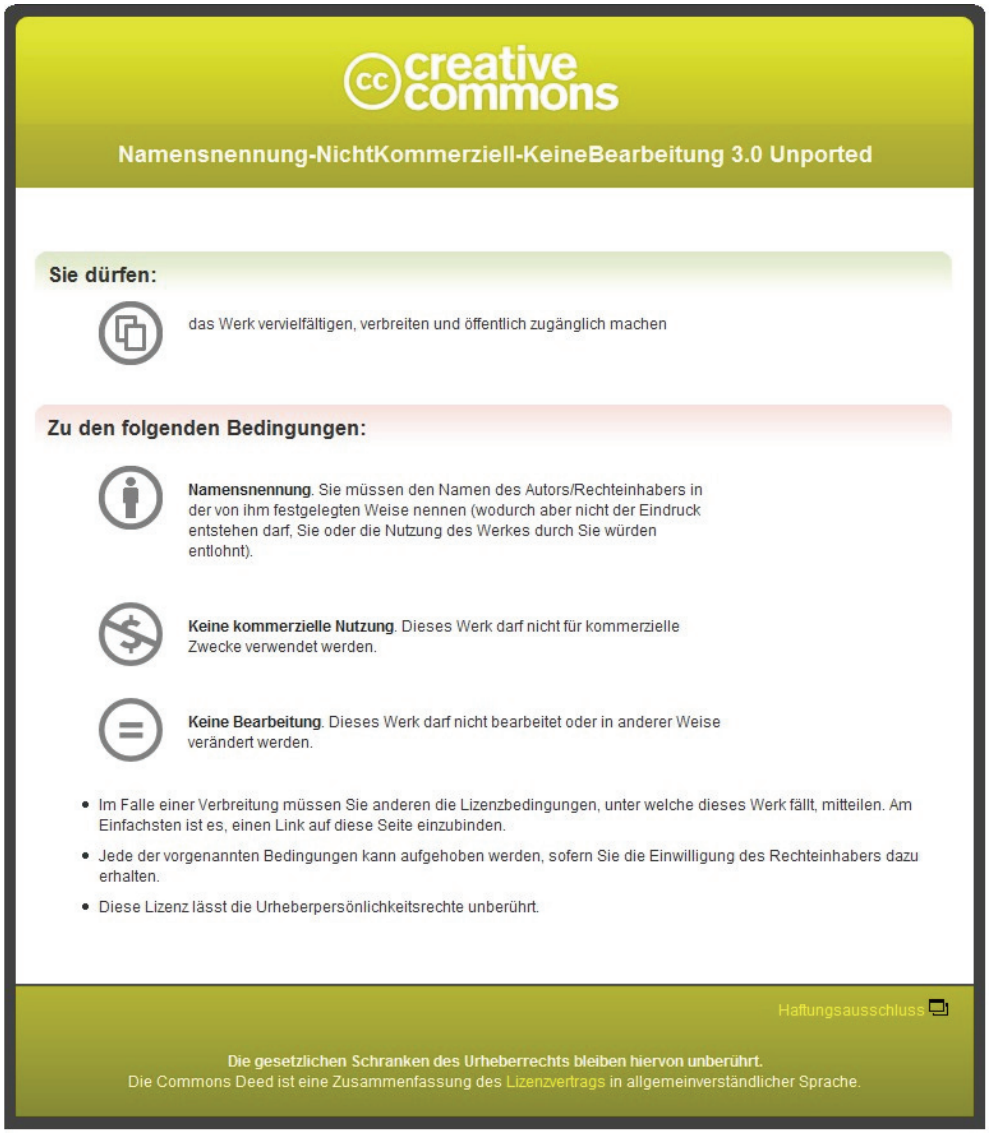
Listing 170 – SLOCCount Ausgabe

Die eigentlichen CMG-Quellcodedateien im Unterordner **CMG** bestehen somit aus etwas über 5300 Zeilen Code. Der Schriftarten-Konverter benötigt 878 Zeilen C#-Code. Die angegeben fast 1500 Zeilen Quelltext im Hauptverzeichnis, setzen sich aus der Config-Datei, den Beispielen und den erstellten Schriftarten zusammen.

Zählt man nicht nur die echten Codezeilen, sondern alle, ergeben sich diese, haupsächlich wegen den Kommentaren erhöhte, Werte: Insgesamt 67 Quellcode-Dateien, 17.750 Textzeilen und 813.000 Zeichen – das ergibt durchschnittlich 45 Zeichen pro Textzeile.

[36] SLOCCount, (a set of tools for counting physical Source Lines of Code), David A. Wheeler, http://www.dwheeler.com/sloccount/

13. Copyright und Lizenz



14. Begleit-CD

Auf der Begleit-CD der Masterarbeit befinden sich folgende Hauptverzeichnisse:

Verzeichnis	Beschreibung
/	Root-Verzeichnis: Masterarbeit im PDF-Format.
/Datasheets	Datenblätter zu den Bereichen: Mikrocontroller, Display-Controller, Displays und elektronische Bauteile.
/DemoReleases	Beispielprogramme und Screenshots.
/Documents	Zusätzliche Unterlagen zum Projekt.
/Photos	Während dieser Arbeit entstandene Fotos zur Hardware.
/Schematics	Erstellte und verwendete Schaltpläne.
/Source	Quellcode-Hauptverzeichnis: Beispielprojekt mit Config-Dateien und Schriftarten.
/Source/CMG	Gesamter CMG-Sourcecode.
/Source/CMGFontConvert	Quelltext des Schriftarten-Konvertierers.
/Source/make	Projekt- und Make-Dateien für die verschiedenen Plattformen.

Tabelle 42 – Begleit-CD

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGVERZEICHNIS

LITERATURVERZEICHNIS

[AM1S] **Datasheet:**
AIMTEC:
AM1S Series
1 Watt DC-DC Converters

[Bres1] **Website:**
Wikipedia (Verschiedene Autoren), vom 13.09.2007:
Bresenham-Algorithmus
http://de.wikipedia.org/w/index.php?title=
Bresenham-Algorithmus&oldid=36684771

[Bres2] **Website:**
Wikipedia (Verschiedene Autoren), vom 14.09.2007:
Bresenham's line algorithm
http://en.wikipedia.org/w/index.php?title=
Bresenham%27s_line_algorithm&oldid=157570485

[DATAVISION] **Datasheet:**
DataVision:
DG12864-12
LC-Display DataVision DG12864-12

[EMBART] **Datasheet:**
Embedded Artists AB:
LPC2104
LPC2104 Color LCD Game with Bluetooth

[EW32] **Datasheet:**
EMERGING DISPLAY TECHNOLOGIES CORPORATION:
EW32FY0

[LH155] **Datasheet:**
SHARP:
LH155BA
128-Segment and 64-Common Outputs LCD Driver IC
with A Built-in RAM

[LM1117] **Datasheet:**
NIKO-SEM:
L1117 Series
1A Fixed and Adjustable Low Dropout Linear Regulator (LDO)

[M078CKA] **Datasheet:**
SHARP:
M078CKA
LC-Display Sharp M078CKA-A3QKLA0057

[POWERTIP] **Datasheet:**
POWERTIP:
PG 240128--A

[S1D13700] **Datasheet:**
EPSON:
S1S13700
LCD Controller ICs Technical Manual

[STK500] **Datasheet:**
Atmel Corporation:
AVR STK500 User Guide
http://www.atmel.com

[T6963C] **Datasheet:**
TOSHIBA
T6963C
Dot Matrix LCD Control LSI

INDEX