Programmierhandbuch für die 2,1” Display-Module mit 65.536 Farben von www.display3000.com

V 1.41
7. März 2008

Inhaltsverzeichnis:

Herzlichen Glückwunsch zum Erwerb unseres Displaymodul-Bausatzes rund um das 2,1“ Display.

Sie werden feststellen, dass Sie sich in Kürze nicht mehr vorstellen könnten, ohne Farbdisplay zu arbeiten. Die hiermit realisierten Projekte bekommen automatisch einen professionellen Touch.

Unsere Software-Library sowie das Programmierhandbuch sowie hat über 70 Seiten und stehen nur unseren Kunden zur Verfügung.
Nachfolgend finden Sie als ersten Eindruck das Inhaltsverzeichnis sowie einige Beispielseiten aus dem Programmierhandbuch.

Über 50 Seiten dieses Manuals zur Programmierung des Displays sowie der Programmierreferenz fehlen hier.

Überblick

Die Verbindung vom Mikrocontroller zum Grafikmodul geschieht seriell, d.h. die Daten werden dem Displaymodul bitweise mitgeteilt.

Leider ist dieses Display ziemlich „dumm“ und bietet uns nicht allzu viele Möglichkeiten der Unterstützung. So müssen wir leider fast alles softwaretechnisch abwickeln. Die komplette Datenaufbereitung eines Bildschirms muss hier von der Software aus geschehen. Egal, ob ein String geschrieben, ein Bild dargestellt oder eine Linie gezogen werden soll: Sämtliche Operationen werden von der Software aus initiiert und jedes einzelne Pixel muss auch von der Software aus gezeichnet werden.

Glücklicherweise kennt das Display wenigstens den Befehl, um den Ausgabebereich auf ein beliebiges Areal zu beschränken (also quasi das Öffnen eines Ausgabefensters) und um die Ausgaberichtung festzulegen. So müssen wir, wenn sich z.B. nur eine dargestellte Ziffer auf dem Bildschirm ändern soll, lediglich den zu ändernden Bereich definieren und diese Daten übertragen. Um z.B. einen 10x10 Pixel großen Kasten zu zeichnen, öffnen wir lediglich ein „Fenster“, welches 10x10 Pixel groß ist und übertragen dann die 10x10=100 Pixel an das Display.
Dies ist der Vorteil eines Displays mit eigenem Display-Memory (hier knapp 50 KByte Speicher) – es verschont uns vor einem ständigen Refresh, was einen Mikrocontroller sicherlich überfordern würde (zumindest hätte dieser dann für nichts anderes mehr Zeit).

Wie schon erwähnt, werden die Daten seriell an das Display übertragen. Um hierfür nicht zu viele Resourcen zu verbrauchen, nutzen wir bei diesem Modul das Hardware-SPI des Mikrocontroller (d.h. das Senden der notwendigen Bits benötigt keine großen Resourcen innerhalb Software). Daher wurde das Display direkt mit dem SPI Port des Mikrocontroller verbunden, wobei hier hauptsächlich die Datenleitung und die Clock-Leitung wichtig sind, die anderen Leitungen DC, CS und Reset werden sowieso manuell angesteuert.

Ihre neuen Befehle für das Display (Bascom Basic)
Folgende Befehle stehen Ihnen nun zur Verfügung:

`Orientation` `= Portrait | Portrait180 | Landscape | Landscape180`
Hiermit legen Sie fest, ob die folgenden Kommandos das Display im Hochformat oder Querformat beschreiben sollen; bzw. jeweils um 180° gedreht.

`Graphics_mode` `= 65k_uncompressed | 65k_compressed |256low_uncompressed | 256low_compressed | 256high_uncompressed | 256high_compressed |`
Legt vorab fest, ob eine darzustellende Bitmap im 65.536 Farben-Modus, im 256-Farben aus 65.536 Farben-Modus oder im 256-aus-4096 Farben-Modus sowie als in komprimierter oder unkomprimierter Datenstrom vorliegt.

`Call LCD_CLS`
Löscht den Bildschirminhalt (faktisch wird hier lediglich eine weiße Box in Bildschirmgröße gezeichnet)

`Call LCD_Print (String, x, y, Font, ScaleX, ScaleY, VFarbe, HFarbe)`
Druckt einen Textstring auf Position x, y mit der angegebenen Schriftart. Dann folgt die Skalierung die beliebig gewählt werden kann (also z.B. „3,2,“: dreifach breit und doppelt hoch) sowie die gewünschten Vordergrund- und Hintergrundfarben

Beispiel: ***Call Lcd_print("Hello World" , 1 , 10 , 2 , 1 , 2 , Dark_red , Yellow)*** schreibt “Hello World” an x-y-Position 1,10 in der Schriftart 2, normal breit und doppelt hoch in dunkelrot auf gelbem Hintergrund.

Numerische Variablen bedürfen einer kleinen „Vorbehandlung“ um mit LCD_Print ausgegeben werden zu können. Hierzu erfahren Sie mehr auf Seite **Fehler! Textmarke nicht definiert.**.

`Call LCD_Plot (x, y, Dicke, Farbe)`
Druckt ein einziges Pixel an die gewünschte x-y Position. Dicke = 0 bedeutet 1 Pixel groß; *Dicke* = 1 bedeutet 2x2 Pixel groß (das zweite Pixel wird jeweils rechts und unten angesetzt – dies gilt es evtl. bei der Angabe der Koordinaten zu berücksichtigen – z.B. müsste ein dicker Außenrahmen nicht an 0,0,131,175 sondern an 0,0,130,174 gezeichnet werden, sonst würde rechts und unten nur eine Linie sichtbar sein – denn das 2. Pixel liegt dann außerhalb des Sichtbereichs).

`Call LCD_Draw (x1, y1, x2, y2, Dicke, Farbe)`
Zeichnet eine Linie von der Koordinate x1,y1 zur Koordinate x2,y2. Die Richtung ist egal. Dieser Algorithmus arbeitet nur mit Integerzahlen und ist sehr schnell. Wg. *Dicke*: siehe LCD_Plot

`Call LCD_Rect (x1, y1, x2, y2, Dicke, Farbe)`
Zeichnet ein Rechteck (nicht gefüllt) von der Koordinate x1,y1 zur Koordinate x2,y2. Wg. *Dicke*: siehe LCD_Plot

`Call LCD_Box (x1, y1, x2, y2, Farbe)`
Zeichnet eine farbige Box. Wichtig: x1,y1 ist immer die obere linke Ecke, x2 und y2 die untere rechte Ecke im jeweiligen .

`Call LCD_Bitmap (x1, y1, x2, y2)`
Öffnet ein Ausgabefenster und füllt es mit genauso viel Bitmapdaten, wie das Fenster Pixel hat. Die Bitmapdaten (z.B. Fotos, Icons etc.) müssen als Data-Zeilen (siehe Beispiele) vorliegen. Daher muss mittels des Befehls **`Restore`** vor Aufruf dieser Funktion der Pointer auf den Beginn der gewünschten Grafikdaten gesetzt werden. Natürlich kann diese Routine auch umgeschrieben werden um anderweitig vorliegende / übergebende Daten darzustellen.
Bei allen Koordinateneingaben gilt: X1 muss kleiner als X2 sein, Y1 kleiner als Y2 (also immer zuerst die linke und obere Position angeben, dann die untere und rechte).

Ihre neuen Befehle für das Display (WinAVR C)
Die folgenden Befehle stehen Ihnen im C-Code zur Verfügung:

`Orientation` `= Portrait | Portrait180 | Landscape | Landscape180`
Hiermit legen Sie fest, ob die folgenden Kommandos das Display im Hochformat oder Querformat beschreiben sollen; bzw. jeweils um 180° gedreht.

`LCD_CLS (Farbe)`
Löscht den gesamten Bildschirminhalt mit einer beliebigen Farbe

`LCD_Print (String, x, y, Font, ScaleX, ScaleY, VFarbe, HFarbe)`
Druckt einen Textstring auf Position x, y mit der angegebenen Schriftart. Dann folgt die Skalierung die beliebig gewählt werden kann (also z.B. „3,2,“: dreifach breit und doppelt hoch) sowie die gewünschten Vordergrund- und Hintergrundfarben

Beispiel: ***Lcd_print("Hello World" , 1 , 10 , 2 , 1 , 2 , Dark_red , Yellow)*** schreibt “Hello World” an x-y-Position 1,10 in der Schriftart 2, normal breit und doppelt hoch in dunkelrot auf gelbem Hintergrund

`LCD_Plot (x, y, Dicke, Farbe)`
Druckt ein einziges Pixel an die gewünschte x-y Position wenn als Dicke eine 0 übergeben wird – bei einer 1 wird ein Punkt von der Größe 2x2 Pixel gesetzt (das zweite Pixel wird jeweils rechts und unten angesetzt – dies gilt es evtl. bei der Angabe der Koordinaten zu berücksichtigen – z.B. müsste ein dicker Außenrahmen nicht an 0,0,131,175 sondern an 0,0,130,174 gezeichnet werden, sonst würde rechts und unten nur eine Linie sichtbar sein – denn das 2. Pixel liegt dann außerhalb des Sichtbereichs).

`LCD_Draw (x1, y1, x2, y2, Dicke, Farbe)`
Zeichnet eine Linie von der Koordinate x1,y1 zur Koordinate x2,y2. Die Richtung ist egal. Dieser Algorithmus arbeitet nur mit Integerzahlen und ist sehr schnell. Wenn als Dicke eine 1 übergeben wird, wird die Linie 2 Pixel breit gezeichnet (bei Dicke=0:1 Pixel) – ansonsten siehe auch LCD_Plot

`LCD_Box (x1, y1, x2, y2, Farbe)`
Zeichnet eine farbige Box. Wichtig: x1,y1 ist immer die obere linke Ecke, x2 und y2 die untere rechte Ecke im jeweiligen Modus

`LCD_Rect (x1, y1, x2, y2, Dicke, Farbe)`
Zeichnet einen farbigen Rahmen, Dicke legt fest ob 1 Pixel breit (0) oder 2 Pixel breit (1) – ansonsten siehe auch LCD_Plot.

`LCD_Circle(x1, y1, Radius, Füll, Dicke, Farbe)`
Bei Füll = 0 wird lediglich ein Rand gezeichnet, mit Füll=1 wird der Kreis gefüllt. Dicke legt wieder fest, ob ein Rand 1 Pixel oder 2 Pixel breit sein soll.

`LCD_Bitmap_65k(x1, y1, x2, y2, Bitmap, Compressed)`
Öffnet ein Ausgabefenster und füllt es mit genauso viel Bitmapdaten, wie das Fenster Pixel hat. Die Bitmapdaten (z.B. Fotos, Icons etc.) müssen als Array (siehe Beispiele) vorliegen. In

diesem Fall als 2-Byte-Werte (65.536 Farben). Der Marker Compressed zeigt an, ob die Daten komprimiert (1) oder unkomprimiert vorliegen.

LCD_Bitmap_256low(x1, y1, x2, y2, Bitmap, Compressed)

Öffnet ein Ausgabefenster und füllt es mit genauso viel Bitmapdaten, wie das Fenster Pixel hat. Die Bitmapdaten (z.B. Fotos, Icons etc.) müssen als Array (siehe Beispiele) vorliegen. In diesem Fall als 1-Byte-Wert (256 Farben nach RGB-332-Standard). Der Marker Compressed zeigt an, ob die Daten komprimiert (1) oder unkomprimiert vorliegen.

LCD_Bitmap_256high(x1, y1, x2, y2, Bitmap, Farbtabelle, Compressed)

Öffnet ein Ausgabefenster und füllt es mit genauso viel Bitmapdaten, wie das Fenster Pixel hat. Die Bitmapdaten (z.B. Fotos, Icons etc.) müssen als Array (siehe Beispiele) vorliegen. In diesem Fall als 1-Byte-Wert (256 Farben indiziert). Die für die Dekodierung der indizierten Farben notwendige Farbtabelle wird hier ebenfalls mitgeteilt. Der Marker Compressed zeigt an, ob die Daten komprimiert (1) oder unkomprimiert vorliegen.

Bei allen Koordinateneingaben gilt: X1 muss kleiner als X2 sein, Y1 kleiner als Y2 (also immer zuerst die linke und obere Position angeben, dann die untere und rechte).

Die Datenausgabe auf dem Display

Grundsätzlich funktioniert die Ausgabe von Daten auf dem Display immer auf die gleiche Art und Weise:

1) Durch Setzen der „Fensterkoordinaten“ wird dem Display mitgeteilt, in welchem Bereich des Display-RAMs es nun die nachfolgenden Daten (Pixel) setzen soll (zum Glück müssen Sie also bei Änderungen nicht immer das komplette Display neu beschreiben, das würde relativ lange dauern)

2) Jedes geschickte Datum (Singular von Daten) wird nun vom Display automatisch als Pixel angesehen und gesetzt. Es fängt dabei oben links im definierten Ausgabefenster an und wandert mit jedem empfangenen Pixel automatisch eine Spalte weiter nach rechts – ist die Fenstergrenze erreicht springt der Pointer eine Zeile tiefer und wieder an den linken Rand des Ausgabefensters.

3) Die Leitung **DC (**Data / Communication ID) des Displays teilt dem Display mit, ob die ankommenden Daten nun Pixeldaten oder Befehle darstellen. Eine logische 1 auf der Leitung DC bedeutet Befehl, eine 0 bedeutet Pixeldaten.

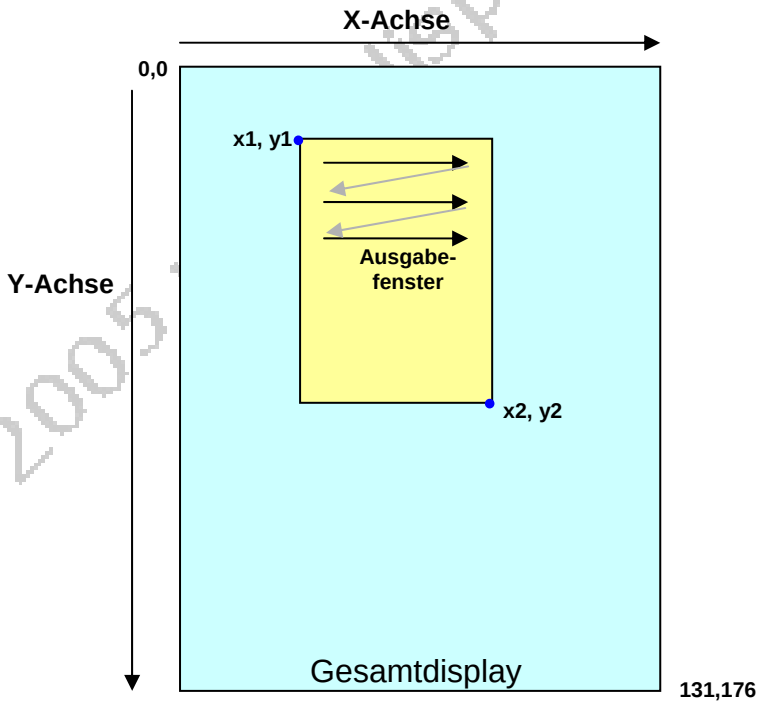
4) Ein Pixel setzt sich immer aus 2 Byte (65.536 Farben) zusammen, d.h. für jedes darzustellende Pixel müssen zwei Byte an das Display übertragen werden. Bei einem Vollbild müssen also 132x176x2 = 46 KByte seriell übertragen werden, also insg. 371.712 Bit.

Das Ausgabefenster

Nach Ermittlung der Ausgabedaten wird das Ausgabefenster „geöffnet“ und diese Pixeldaten ausgegeben. Bei der Ausgabe z.B. eines kleinen Schriftzeichens ist das Ausgabefenster immer genau 6 Pixel breit (5 Pixel für das Zeichen plus ein Pixel für eine Leerspalte – die Zeichen sollen ja nicht aneinander kleben) sowie 8 Pixel hoch. Größere Schriften sind entsprechend höher und/oder breiter.

Die Definition des Ausgabefensters geschieht in der Unterroutine **LCD_Window** durch vier Werte. X und Y Position der linken oberen Ecke sowie X & Y der rechten unteren Ecke. Bei einfachen Schriftzeichen liegt die rechte untere Ecke immer 6 Pixel weiter rechts und 8 Pixel tiefer als die linke obere Ecke. Bei doppelt großen Zeichen entsprechend mehr. Bei doppelt großen Zeichen werden die Zeichendaten der einfach großen Zeichen im Übrigen 2x ausgegeben. Das spart Speicherplatz, denn wir müssen hierfür keinen eigenen Font anlegen, allerdings sehen diese Zeichen dadurch etwas „gröber“ aus. Selbstverständlich ließen sich auch doppelt große Zeichen in höherer Auflösung in eigenen Data-Zeilen definieren. Diese Zeichen könnten dann feiner gestaltet werden, benötigen jedoch 4x mehr Speicherplatz (doppelte Auflösung vertikal und doppelte Auflösung horizontal). Für die meisten Anwendungen sollten die beigefügten Schriften ausreichend sein, aber Sie können jederzeit eigene Schriften oder Sonderzeichen definieren und integrieren.

Unsere Routine **LCD_Window** setzt die Fensterpositionen wie folgt:



X1 und Y1 legen also die linke obere Ecke, X2 und Y2 die rechte untere Ecke des Ausgabefensters fest. Die Pfeile zeigen die Ausgaberichtung der gesendeten Bits (Pixel) – also von links nach rechts.

Einführung in das Farbsystem des Displays

Farben bedeuten immer auch Speicherverbrauch und Rechenzeit, denn jedes Pixel braucht ja nicht nur 1 Bit für an oder aus, sondern auch noch die Informationen über die entsprechende Farbe. Dieser unterscheidet ein Farbdisplay von einem monochromen Display.

Die üblichen Grafikformate wie BMP, JPG etc. kennen 16 Millionen und mehr Farben, das sind 24 Bit an Information für jedes einzelne Pixel – 8 Bit für jede Farbe (Rot, Grün und Blau – durch Mischung dieser drei Farben wird jede der 16 Millionen Farben dargestellt). Für jede dieser drei Grundfarben gibt es also 256 Intensitätswerte.

Übliche Grafikprogramme arbeiten mit CMYK (wird nur für den Druck auf Papier genutzt und ist daher für das Display irrelevant) und mit RGB-Farben (RGB: **R**ot, **G**rün, **B**lau). Dort können Sie jede Farbe in 256 Schattierungen darstellen (0 bis 255). Aus der Kombination von Rot, Grün und Blau erhalten wir alle anderen sichtbaren Farben. Die Kombination von 256 x 256 x 256 Farben ergibt am PC eine Fülle von 16 Millionen Farben.

TIPP

Wenn Ihnen dies nicht geläufig ist, starten Sie zum Ausprobieren entweder ein Grafikprogramm oder z. B. Microsoft Word®: So können Sie z. B. nach Zeichnen eines Rechteckes dieses mit Doppelklick bearbeiten, auf *Farben* klicken, dann *weitere Farben* und dann *Benutzerdefiniert*. Dort können Sie die gewünschte Farbe aus den 3 x 256 Varianten mischen.

Beispiele der Farbmischung am PC:

Rot	Grün	Blau	Ergebnis	Bemerkung	Darstellung
0	0	0	Schwarz	Kein Farbanteil einer Farbe	
255	255	255	Weiß	Jede Farbe zu 100%	
255	0	0	Rot	100% Rot, keine andere Farbe	
136	0	0	Dunkelrot	Nur 50% Rot	
0	255	0	Grün	100% Grün	
0	0	255	Blau	100% Blau	
255	255	0	Gelb	Je 100% Rot und Grün gemischt	
136	0	136	Lila	Je 50% Rot und Blau gemischt	
119	119	119	Mittelgrau	Je Farbe 47%	
255	136	0	Orange	100% Rot und 50% Gelb	

Eine bildschirmfüllende Grafik mit 176x132 Pixel (=23.232 Pixel) bräuchte bei 8 Bit Farbinformation pro Farbe (RGB) jedoch 3x8=24 Bits an Farbinformationen pro Pixel, das sind insg. fast 70 KByte Programmspeicher. Daran sieht man schon, dass wir die Farbinformationen reduzieren müssen, denn 70 KByte haben wir in einem Mikrocontroller selten übrig.

Native Farbmodi des Displays: 256 oder 65.536 Farben

Das vorliegende Display arbeitet je nach Start-Initialisierung mit 256 oder mit 65.536 Farben. Im 256 Farben Modus wird für jedes Pixel ein Byte an Farbinformation übertragen, im 65.536 Farben Modus sind 2 Byte für die Farbinformation notwendig. Unsere mitgelieferte Software konzentriert sich auf den 65K-Farben Modus, kann jedoch schnell und problemlos auf 256 Farben geändert werden. Der Vorteil des 256 Farben Modus ist die quasi verdoppelte Ausgabegeschwindigkeit, da nun nur noch 8 Bit statt 16 Bit pro Pixel übertragen werden müssen. Leider vermag es das Display nicht, während des Betriebs einfach nur umgeschaltet zu werden. Dies bedarf einer Neuinitialsierung (also Aufruf von LCD_Init) mit dem entsprechenden Zeitbedarf. Ein sinnvolles Mischen beider Modi ist daher schwierig, aber natürlich möglich. Denkbar wäre z.B. das Zeigen eines 65K-Farben-Startlogos, dann eine Neuinitialisierung und ein Betreiben der „Hauptsoftware“ mit 256 Farben.

Ein Nachteil des 256 Farben Modus ist die Beschränkheit auf die vorgegebenen 256 Farben die sich nicht vordefinieren lassen. Das ist für reinen Text oder gezeichnete Grafikdaten in der Regel ohne Belang, Fotos oder Logos jedoch lassen sich in diesem Modus jedoch häufig nicht gut darstellen.

Alternative II: xxx aus 65.536 Farben (Farbtabelle)

Dieser Modus arbeitet mit indizierten Farben (auch Palette-Modus genannt) und ist weitaus interessanter als der vorhergehende Modus (RGB 332). Er erlaubt die Nutzung indizierter Grafikdateien (z.B: GIF oder BMP). Im Gegensatz zum oben erläutertem 256-Farben Modus, der nach dem festen Schema je 3 Bit für Rot und Grün sowie 2 Bit für Blau arbeitet, werden hier alle echten 65.536 Farben ermöglicht. Es gibt keine Einschränkung auf z.B. lediglich 4 Blautöne – Sie können aus dem vollen Farbraum des Displays schöpfen, jedoch bei einer Grafik immer nur maximal 256 verschiedene Farben gleichzeitig. Wenn Sie möchten, kann Ihre Grafik 200 Blautöne und 56 Röttöne beinhalten.

Dies wird durch eine vorgelagerte Farbtabelle erreicht. Anstatt nun den einzelnen Pixeln direkt die Farbe zuzuweisen, enthalten die einzelnen Pixeldaten der Grafikdatei einen Pointer auf eine Tabelle, in der die korrekte Farbe abgelegt ist. Bei einer Grafik mit 256 verschiedenen ! Farben, ist die Farbtabelle also 256 Felder groß – und jedes dieser Felder enthält 2 Byte mit einer 16-Bit-Farbe (also 65.536 mögliche). Eine Grafik mit 16 verschiedenen Farben erhält eine Farbtabelle mit 16 Feldern und je einer 16-Bit-Farbe.

Diese Art Grafiken begegnen Ihnen jeden Tag – im Internet. Die üblichen GIF-Grafiken arbeiten immer mit indizierten Farben und können niemals mehr als 256 Farben gleichzeitig darstellen.

Der Vorteil: Jedes Pixel kann aus dem vollen Farbspektrum schöpfen, braucht aber trotzdem nur 1 Byte Farbinformation pro Pixel. Eine 176 x 132 Pixel große Grafik benötigt trotz des vollen Farbumfangs von 65.536 Farben somit statt 46.464 Byte nur noch 23.232 Byte. Hinzurechnen muss man allerdings die Farbtabelle von einer Größe von bis zu 512 Byte (pro Farbe 2 Byte Farbinformation für die Nutzung von 65536 Farben)

Tipp 1: Wenn Sie mehrere ähnlicher Grafiken nutzen möchten, arbeiten Sie am besten immer mit der gleichen Farbtabelle, dann müssen Sie diese auch nur 1 mal und nicht für jedes Bild einzeln ablegen (also auch nur 1x Speicher hierfür verbrauchen). Diese Farbtabelle können Sie z.B. in Adobe Photoshop® unter dem Menü „Image – Mode – Color Table und dann Save“ abspeichern und für das Umwandeln einer z.B. JPG-Grafik in das GIF-Format vorher im gleichen Menü mit „Load“ auswählen und einladen. Das macht natürlich nur dann Sinn, wenn Ihre Grafiken eine ähnliche Farbgebung haben. Ein Tipp hierfür: Wenn Sie bereits wissen, welche Grafiken Sie alle nutzen möchten, platzieren Sie alle Ihre Grafiken auf einmal im Grafikprogramm auf einer Seite und lassen sich dann eine Farbtabelle vorschlagen. Wenn alle Ihre Bilder damit OK aussehen, brauchen Sie auch nur diese eine Farbtabelle ablegen und alle Bildelemente können damit umgewandelt werden.

Tipp II: Wenn Sie eine Grafik mit nur 16 oder weniger Farben nutzen möchten, so kommt eine weitere praktische Variante zum Tragen: Der Pointer auf die 16 möglichen Tabellenzellen benötigt hierfür nur 4 Bit. Dies bedeutet, in jedes Byte der Pixeldaten passen nun 2 Pixel. Bei nur 4 Farben benötigen wir sogar nur 2 Bit – es passen 4 Pixel in ein Byte und bei nur 2 Farben passen 8 Pixel in ein Byte.

	65.536 Farben	**256 Farben**	**16 Farben**	**4 Farben**	**2 Farben**
Speicherbedarf Vollbild mit 176x132 Pixeln	46.464 Byte	23.232 Byte	11.616 Byte	5.808 Byte	2.904 Byte

Um solche Grafiken einzubinden, wählen Sie im Tool LCD_Convert bitte die Option „Palette“. Mit dieser gewählten Option können Sie ausschließlich Grafiken behandeln, die bereits eine Farbtabelle beinhalten. Das sind sowohl GIF-Dateien als auch indizierte 8-Bit-BMP-Dateien (und nur solche, normale BMP-Dateien sind in diesem Modus nicht möglich).

Solche GIF-Dateien erstellen Sie z.B. in Adobe Photoshop® unter den Menüs „Image – Mode – Indexed Color“. Sie wählen dann dort die Anzahl der Farben aus und erkennen auch schnell, mit wie vielen (wenig) Farben Ihre Grafik noch akzeptabel aussieht.

Grafikkomprimierung / Dekomprimierung

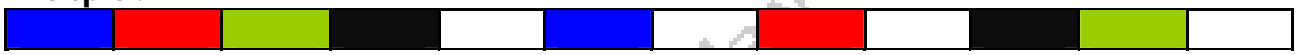
Unser Tool LCD_Convert bietet auch eine Option zur Grafikkomprimierung. Da der Programmspeicher im Mikrocontroller begrenzt ist, bietet sich eine Grafikkomprimierung geradezu an. Allerdings würde eine Dekomprimierungsroutine à la JPG den Mikrocontroller überfordern. Die Rechenzeit wäre einfach zu lang und der notwendige RAM-Speicher einfach zu groß.
Daher wurde hier der relativ einfache RLL (Run length limited) Code verwandt. Im Nachfolgenden erläutern wir das Komprimierungsverfahren.

Kurz vorab zusammengefasst: Mehrfach hintereinander auftretende gleichfarbige Pixel werden zusammengefasst als [Farbe – Anzahl].

Im Detail (hier am Beispiel des 65.536-Farben-Modus):

Steht ein farbiges Pixel alleine, d.h. ist dem darauf folgenden Pixel eine andere Farbe zugeordnet, so wird dieses wie bisher auch einzeln abgelegt.

Beispiel:



Datendarstellung:

h001F	hF800	h07E0	h0000	hFFFF	h001F	hFFFF	hF800	hFFFF	h0000	h07E0	hFFFF

D.h. Wenn niemals 2 gleiche Pixel aufeinander folgen, ist eine Komprimierung nicht möglich. 12 Pixel benötigen 12 x 2 Byte = 24 Byte

Wenn ein Pixel sich wiederholt, d.h. haben 2 aufeinander folgende Pixel die identische Farbe, wird dieses Pixel 2 x hintereinander im Datenbereich aufgeführt und als nächstes Datum folgt die Anzahl der Wiederholungen.

Beispiel (6 blaue und 4 weiße Pixel):

Datendarstellung:

h001F	h001F	h0004	hFFFF	hFFFF	h0002	h07E0	hFFFF
2 x Blau		Blau 4 x wiederholt	2 x Weiß		Weiß 2 x wiederholt	Grün	Weiß

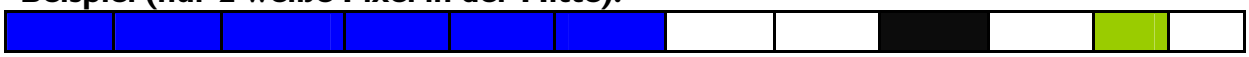
Sind mindestens 2 aufeinander folgende Pixel identisch, wird <u>immer</u> als nächstes die Anzahl der Wiederholungen eingefügt.

<u>Hinweis:</u> Wenn die Daten im 65.000 Farbenmodus abgelegt werden, werden für jedes Pixel 2 Byte an Daten abgelegt – auch die Anzahl der Wiederholungen wird dann in 2 Bytes gespeichert.

12 Pixel benötigen im obigen Beispiel 8 x 2 Byte = 16 Byte (33% Reduktion)

Information zum Sonderfall 2 Pixel: Wenn eine Pixelfarbe nur 2 Mal aufeinander folgt, besitzt das dritte Byte (welches die Anzahl der Wiederholungen beinhaltet) den Wert 0.

Beispiel (nur 2 weiße Pixel in der Mitte):



Datendarstellung:

h001F	h001F	h0004	hFFFF	hFFFF	h0000	h0000	hFFFF	h07E0	hFFFF
2 x Blau		Blau 4 x wiederholt	2 x Weiß		Keine weitere Wiederh.	Schwarz	Weiß	Grün	Weiß

Aus diesem Fall resultiert, dass die Gesamtmenge an Daten zwangsläufig steigt, wenn sehr häufig Pixel lediglich doppelt vorkommen, denn dann verbrauchen diese beiden Pixel durch das zusätzliche Datum (0 Wiederholungen) sogar mehr Speicher – die Komprimierung kann also auch zum Gegenteil führen: die Datenmenge wird aufgebläht.

Die Dekomprimierung:
Die oben erläuterte Komprimierungsroutine kann vom Mikrocontroller relativ einfach dekomprimiert werden. Dazu ist es nur notwendig zu ermitteln, ob die gleiche Farbe 2x hintereinander vorkommt. Wenn ja, beinhaltet der nächste Wert die Anzahl der Wiederholungen, wenn nein, werden die Daten wie üblich als eigenes Pixel behandelt.

Komprimierung und Farbtiefe
Es liegt auf der Hand, dass eine Komprimierung bei Bildern, die nur wenige Farben aufweisen, Erfolg versprechender ist, als bei einem Bild, welches viele unterschiedliche Farben in nicht zusammenhängenden Bereichen beinhaltet.

Daher lassen sich Bilder mit 256 Farben in der Regel auch stärker komprimieren, als Bilder mit 65.536 Farben.

Das Display3000-Logo z.B. (siehe rechts, befindet sich auch auf der CD) beinhaltet nur max. 256 unterschiedliche Farben und hat große Bereiche mit identischen Farben. Daher kann es, egal ob im 256-Farben-Modus (Byte) oder im 65K-Modus (Word), auch mit 76% bzw. 71% komprimiert werden. Es macht schon etwas aus, ob eine Grafik im Controller 46.464 Bytes oder nur noch ca. 13.000 Byte benötigt.

Hinweis: im reinen! 256-Farben Modus (RGB332) sieht die nebenstehende Grafik dann allerdings nicht mehr ganz so gut aus, denn dort wird ja aktuell noch eine feststehende Farbtabelle genutzt, die z.B. nur 3 bzw. 4 blaue Schattierungen kennt.

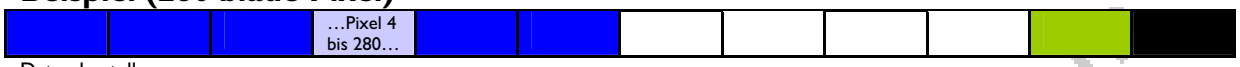
Hinweis: Wenn die Daten im 256-Farbenmodus abgelegt werden, wird für jedes Pixel 1 Byte an Daten abgelegt – die Anzahl der Wiederholungen wird dann ebenfalls in nur 1 Byte gespeichert – es sind also nur max. 255 Wiederholungen eines Pixels möglich – sind mehr Pixel zu wiederholen, fängt die Sequenz von vorne an.

Komprimierung bei Nutzung indizierter Farben
Die Komprimierung des Programms LCD_Convert funktioniert bei indizierten Farben immer nach dem Byte-Prinzip – die Anzahl der Wiederholungen werden ebenfalls in einem Byte gespeichert (es sind also max. 255 Wiederholungen pro Pixeldatum möglich).

Beispiel – Grafik mit 256 indizierten Farben
Farbtabelle (hier 256 Werte)

Farbe 01 = Blau	Farbe 02 = Grün	Farbe 03 = Rot
Farbe 04 = Weiß	…… Farbe 05 bis FE……	Farbe FF = Schwarz

Beispiel (280 blaue Pixel)



Datendarstellung:

h01	h01	hFF	h01	h01	h15	h04	h04	h02	h02	hFF
2 x Blau		Blau 255 x wiederholt	2 x Blau		Blau noch 21 x wiederholt	Weiß	Weiß	Weiß noch 2 x Wiederholt	Grün	Schwarz

2 + 255 + 2 + 21 = 280 blaue Pixel

Besonderheit bei reduzierter Farbanzahl und reduziertem Speicherbedarf
LCD_Convert kann Grafiken mit indizierten Farben so ablegen, dass bei der Nutzung von wenigen Farben auch weniger Speicher benötigt wird. Bei der Beschränkung auf z.B. 16 Farben, kann das Konvertierungsprogramm pro Byte 2 Pixel ablegen (jedes Pixel braucht dann nur 4 Bit); bei 8 Farben können es 4 Pixel pro Byte sein etc.
Diesen Modus unterstützen wir bei unseren Grafikroutinen z. Zt. nicht, aber dies können Sie bei Bedarf auch sicher selbst programmieren.

Sonderfall: Eine Besonderheit bei der Komprimierung zu Tage, wenn mehrere Pixel sich ein Byte teilen: Der Kompressionsmodus betrachtet immer nur komplette und zu wiederholende Bytes und nicht die einzelnen, zu wiederholenden Pixel.

Beispiel: 16 Farben-Modus (=2 Pixel pro Byte)

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10

Datendarstellung:

h31		**h11**		**h04**	**h12**		**h22**		**h24**	
h03	h01	h01	h01	h04	h01	h02	h02	h02	h02	h04
Rot	Blau	Blau	Blau	4 Wiederholungen	Blau	Grün	Grün	Grün	Grün	Weiß

Erläuterung: Byte 1 ist nicht identisch mit Byte 2 – daher keine Wiederholung, Byte 3 ist identisch mit Byte 2 und wird dann noch 4x wiederholt (Byte 4-7); Byte 8 ist nicht identisch (daher folgt also noch das einzelne blaue Pixel in der Ausgabe) und auch die folgenden 4 grünen Pixel werden einzeln als Datum abgelegt, da die Bytegrenzen, innerhalb deren sie sich befinden, keine Wiederholung zulassen.

Erstellung und Ausgabe von Grafikelementen / Fotos
Nach der Theorie nun die Praxis: Nun kommen wir zur Ausgabe komplexer grafischer Elemente wie Fotos oder Icons. Praktisch ist die Ausgabe identisch zur Ausgabe einer farbigen Box (wie z.B. mit der Routine LCD_Box). Der einzige Unterschied besteht darin, dass nun nicht jedes Pixel eine identische Farbe, sondern seine individuelle Farbe erhält.

Das größte Problem: Wenn Sie die grafischen Elemente nicht programmtechnisch (wie z. B. das Rechteck in obigem Beispiel) erzeugen möchten oder können, müssen Sie die Pixeldaten für diese Grafik im Programmspeicher oder einem externen Speicher ablegen (oder auch dem Eeprom).

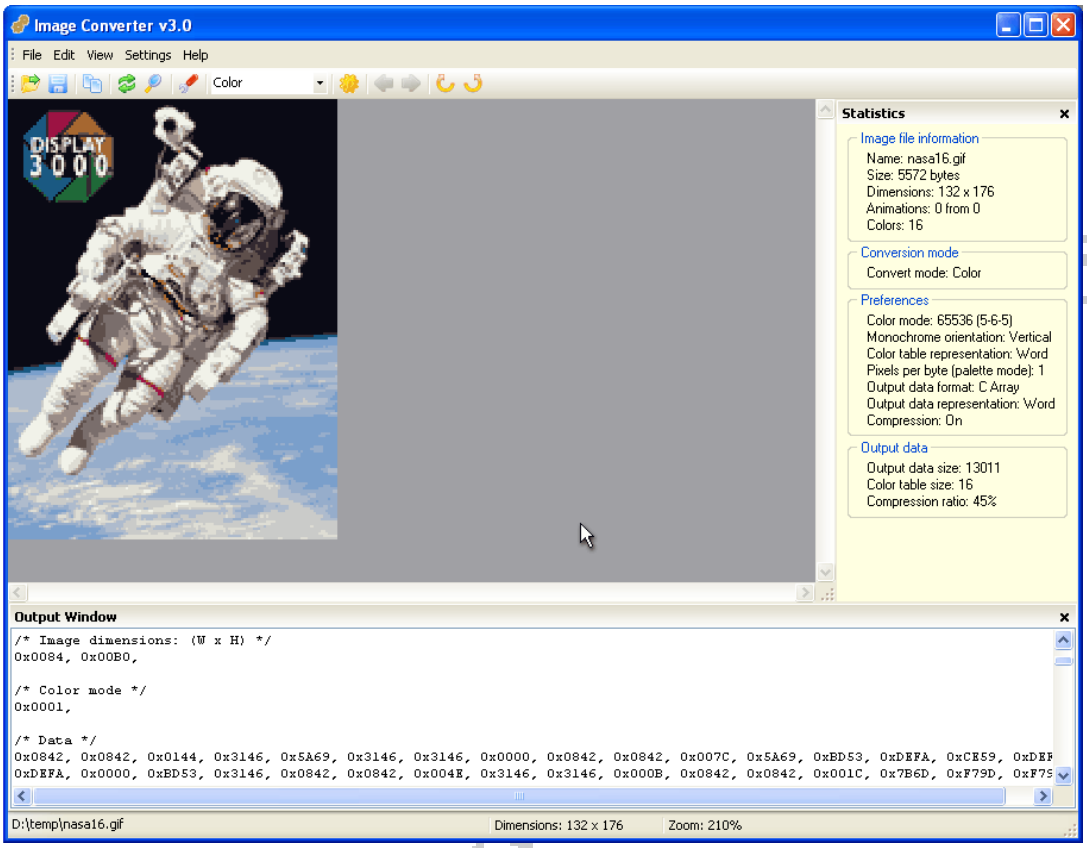
Abgesehen davon, dass diese Datenmengen je nach Größe und Anzahl der Farben der Grafik eine Menge Speicher belegen, müssen diese als Speicherbereich in die Software eingefügt werden (z.B. als Data-Bereich in Basic oder als Array in C).

Leider kennen wir kein Grafikprogramm, welches eine Grafik im notwendigen Format (näheres hierzu im Kapitel Einführung in das Farbsystem ab Seite **Fehler! Textmarke nicht definiert.**) bereits als Datenwerte ausgibt und brauchten hierfür eine Lösung. Das auf der CD vorhandene Grafik-Werkzeug GLCD_Convert wurde nach unseren Vorgaben erstellt: Das komfortable Windows-Programm erstellt aus einer Grafik entweder Data-Zeilen für Basic oder direkt C-Header-Zeilen. Auch Funktionen für die Komprimierung und die Nutzung indizierter GIFs/BMPs sind enthalten. Hierzu später noch mehr.

Das Windows-Programm G*LCD_Convert.exe* befindet sich auf der CD. Für den ersten Start befinden sich im Verzeichnis \Grafiken auch bereits einige Grafikdateien in der richtigen Größe auf der CD.

Der GLCD_Converter

Dieses Programm wurde speziell für dieses Display geschrieben, bei der Entwicklung wurden jedoch auch unsere zukünftigen Weiterentwicklungen im Hinterkopf behalten, so dass diese Software auch in Zukunft noch nutzbar sein wird.



Der Bildschirm besteht neben der Menüzeile aus 3 Hauptbereichen

a) Der Input: Hier wird die geladene Grafikdatei angezeigt.
b) Das Ausgabefenster: Hier werden nach Drücken von *F5* (oder *Edit/Convert*) die für den Mikrocontroller aufbereiteten Daten angezeigt.
c) Das Statistikfenster: Hier werden die Daten über die geladene Grafik, die gewählten Parameter sowie Informationen zur aktuellen Konvertierung gezeigt.

Die Bedienung:

Die Bedienung ist sehr einfach und fast selbsterklärend.

1) Mittels Drag and Drop (wenn in den Preferences eingeschaltet) oder mittels *File/Open* laden Sie Ihre Grafikdatei ein.
2) Unter *Settings/Preferences* stellen Sie die gewünschten Konvertierungsparameter ein. Diese werden in einer Konfigurationsdatei abgelegt, so dass Sie bei gleich bleibenden Parametern diese nur 1x einstellen müssen. Tipp: Sie erkennen diese Einstellungen auch bereits im Statistics-Fenster und können die wichtigsten Punkte dort auch direkt anklicken.
3) Mit *F5* starten Sie den Konvertierungsvorgang.
4) Im Ausgabebereich werden die Daten angezeigt.

…
5 fehlende Seiten
zum Grafikkonverter

…
4 fehlende Seiten
mit Tipps zur Pro-
grammierung

…
11 fehlende Seiten
mit Referenz zum
Display und der Pro-
grammierung

…
6 fehlende Seiten
…

Kontakt:

Speed IT up
Inhaber Peter Küsters
Wekeln 39
47877 Willich
Telefon: (0 21 54) 88 27 5-10
Telefax: (0 21 54) 88 27 5-22

Weitere Informationen und Updates: www.display3000.com

Autor dieses Manuals: Peter Küsters.