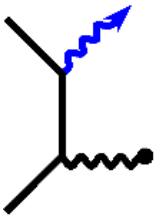
Quickstart Tutorial

für die Simplify Technologies GUI-Bibliothek



Simplify Technologies GmbH
Steinbühlstraße 15
35578 Wetzlar

Simplify Technologies GmbH
Steinbühlstraße 15
35578 Wetzlar
Tel.: (+49) (0)6441-210390
FAX.: (+49) (0)6441-210399
Internet: www.simplify-technologies.de
Email: hansjuergen.dreuth@simplify-technologies.de

Inhaltsverzeichnis

In diesem Tutorial sollen die nötigen Schritte beschrieben werden, um die GUI–Bibliothek einsetzen zu können.
Der Einfachheit halber wird zunächst nur die Erstellung eines klassischen “Hello World”–Programms erläutert, welches einen Text auf das LC–Display schreibt. Dieses Beispiel wird anschließend um weitere Grafikfunktionen erweitert, so daß Sie einen Eindruck von den Möglichkeiten der Bibliothek erhalten.
Sollen über die grafische Darstellung hinaus auch Informationen eines Touch–Glases eingelesen und verarbeitet werden, dann muß die zugrundelegende Hardware über einen Systemtimer verfügen. Die erforderlichen Konfigurationen werden hierzu beschrieben. Danach wird auf die Kalibrierung und die Benutzung des Touch–Glases eingegangen.
Alle Beispiele in diesem Tutorial befinden sich in den entsprechenden Ordnern auf der CD der GUI–Bibliothek (siehe die Datei “Inhalt.txt”).

1 Grundlagen

Um möglichst schnell mit der Simplify–Technologies GUI–Bibliothek arbeiten zu können, empfehlen wir Ihnen folgende Vorgehensweise:

1. Kopieren Sie die Dateien der GUI–Bibliothek und der Hardwaretreiber auf Ihre Festplatte (an einen beliebigen Ort, den Sie für Ihr Projekt als geeignet empfinden). Das Entpacken und Installieren der Dateien wird in der Datei “INSTALL.TXT” beschrieben.

2. Fügen Sie in Ihrer Entwicklungsumgebung sämtliche Dateien der Bibliothek zu Ihrem aktuellen Projekt (im folgenden Abschnitt wäre dies das Projekt “Hello World”) hinzu. Dieses Hinzufügen von Dateien ist in der Anleitung Ihrer Entwicklungsumgebung beschrieben.

3. Passen Sie die Datei portab.h an die Gegebenheiten Ihrer Entwicklungsumgebung an: Da der ANSI–C–Standard keine festen Längen für Integer–Variablen hat, sondern nur Mindestlängen vorschreibt, wurden in der gesamten Bibliothek nur die Integer–Datentypen verwendet, die in “portab.h” als `uint8`, `int8`, `uint16` usw. definiert sind (die Zahl gibt hier an, wieviele Bits für die einzelnen Variablen zur Verfügung stehen). Welche Variablengrößen Ihr Compiler verwendet, entnehmen Sie bitte der Datei “limits.h”, die mit dem Compiler mitgeliefert wurde. Ändern Sie dann entsprechend die Definitionen in “portab.h”.

4. Die zur Konfiguration der GUI–Bibliothek eingesetzten Dateien “config_gui.h ”und “config_hardware_gui.h ”müssen ebenfalls an die Gegebenheiten angepaßt werden. Die folgenden Abschnitte geben dazu entsprechende Hinweise.

2 Hello World

Damit Sie auf Ihr LCD zugreifen können, benötigen Sie einen passenden Treiber für den verwendeten LCD-Controller. Alternativ dazu können Sie die hardwarespezifischen Treiberfunktionen selber implementieren (siehe die offengelegte Schnittstelle im Handbuch).

2.1 Konfiguration der GUI–Bibliothek

Die Konfiguration der Bibliothek erfolgt über die Dateien "config_hardware_gui.h" und "config_gui.h", so daß es nicht notwendig ist den Source–Code der Funktionen der GUI–Bibliothek zu modifizieren.
In "config_hardware_gui.h" werden Variablen definiert, durch die die Hardwareumgebung für das User-Interface des Systems beschrieben wird (z. B. die Auflösung des Displays). Variablen für Hardwarekomponenten, die Sie nicht verwenden (z. B. Touch-Glas) können Sie hier unberücksichtigt lassen.

Um für Ihre Systeme keinen unnötigen Code mit einzubinden, steht hierfür die Konfigurationsdatei "config_gui.h" zur Verfügung, in der über `#define` eingestellt werden kann, welche Module benötigt werden, und welche Umgebungsvoraussetzungen gegeben sind.
Die Datei "config_gui.h" enthält zwei Sektionen: In Sektion 1 werden Einstellungen zur Softwareumgebung vorgenommen. Hier können Sie auch auswählen, welche Komponenten der Bibliothek in Ihrem Zielsystem verwendet werden sollen. Durch die bedingte Compilierung auch innerhalb einzelner Funktionen wird so verhindert, daß der Code unnötig Speicher belegt.

Beachten Sie bitte, daß die nicht ausgewählten Komponenten auch nicht für die Programmierung der Applikation zur Verfügung stehen. Aufrufe solcher Funktionen würden daher zu Compiler–Fehlern führen, ohne daß dies einen Defekt in der GUI-Bibliothek bedeutet. Nach Einer Änderung der Konfiguration ist deshalb auch oft eine Neukompilierung aller Dateien, die die Datei config_gui.h einbinden, notwendig.

In der darauffolgenden Sektion 2 wird Ihre Auswahl ausgewertet. Sollten Sie Einstellungen gewählt haben, die nicht miteinander verträglich sind, werden Sie durch Fehlermeldungen beim Compilieren darauf aufmerksam gemacht (so ist es z. B. nicht sinnvoll möglich, Touch–Tastaturen zu haben, ohne ein Touch–Glas mit den Touch–Funktionen zu benutzen).

Zur Darstellung von Text und einfacher Grafik sind jedoch nur sehr wenige Einträge nötig.

2.1.1 config_hardware_gui.h

Zur Ansteuerung der LCD–Hardware ist im wesentlichen nur die Auflösung mit den Optionen `_LCD_PHYS_SIZE_X` und `_LCD_PHYS_SIZE_Y` vorzugeben und das LCD mit `_LCD_PRESENT = 1` zu aktivieren. Tragen Sie bei `_LCD_PHYS_SIZE_X` und `_LCD_PHYS_SIZE_Y` die entsprechenden Werte Ihres Displays ein. Weiterer Hardware–Funktionen, etwa für das Touch-Glas und den Lautsprecher werden abgeschaltet. Die Datei "config_hardware_gui.h" sieht dann im wesentlichen folgendermaßen aus:

```
/*************************************************************************/
/* config_hardware_gui.h                                                 */
/* Defines hardware and options                    .                     */
/*                                                                       */
/* (C) SIMPLIFY TECHNOLOGIES GmbH, www.simplify-technologies.de          */
/*************************************************************************/
```

```
#ifndef __CONFIG_HARDWARE_GUI_H
#define __CONFIG_HARDWARE_GUI_H


/*******************************/
/* defines for the lcd display */
/*******************************/
#define _LCD_PRESENT         1 /* makes display available */
#define _LCD_LIGHT           0
#define _LCD_LIGHT_ADJUST    0
#define _LCD_CONTRAST_ADJUST 0
#define _LCD_INVERTED        0

#define _LCD_PHYS_SIZE_X   320 /* width in pixels */
#define _LCD_PHYS_SIZE_Y   240 /* height in pixels */


/*******************************/
/* other functions are shut off */
/*******************************/
#define _TGLASS_PRESENT      0 /* no touch glass */
#define _PIEZO_PRESENT       0 /* no piezo loudspeaker */


#endif /* __CONFIG_HARDWARE_GUI_H */
```

Die Originaldatei der Distribution enthält ausführliche Kommentare und ist entsprechend umfangreicher.

2.1.2 config_gui.h

Wie weiter oben schon erwähnt besteht die Datei “config_gui.h” aus zwei Sektionen, wobei der Nutzer der GUI–Bibliothek Veränderungen nur in der ersten Sektion durchführen sollte. Eine genaue Beschreibung der einzelnen Optionen finden Sie im Handbuch und in den Kommentaren in der Datei selbst. Für das erste Beispiel reichen die folgenden `#define`–Anweisungen aus in der ersten Sektion von “config_gui.h” aus. **Alle nicht erwähnten Optionen müssen Sie auskommentieren und die zweite Sektion der Datei (internal definitions) unberührt lassen.**

```
/*****************************************************************************/
/* config_gui.h                                                              */
/* Defines configuration options for the GUI Library     .                   */
/*                                                                           */
/* (C) SIMPLIFY TECHNOLOGIES GmbH, www.simplify-technologies.de              */
/*****************************************************************************/


#ifndef __CONFIG_GUI_H
#define __CONFIG_GUI_H


/* read definition of hardware features used by the GUI */
#include "config_hardware_gui.h"


/*###########################################################################*/
/* Defines for customizing the GUI library                                   */
/*                                                                           */
/* Conditional compilation is used to tailor the code so that only the       */
/* modules and options needed are included into the binary thus optimizing   */
/* memory usage and execution speed.                                         */
/*                                                                           */
/* Please carefully read the following customizing options and activate the  */
```

```
/* defines if you want to activate the respective feature                    */
/*###########################################################################*/


/*****************************************************************************/
/* general customization                                                     */
/*****************************************************************************/

/* define here how many single system ticks occur during one second. Please note
that the maximum value allowed is 49700 */

#define _TICKS_PER_SECOND 100


/* the following define does not need to be changed. It allows you to call the
function system_wait_ticks like this: system_wait_ticks(3*SEC); */

#define SEC _TICKS_PER_SECOND


/* if you do not define _SYSTEM_TIMER_PRESENT the only timing function available
will be system_wait_ticks, which will just be waiting in a loop for a while.
In that case if you want to use the system_wait_ticks function define the
following calibration constant to a value that timing by software delay is
allright (for a Hitachi H8/300H with 14.7456 MHz a value of 370000 is ok): */

#define _SOFTWARE_DELAY_NUMBER 370000


/* Define the following, if your system features dynamic memory
allocation as required by ANSI-C. */

/*
#define _MALLOC_AVAILABLE
*/


/* If you do not provide dynamic memory allocation, a simple memory
allocator is provided, which can be enhanced if neccessary.
This memory allocater uses a static heap, which size can be configured with
_STORAGE_ALLOCATER_HEAP_SIZE (in bytes).
A list of free memory areas is needed as well, to organize the heap. This
list has a maximum number of entries (_MAX_FREE_LIST_ENTRIES). */

#define _STORAGE_ALLOCATOR_HEAP_SIZE 5000
#define _MAX_FREE_LIST_ENTRIES       30


/* Defines how many bytes a single display line on a virtual display may need.
This should be defined always. */

#define _MAX_BYTES_PER_LINE 320


/* if you are sure that connection of devices to channels is done correctly in
your application and no bogous pointers would mess up the system, you may omit
the checks for verifying these connections. In embedded systems with fixed
application code you eventually will reach this stage anyway ;-)
Note that parameter checking is not ommited */

#define _MAKE_CHECKS


/*****************************************************************************/
/* customizing the display output                                            */
/*****************************************************************************/

/* define, if you want to use the lcd device (i. e. a lcd display) */
```

```
#define _USE_LCD_DEVICE


/* define, if you want to use the sprite functions */

#define _SPRITE_WANTED


/* please define how many bits per pixel will be used to represent the color or
   grayshade of one LCD pixel (in monochrome: 1, in 256 colors: 8) */

#define _BITS_PER_PIXEL 1


/* if you want to include the code for the functions for lines, define: */

#define _GENERAL_LINES_WANTED


/* if you want to include the code for black and white bmp pictures define: */

#define _BMP_WANTED


/* if you want to include the code for the functions for circles and arcs,
define: */

#define _CIRCLES_WANTED


/*############################################################################*/
/*############################################################################*/
/*  Internal definitions based on what was defined above,                     */
/*     !!! NO CHANGES REQUIRED BELOW THIS LINE !!!                            */
/*############################################################################*/
/*############################################################################*/


...


/*############################################################################*/
#endif /* __CONFIG_GUI_H */
```

2.2 Das Programm

Als Ausgangsbasis dient ein möglichst einfaches Programm, daß auf dem Bildschirm lediglich einen Text ausgibt (traditionell hat sich hier *Hello World* bewährt). Das Projekt dieses Programms befindet sich auf der CD.

```
/*****************************************************************************/
/* main.c      "Hello World"                                                 */
/*                                                                           */
/* (C) SIMPLIFY TECHNOLOGIES GmbH, www.simplify-technologies.de              */
/*                                                                           */
/* last change: 7 Mar 2006                                                   */
/*****************************************************************************/

#include "main.h"              /* includes for this file                     */
#include "platform_periphery.h" /* includes for special features of the      */
                               /* hardware                                   */

#include "portab.h"      /* includes for operating system functions and defs */
#include "errors.h"      /* error codes                                      */
```

```
#include "system.h"      /* includes for the "kernel"                        */
#include "lcd.h"         /* includes lcd-driver                              */
#include "display.h"     /* includes for graphical functions                 */
#include "mono8x16.h"    /* includes of the font data                        */


int main( void)
{
   lcd_device      *system_lcd;  /* pointer to lcd driver                    */
   display_channel  display;     /* data structure for the drawing functions */

   platform_periphery_init();    /* init special periphery of your hardware  */
   system_init();                /* initialize system timer (not yet) and    */
                                 /* font list                                */

   /* define a screen with your systems resolution                          */
   lcd_init( _LCD_PHYS_SIZE_X, _LCD_PHYS_SIZE_Y, LCD_NORMAL, &system_lcd);

   disp_open( &display);         /* generate the display and connect it to   */
   disp_make_current( &display); /* lcd-driver, which controlls the          */
   disp_connect( system_lcd);    /* hardware                                 */

   disp_set_font( __Mono8x16);          /* desired font for text output on   */
   disp_set_drawmode( DRAW_SET);        /* the display                       */
   disp_set_textstyle( TXT_NORMAL);

   disp_set_cursor_position( 100, 120); /* print some text at the defined    */
   disp_text( "Hello World!");          /* position                          */
   disp_set_cursor_position( 100, 100);
   disp_text( "Hello User!");

   /* cleanup would shut off the lcd and also the printed text               */
   /*
   disp_disconnect();
   disp_close( &display);
   lcd_close();
   */

   while (1) {}; /* do something ... */

   return (ERR_OK);
}/* main */
```

Betrachten wir nun die einzelnen Abschnitte des Quellcodes. Als erstes müssen Sie natürlich ihre Systemhardware initialisieren und die Fonts anlegen. In diesem Fall wird nur ein Font zur Verfügung gestellt:

```
   platform_periphery_init();    /* init special periphery of your hardware  */
   system_init();                /* initialize system timer (not yet) and    */
                                 /* font list                                */
```

Wie schon in der Einführung erwähnt, benutzt die GUI–Bibliothek ein zweistufiges Konzept, um auf Hardware–Komponenten zuzugreifen. Die direkte Steuerung erfolgt durch eine hardwarenahe Schicht (*device*), auf die über eine übergeordnete Ebene (*channels*) zugegriffen wird. Dies hat zum einen den Vorteil, daß ein Channel in der Regel leichter handhabbar ist und zum anderen ein Channel auch mit einem anderen Device (beispielsweise einem Drucker anstatt eines LCDs) verbunden werden kann und die Ausgabe immer noch identisch ist. Dies wird in dem Beispiel–Programm durch folgende Zeilen bewerkstelligt:

```
   /* define a screen with your systems resolution                          */
   lcd_init( _LCD_PHYS_SIZE_X, _LCD_PHYS_SIZE_Y, LCD_NORMAL, &system_lcd);

   disp_open( &display);         /* generate the display and connect it to   */
   disp_make_current( &display); /* lcd-driver, which controlls the          */
   disp_connect( system_lcd);    /* hardware                                 */
```

Anschließend wählt man den Font und dessen Darstellungsparameter. In diesem Fall bedeuten DRAW_SET und TXT_NORMAL, daß der Fonts in der aktuellen Farbe gezeichnet und normal dargestellt wird.

```
disp_set_font( __Mono8x16);          /* desired font for text output on   */
disp_set_drawmode( DRAW_SET);        /* the display                       */
disp_set_textstyle( TXT_NORMAL);
```

Der gewünschte Text kann nun durch Positionierung des Zeichen–Cursors auf dem LCD gezeichnet werden. (Dieses und die folgenden Beispielprogramme versuchen zwar möglichst keine Voraussetzungen hinsichtlich der Auflösung Ihres LCDs zu machen, jedoch läßt sich dies manchmal nicht vermeiden (hier: 320×240 Pixel). Beachten Sie deshalb die absoluten Positionsangaben der Zeichenbefehle.)

```
disp_set_cursor_position( 100, 120); /* print some text at the defined    */
disp_text( "Hello World!");          /* position                          */
disp_set_cursor_position( 100, 100);
disp_text( "Hello User!");
```

Im allgemeinen ist zu beachten, daß der Ursprung des Koordinatensystems der GUI–Bibliothek sich in der linken, unteren Ecke befindet und die Zeichenebene nach oben und rechts aufgespannt wird. Nach getaner Arbeit können die benutzten Ressourcen wieder freigegeben werden. Diese drei Befehle wurden hier auskommentiert.

```
disp_disconnect();
disp_close( &display);
lcd_close();
```

Ihr System kann sich nun anderweitig beschäftigen.

```
while (1) {}; /* do something ... */
```

2.2.1 Linien und Kreise

Das Programm aus dem vorangegangenen Abschnitt soll anstelle des Textes einige Kreise und Linien zeichnen. Das Projekt dieses Programms finden Sie auf der CD. Wenn Sie sich den Programmcode von “main.c” anschauen, werden Sie feststellen, daß im wesentlichen die folgenden Zeilen zum ursprünglichen “Hello World” neu sind.

```
/* draw some lines                                                        */
disp_set_color( BLACK);
disp_set_drawmode( DRAW_EXOR);
for (i = 0; i < _LCD_PHYS_SIZE_X; i++)
{
   disp_set_cursor_position( 0, 0);
   disp_line( i, _LCD_PHYS_SIZE_Y - 1);
}/* for */
for (i = 0; i < _LCD_PHYS_SIZE_Y; i++)
{
   disp_set_cursor_position( 0, 0);
   disp_line( _LCD_PHYS_SIZE_X - 1, i);
}/* for */
/* draw some circles                                                      */
disp_set_color( WHITE);
disp_set_drawmode( DRAW_SET);
disp_set_cursor_position( _LCD_PHYS_SIZE_X/2, _LCD_PHYS_SIZE_Y/2);
for (i = 1; i <= 100; i = i + 5)
{
   disp_circle( i);
   disp_circle( i + 1);
   disp_circle( i + 2);
}/* for */
```

Mit `disp_set_color()` kann die Zeichenfarbe eingestellt werden (dies wäre im ersten Fall nicht nötig gewesen, da standardmäßig BLACK angewählt ist). Ferner wird ein neuer Zeichenmodus eingeführt: EXOR. Bei allen darauffolgenden Zeichenoperationen werden die zu setzenden Punkt mit dem schon vorhandenen Bildinhalt exklusiv–oder verknüpft. Um eine Linie zu zeichnen müssen Sie zunächst den Grafikcursor auf den Startpunkt bewegen `disp_set_cursor_position()` und dann mit `disp_line()` die Linie zum Endpunkt durchziehen lassen. Das Darstellen von Kreisen erfolgt analog: zunächst muß der Cursor zum Mittelpunkt bewegt werden und anschließend der Kreis mit dem entsprechenden Radius gezeichnet werden `disp_circle()`.

2.2.2 Bitmaps und Sprites

Das ursprüngliche “Hello World”–Programm wird nun so erweitert, daß eine Bitmap auf dem Display ausgegeben werden, außerdem soll sich eine Zeiger (Sprite) über das Bild bewegen. Das zu diesem Projekt gehörende Programm befindet sich auf der CD.
Die Darstellung eines Bildes im bmp–Format ist denkbar einfach. Zunächst setzt man den Grafikcursor auf die linke, untere Ecke des Bildes und gibt anschließend die Bitmap mit dem Befehl `disp_bmp()` aus. Hierbei wird lediglich ein Zeiger auf die Bilddaten benötigt. In diesem Beispiel wurde das Bild zunächst Hex–Zahlen umgewandelt, die dann in ein Feld abgelegt wurden, diese wird über die Datei simplifylogo.h eingebunden. Im Vergleich zum “Hello World”–Programm werden anstelle der Textausgabebefehle folgende Kommandos benötigt.

```
disp_set_cursor_position( 55, 80); /* set the bitmap of the logo          */
disp_bmp( simplify_logo);          /* on the LCD                          */
```

Bei einem Sprite handelt es sich um ein kleines Bild mit einer Auflösung von 16×16 Punkten, welches über den eigentlichen Bildschirminhalt plaziert bzw. bewegt werden kann, ohne diesen zu verändern. In dem Beispielprogramm werden die Daten des Sprites über ein Feld eingelesen. Dieses Feld enthält neben den Bildinformationen auch noch eine ebenfalls 16×16 Pixel große Maske, die festlegt welche Teile des Sprites überhaupt ausgegeben werden.

```
/* pattern for sprite cursor with its respective mask pattern */
fillstyle arrow_sprite[32] =
{
   0x0600, 0x0900, 0x0900, 0x9200, 0xD200, 0xA400, 0x8400, 0x8780,
   0x8100, 0x8200, 0x8400, 0x8800, 0x9000, 0xA000, 0xC000, 0x8000,
   0x0600, 0x0F00, 0x0F00, 0x9E00, 0xDE00, 0xEC00, 0xFC00, 0xFF80,
   0xFF00, 0xFE00, 0xFC00, 0xF800, 0xF000, 0xE000, 0xC000, 0x8000
};
```

Mit `disp_set_sprite_pattern()` werden die Daten des Sprites übergeben und die Koordinaten des “hot spots”festgelegt. Bei Zeichenoperationen wird das Sprite relativ zu seinem “hot spot ”ausgegeben. Vor dem ersten Einschalten des Sprites mit `disp_sprite_on()` sollten Sie durch `disp_set_sprite_position()` die Zeichenkoordinaten festlegen. Mit dem Befehl `disp_sprite_on()` können Sie gleichzeitig den dazugehörigen Zeichenmodus wählen. Das Ausschalten des Sprites erfolgt analog mit `disp_sprite_off()`).

```
disp_set_sprite_pattern( arrow_sprite, 0, 15); /* activate sprite         */
disp_set_sprite_position( 16, _LCD_PHYS_SIZE_Y/2);
disp_sprite_on( DRAW_EXOR);
```

Die folgenden Programmzeilen sorgen dafür, daß der Spritezeiger ständig hin und her über die zuvor ausgegebene Bitmap bewegt wird. `system_wait_ticks()` dient dazu die Bewegung zu verlangsamen. Denken Sie daran diese Warteprozedur in der Datei config_gui.h entsprechend zu kalibrieren (vgl. Abschnitt 2.1.2).

Anstelle von `system_wait_ticks()` kann auch ein beliebiger “delay loop” verwendet werden.

```
system_wait_ticks( 100);
while (1)
{
   for (i = 16; i < _LCD_PHYS_SIZE_X - 16; i++)
   {
      disp_set_sprite_position( i, _LCD_PHYS_SIZE_Y/2);
      system_wait_ticks( 10);
   }/* for */
   for (i = _LCD_PHYS_SIZE_X - 16; i > 16; i--)
   {
      disp_set_sprite_position( i, _LCD_PHYS_SIZE_Y/2);
      system_wait_ticks( 10);
   }/* for */
}/* while */
```

3 Systemtimer

Häufig wird ein automatisch blinkender Cursor gewünscht. Dieser kann durch die GUI–Bibliothek mit Hilfe der Sprites bereitgestellt werden, wenn Ihre Hardware zusätzlich einen Systemtimer zur Verfügung stellt. Der Systemtimer wird ferner zum Auslesen eines Touch–Displays benötigt.

3.1 Treiberfunktionen für den Systemtimer

Für den Systemtimer müssen Sie einige hardwareabhängige Funktionen erstellen. Am wichtigsten ist eine Interrupt-Routine, die über einen Timer `TICKS_PER_SECOND` mal pro Sekunde aufgerufen wird und die entsprechenden Zeitvariablen `TIMERDATA` und `DAYDATA` hochzählt (vgl. Abbildung 1). `TIMERDATA` und `DAYDATA` müssen dabei globale 32-Bit-Variablen sein, auf die die GUI-Bibliothek zugreifen kann, wenn Sie eine Zeitinformation benötigt. Implementieren Sie diese Funktion angepaßt an den von Ihnen verwendeten Microcontroller. Damit in diesem Fall `TIMERDATA` nicht versehentlich vor dem planmäßigen übertrag nach `DAYDATA` überläuft, darf der Systemtimer nicht mit einer höheren Frequenz als 49710 Hz betrieben werden (d.h. `TICKS_PER_SECOND` muß auch entsprechend kleiner als 49710 sein). In der Regel reicht es aus, den Timer–Interrupt 50 odr 100 mal pro Sekunde ausführen zu lassen.

SYSTEMTIMERFUNC — Zu implementierende Interrupt–Routine des Systemtimers

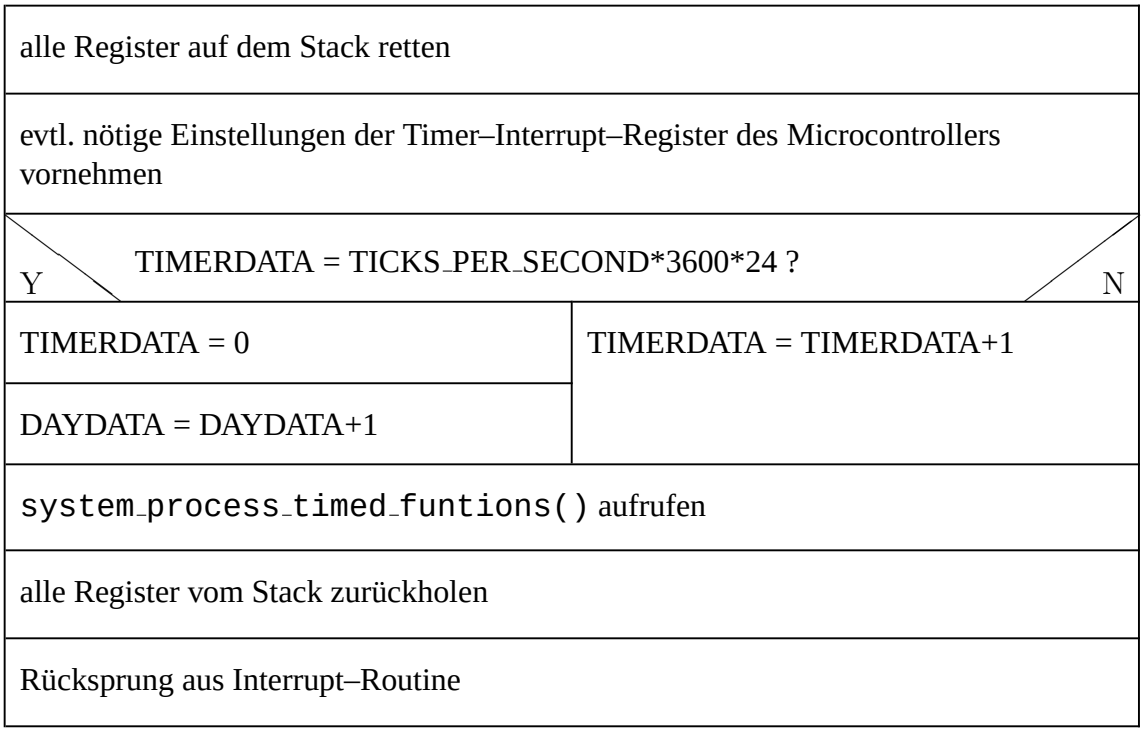


Abbildung 1: Struktogramm der Timer–Interrupt–Routine

os_flag_block() — Zu implementierende Funktion, die Timer–Interrupt-Semaphoren setzt

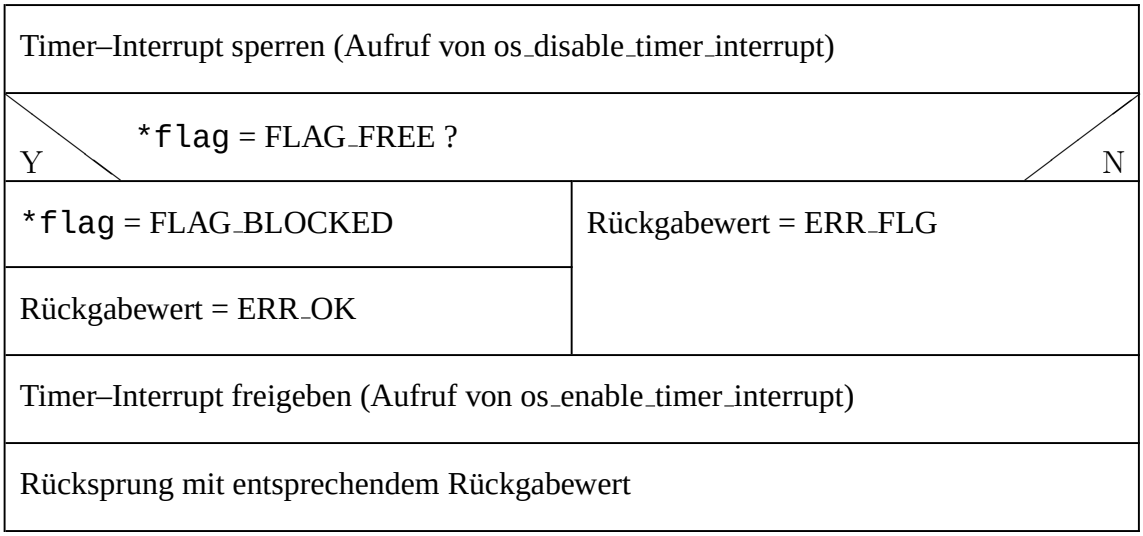


Abbildung 2: Struktogramm von `os_flag_block(uint8 *flag)`

Die Funktion `system_init()` (aus system.c) setzt die beiden Zählvariablen `TIMERDATA` und `DAYDATA` auf 0 und startet den Systemtimer über `os_systemtimer_start()`. Die Funktion `os_systemtimer_start()` muß von Ihnen so implementiert werden, daß der für den Systemtimer zuständige Timer–Interrupt korrekt gestartet wird.
Am Ende der Timer–Interrupt–Routine rufen Sie die bereits in der Bibliothek vorhandene Funktion `system_process_timed_functions()` auf. In dieser Prozedur werden über eine Zeigerliste andere Funktionen aufgerufen, die bei jedem Timer–Interrupt zur Ausführung kommen sollen. Aufgabe dieser Funktionen ist die Steuerung periodischer Vorgänge, wie z. B. das eingangs erwähnte Cursorblinken. Da diese Liste der *"timed"*–Funktionen während der Programmlaufzeit geändert werden kann, treten Probleme auf, wenn ein Timer–Interrupt mitten in einer solchen Änderung erfolgt: Die Zeiger der Liste sind dann u. U. ungültig.
Dieses Problem wird durch die im folgenden beschriebenen Funktionen `os_flag_block()` und `os_flag_free()` gelöst, die den Timer–Interrupt kontrolliert sperren und wieder freigeben und dies mit zwei einfachen *8–bit–Semaphoren* signalisieren, die die Werte `FLAG_BLOCKED` und `FLAG_FREE` (beide definiert in os_gui.h) annehmen können. Wie Sie der Beschreibung von `os_flag_block()` entnehmen können, wird der Timer–Interrupt nicht wirklich für eine längere Zeit gesperrt, sondern lediglich ein `flag` entsprechend gesetzt. Die kritischen Routinen der GUI–Bibliothek richten sich dann nach diesem, um Konflikte mit den *timed*–Funktionen zu vermeiden.
Implementieren Sie also die beiden Funktionen `os_flag_block()` und `os_flag_free()` für Ihr System, wie hier beschrieben.

err_code os_flag_block(uint8 *flag): Diese Funktion (siehe Abbildung 2) muß den Timer-Interrupt sperren (durch Aufruf von `os_disable_timer_interrupt()` (s. u.)). Anschließend überprüft sie das angegebne Flag und setzt es auf `FLAG_BLOCKED`, wenn es vorher `FLAG_FREE` war. Anschließend wird der gesperrte In-

terrupt wieder freigegeben (durch Aufruf von `os_enable_timer_interrupt()` (s. u.)). Rückgabewerte: `ERR_OK`, wenn das Flag noch frei war und erfolgreich gesperrt werden konnte, `ERR_FLG`, wenn es bereits gesperrt war.

err_code os_flag_free(uint8 *flag): Diese Funktion setzt das Flag auf `FLAG_FREE`, und gibt somit die gesperrte Resource wieder frei. Das Auslesen des Flags selbst darf nicht durch andere Interrupts gestört werden, Wenn dies bei Ihrem System passieren kann, müssen auch hier die betreffenden Interrupts voher deaktiviert und nach dem Lesen des Flags wieder aktivert werden.

`os_flag_free()` wird und darf nur benutzt werden, wenn die aufrufende Routine das Flag auch vorher für sich gesperrt hat (also insofern das "Zugriffsrecht" am Flag hat). Rückgabewert der Funktion: `ERR_OK`.

Die folgenden Funktionen müssen ein Ein- und Ausschalten des Timer-Interrupts erlauben. Sie werden zum sicheren Verwalten der Semaphoren und für interne Verwaltungsarbeiten der Funktion `system_process_timed_functions()` benötigt.

void os_disable_timer_interrupt(void): Sperrt den Systemtimer-Interrupt.

void os_enable_timer_interrupt(void): Gibt den Systemtimer-Interrupt wieder frei.

Um Wartezeiten eventuell für andere Aufgaben nutzen zu können, wird folgende Funktion benötigt:

void os_process(void): Diese Funktion wird während der `system_wait`-Funktionen aufgerufen, um eventuell dem System zu ermöglichen während der Wartezeit andere Aufgaben zu erfüllen. Die Deklaration und Implementierung dieser Funktion kann in "main.h" bzw. "main.c" erfolgen. Wenn dies nicht nötig ist, können Sie in der Datei "config_gui.h" die Definition von `_OS_USED` weglassen (siehe nächsten Abschnitt) und auf die Implementierung dieser Funktion als Funktionsrumpf verzichten.

Die Deklarationen dieser Funktionen und von Konstanten, die von diesen Funktionen verwendet werden, befinden sich in der Datei "os_gui.h". Die Implementierung der Funktionen sollte in Assembler erfolgen, um eine vollständige Kontrolle des Interrupt–Handlings zu gewährleisten.

3.2 Konfiguration der GUI–Bibliothek

In der Datei "config_gui.h"müssen Sie die im Quelltextauszug erwähnten `#define`–Zeilen in den entsprechenden Abschnitten **zusätzlich** aktivieren (die Kommentare wurden hier teilweise gekürzt). Die Option `__DISABLE_TIMER_INTERRUPT_NEEDED` ist bei den meisten Micorcontrollern nötig, da hier mit einer einzigen Assembleranweisung das manipulieren der Zeigerliste der aufzurufenden Funktionen nicht möglich ist. Die letzte Option `_SPRITE_BLINK_WANTED` sorgt dafür, daß an die Liste der "Timer"–Funktionen die Steuerung des Sprite–Blinkens angehängt wird.

```
/***************************************************************************/
/* general customization                                                   */
/***************************************************************************/

/* Using the library only for displaying information on a lcd display
is not time dependent and therefore does not require you to implement a
ßystem time". However a ßystem time" is required if you want to use the
following functionallity which inherently must distinguish different times:
- Touch glass functions (and therefore also buttons and touch keyboards).
- Automatical blinking of sprite cursors
...
*/

#define _SYSTEM_TIMER_PRESENT

/* if _SYSTEM_TIMER_PRESENT is defined functions for reading touch glasses and
blinking sprite cursors may be (dependend on the configuration) automatically
inserted into the system timer handler.
The system timer must find valid data at each node of the list which handles the
functions to be called periodically when the
timer interrupt occurs (otherwise the system crashes). Thus the process of
inserting or removing the pointer to a function to be triggered by the system
timer interrupt must happen so that writing this information cannot be
corrupted by system timer processing the unfinnished list of timed functions
To ensure this there are 2 ways:
1. If the processor can write the pointers in one machine instruction nothing
   special is required as the system timer interrupt routine will always find
   consistent data
2. If (mostly on smaller controllers) the pointer cannot be written in one
   machine instruction, writing it could be interrupted resulting in
   inconsistent data. To avoid this it then is required to disable the timer
   interrupt during this operation. This is realized by calling the functions
   os_disable_timer_interrupt and os_enable_timer_interrupt, which are included
   into the code of the functions system_timed_function_on and
   system_timed_function_off (in the file system.c) if
   __DISABLE_TIMER_INTERRUPT_NEEDED is defined.
NOTE: This consideration is only relevant if system timer functions are
needed */

#define __DISABLE_TIMER_INTERRUPT_NEEDED


/* Define the following, if your system features dynamic memory
allocation as required by ANSI-C. */

#define _MALLOC_AVAILABLE


/* If you do not provide dynamic memory allocation, a simple memory
allocator is provided, which can be enhanced if neccessary.
This memory allocater uses a static heap, which size can be configured with
_STORAGE_ALLOCATER_HEAP_SIZE (in bytes).
A list of free memory areas is needed as well, to organize the heap. This
list has a maximum number of entries (_MAX_FREE_LIST_ENTRIES). */

#define _STORAGE_ALLOCATOR_HEAP_SIZE 5000
#define _MAX_FREE_LIST_ENTRIES       30


/* if an operating system is used and wants to gain control during wait
functions (in this case it is called by os_process, which you must provide),
define: */

/*
#define _OS_USED
*/
```

```
/*****************************************************************************/
/* customizing the display output                                            */
/*****************************************************************************/

/* define, if you want to have an automatically blinking sprite cursor */

#define _SPRITE_BLINK_WANTED
```

3.3 Beispiel mit blinkendem Sprite

Mit der richtigen Datei “config_gui.h”müssen Sie lediglich eine weitere Zeile zu dem letzten Beispiel aus Abschnitt 2.2 hinzufügen, damit das Sprite blinkt.

```
   disp_set_sprite_blink_interval( 50);           /* now blinking          */
```

Insgesamt hat der Quelltext folgendes Aussehen. Auf die Bewegung des Sprites wird nun verzichtet, damit das Blinken besser erkennbar ist. Beachten Sie, daß der Systemtimer mit dem Aufruf von `system_init()` gestartet werden sollte. Das Beispiel befindet sich auf der CD.

```
/*****************************************************************************/
/* main.c      bitmap and blinking sprite demo                               */
/*                                                                           */
/* (C) SIMPLIFY TECHNOLOGIES GmbH, www.simplify-technologies.de              */
/*                                                                           */
/* last change: 7 Mar 2006                                                   */
/*****************************************************************************/

#include "main.h"               /* includes for this file                    */
#include "platform_periphery.h" /* includes for special features of the      */
                                /* hardware                                  */

#include "portab.h"       /* includes for OS functions and defs              */
#include "errors.h"       /* error codes                                     */
#include "system.h"       /* includes for the "kernel"                       */
#include "lcd.h"          /* includes lcd-driver                             */
#include "display.h"      /* includes for graphical functions                */
#include "simplifylogo.h" /* data for the logo                               */

/* pattern for sprite cursor with its respective mask pattern                */
fillstyle arrow_sprite[32] =
{
   0x0600, 0x0900, 0x0900, 0x9200, 0xD200, 0xA400, 0x8400, 0x8780,
   0x8100, 0x8200, 0x8400, 0x8800, 0x9000, 0xA000, 0xC000, 0x8000,
   0x0600, 0x0F00, 0x0F00, 0x9E00, 0xDE00, 0xEC00, 0xFC00, 0xFF80,
   0xFF00, 0xFE00, 0xFC00, 0xF800, 0xF000, 0xE000, 0xC000, 0x8000
};


int main( void)
{
   lcd_device      *system_lcd; /* pointer to lcd driver                     */
   display_channel  display;    /* data structure for the drawing functions  */

   platform_periphery_init();   /* init special periphery of your hardware   */
   system_init();               /* initialize system timer and font list     */

   /* define a screen with your systems resolution                           */
   lcd_init( _LCD_PHYS_SIZE_X, _LCD_PHYS_SIZE_Y, LCD_NORMAL, &system_lcd);

   disp_open( &display);         /* generate the display and connect it to   */
   disp_make_current( &display); /* lcd-driver, which controlls the          */
   disp_connect( system_lcd);    /* hardware                                 */

   disp_set_cursor_position( 55, 80); /* set the bitmap of the logo          */
```

```
   disp_bmp( simplify_logo);          /* on the LCD                          */

   disp_set_sprite_pattern( arrow_sprite, 0, 15); /* activate sprite        */
   disp_set_sprite_position( 50, 50);
   disp_sprite_on( DRAW_EXOR);

   disp_set_sprite_blink_interval( 50);           /* now blinking           */

   while (1)
   {
      system_wait_ticks( 10);
   }/* while */

   /* disp_sprite_off(); */
   /* other clean up ... but never reached */

   return (ERR_OK);
}/* main */
```

4 Touch–Funktionalität

Häufig werden LCD–Anzeigen mit einem Touch–Glas ausgestattet, so daß der Benutzer über das Display gleichzeitig Eingaben vornehmen kann. Die GUI–Bibliothek unterstützt diese Funktion, allerdings muß das System dann über einen Timer verfügen und die entsprechenden Anpassungen des Kapitel 3 müssen durchgeführt worden sein.

4.1 Treiberfunktionen zum Ansteuern des Touch–Glases

Zur Verwendung eines Touch–Glases für die Benutzereingaben müssen Sie zwei elementare Funktionen bereitstellen, die abhängig von der von Ihnen verwendeten Hardware sind. Dies ist einmal eine Funktion zur Initialisierung Ihrer Touch–Hardware und eine Funktion zum Auslesen des Touch–Glases in willkürlichen Einheiten (z. B. den aus einem A/D-Wandler ausgelesenen Werten). Beide sind in "tglass_.h" deklariert:

void c_tglass_ground_state(void): Diese Funktion initialisiert die Touch–Glas–Hardware und versetzt es in den Ruhezustand vor einer Auslese-Sequenz (d. h. z. B. für Folien–Touch–Gläser eine entsprechende Einstellung der Ruhepotentiale an den Folien).

void c_tglass_get(uint16 *ptr): Auslesen der Koordinaten–Werte für das Touch–Glas. Hier müssen Sie noch nicht kalibrierte Koordinaten liefern, sondern *rohe* Werte zur Repräsentation der Position, z. B. die Spannungswerte eines AD–Wandlers, die für x– bzw. y–Koordinaten gewonnen werden. Die Umwandlung in Pixel-Koordinaten wird später automatisch und nur bei Bedarf vorgenommen. Die gewonnenen x– und y–Werte müssen dann in ein Array aus zwei 16-Bit Werten abgespeichert werden, auf das der Pointer `ptr` zeigt. Dabei ist zuerst der y–Wert, **dahinter** der x–Wert abzuspeichern !!

Damit die Berechnung der Kalibrierkoeffizienten mittels Integer–Arithmetik korrekt funktioniert, gibt es eine weitere Anforderung an die von dieser Funktion gelieferten Werte: Sie müssen eine Auflösung haben, die rein numerisch mindestens genauso hoch ist, wie die Auflösung des unter dem Touchglas befindlichen Displays und sich linear zur geometrischen Position der Berührung verändern. Wenn Ihre Hardware eine reale Auflösung liefert, die niedriger ist, müssen die Werte entsprechend *hochskaliert* werden.

Wenn das Touch-Glas nicht gedrückt ist wird der Wert `_AD_TOUCH_NOT_PRESSED` nach *ptr gespeichert. Liegen ungültige Koordinaten vor, so wird die Konstante `_AD_TOUCH_NOT_VALID` an `*ptr` abgelegt. Beide Konstanten sind in "config_hardware_gui.h" (siehe Abschnitt 4.2) als ein 16-Bit-Werte definiert, die ansonsten von der Hardware nicht zurückgeliefert werden.

Da diese Funktion typischerweise periodisch über den Systemtimer–Interrupt abgearbeitet wird, sollte hier auf eine niedrige Laufzeit geachtet werden.

Bemerkung: Bei Touch-Displays mit leitenden Touch–Folien spielt die RC–Zeit dieser Folienanordnung eine wichtige Rolle. Um zuverlässige Koordinatenwerte zu erhalten, muß sich die Spannung an den Folien ausreichend stabilisiert haben. Die

Funktion `tglass_read()` (in “tglass.c”), die `c_tglass_get()` aufruft, sorgt dafür, daß nur Berührungen, die bereits seit einem Intervall des Systemtimers bestehen als gültig ausgewertet werden. Es steht somit die Zeit eines Intervalls des Systemtimers für die *Aufladung* der Folien auf das stabile Endpotential zur Verfügung.

4.2 Konfiguration der GUI–Bibliothek

4.2.1 config_hardware_gui.h

Aktivieren Sie in der Datei “config_hardware_gui.h” **zusätzlich** folgende `#define`–Anweisungen.

```
/*****************************************************************************/
/* defines for touch glass                                                   */
/*****************************************************************************/
#define _TGLASS_PRESENT        1    /* set to 1 if touch glass is present */
#define _AD_TOUCH_NOT_VALID   8192  /* definition for an A/D converter result
                                       considered illegal */
#define _AD_TOUCH_NOT_PRESSED 4096  /* definition for touch glass not pressed */

/* define how the orientation of the x- and y- axis of the physical interface of
the touch glass is aligned with the coordinate system of the lcd display when
this is used in LCD_NORMAL orientation. */
#define _TOUCH_NORMAL
/* #define _TOUCH_ROTATED */

/* now for the Sheng display */
#define _TGLASS_A_DEFAULT_DATA    23937 /* Parameters for calibration */
#define _TGLASS_B_DEFAULT_DATA      -29 /* and conversion to pixel */
#define _TGLASS_C_DEFAULT_DATA    18322 /* coordinates */
#define _TGLASS_D_DEFAULT_DATA      -27
/* These parameters are strongly dependend on the hardware and touch glas
driver implementation.
If _TGLASS_PRESENT is not set 1 the whole tglass software module tglass.c
is ommited.
Then the following modules which depend on the existance of a touch glas are
also ommited: touch.c (the module for the touch channels), t_keyb.c (the
module for the touch keyboard devices) and keyboard.c (the module for the
keyboard channels) */

/* If you want to use the calibration function for the touch glas you need to
1. define the following coordinates as positions for crosses to appear on
   the display for calibration so they will fit on the display.
2. Adjust the function tglass_calibrate in tglass.c so the text output fits
   your display. Text strings for the output can be adjusted in text_gui.h */
#define KALIB_X1    32  /* Coordinates for the crosses marking the calibrion */
#define KALIB_Y1    224 /* points for the touch glas calibration */
#define KALIB_X2    288
#define KALIB_Y2    16
#define KALIB_X3    32
#define KALIB_Y3    16
#define KALIB_X4    288
#define KALIB_Y4    224
```

Zunächst wird mit `_TGLASS_PRESENT` die Touch–Unterstützung aktiviert. Die Konstanten `_AD_TOUCH_NOT_PRESSED` und `_AD_TOUCH_NOT_VALID` wurde bereits im Abschnitt 4.1 besprochen.
Bitte setzen Sie ” #define _TOUCH_NORMAL”, wenn die Orientierung der hardwaremäßigen Anschlüsse des Touch-Glases mit den Koordinatenachsen des LCD-Displays

übereinstimmt, wenn dieses in der Orientierung LCD_NORMAL (ohne Rotation, diese kann nur für bestimmte LCD-Controller erfolgen) betrieben wird. (Dann zeigen also x- und y- Achse von Touch-Glas und LCD-Display in dieselbe Richtung). Diese Einstellung ist für die meisten Aufbauten die richtige. Wenn der Anschluß des Touch-Glases so realisiert ist, daß dessen Koordinatensystem senkrecht zu dem des LCD-Displays steht, setzen Sie bitte "_TOUCH_ROTATED". Es darf selbstverständlich nur einer der beiden Definitionen gesetzt werden. Die folgenden vier Parameter `_TGLASS_A_DEFAULT_DATA bis _TGLASS_D_DEFAULT_DATA` hängen sehr stark von der verwendeten Touch–Hardware ab und werden durch die im folgenden Abschnitt 4.3 besprochenen Kalibration ermittelt. Mit diesen Werten erfolgt die Umwandlung der *Touch–Meßwerte* in Pixel–Koordinaten. Wollen Sie für die Kalibration die zur Verfügung gestellte Routine verwenden, müssen sie außerdem vier Pixel–Koordinatenpaare definieren an denen *Touch–Meßwerte* eingelesen werden sollen. Dies geschieht mit den Konstanten `KALIB_X1` bis `KALIB_Y4`. Die Koordinaten müssen Sie natürlich an die Auflösung der Display–Hardware anpassen.

4.2.2 config_gui.h

Auch in der Datei "config_gui.h" müssen Sie wieder einige `#define`–Optionen **zusätzlich** bereitstellen. Neben dem obligatorischen `_TOUCH_WANTED`, benötigen Sie ferner `_TGLASS_CALIBRATION_WANTED` falls die GUI–Bibliothek Code zur Kalibrierung der Touch–Hardware beinhalten soll. Haben Sie einmal die Kalibrierung durchgeführt und die Parameter in "config_hardware_gui.h" eingetragen, dann werden diese automatisch benutzt.
Die Optionen zur Anwahl der vorprogrammierten GUI–Elemente *Button* und *Keyboard*, `_BUTTONS_WANTED` bzw. `_TOUCH_KEYBOARDS_WANTED`, können vorerst ignoriert werden.

```
/*****************************************************************************/
/* customizing touch glass functions                                         */
/*****************************************************************************/

/* if you want touch functionallity which is also the prerequisite for buttons
and touch keyboards activate the following define: */

#define _TOUCH_WANTED


/* if you want to include code for calibrating the touch glass activate the
following define. Please note, that you probably need to customize the function
tglass_calibrate in tglass.c to nicely fit your touch display */

#define _TGLASS_CALIBRATION_WANTED


/* define the following if you want to make use of touch buttons */

/*
#define _BUTTONS_WANTED
*/


/* define the following if you want to use touch keyboards */

/*
#define _TOUCH_KEYBOARDS_WANTED
```

```
*/


/* define the following if you want to use menus, sliders etc. */

/*
#define _TOUCH_ADDONS_WANTED
*/
```

4.3 Kalibrierung des Touch–Glases

Die Kalibrierkonstanten des Touchglases, **_TGLASS_A_DEFAULT_DATA** bis **_TGLASS_-_DEFAULT_DATA**, hängen von der Hardware ab und müssen experimentell (z. B. durch Verwendung der Kalibrierfunktion `tglass_calibration_sequence()` in der Datei "tglass.h/c") bestimmt werden. Sie sind die Konstanten für die Berechnung der Koordinaten aus den (Spannungs–) Werten, die vom Hardwaretreiber geliefert werden. Es wird ein linearer Zusammenhang der Form $x = A * U_x + B$ zwischen diesen Werten (z. B. Spannungen vom AD-Wandler U_x, U_y) und den Koordinaten angenommen.
Voraussetzung der Kalibrierung mittels der Funktion `tglass_calibration_sequence()` ist, daß das LCD-Display nicht softwaremäßig rotiert ist, sondern in der Orientierung `LCD_NORMAL` betrieben wird.
Die Kalibrierfunktion `tglass_calibration_sequence()` präsentiert vier Fadenkreuze auf dem LCD–Display und fordert den Anwender auf, diese zu berühren, damit die Kalibrierkonstanten berechnet werden können. Danach werden nochmals zwei Fadenkreuze dargestellt, die zu drücken sind um die errechneten werden zu überprüft. Wenn diese Funktion verwendet werden soll, müssen die gewünschten Koordinaten **KALIB_X1** bis **KALIB_Y4** für die Positionen der Fadenkreuze definiert werden. Sinnvoll sind typischerweise Positionen etwas innerhalb der vier Ecken des LCDs.
Das folgende Programm demonstriert die Benutzung der Kalibrierfunktion; Sie finden es auf der CD. Wichtig ist hierbei, daß der aktuelle *Display–Channel* dem LCD entspricht, welches unter dem Touch–Glas liegt. Ansonsten wird hier wieder das zweistufige Konzept zur Ansteuerung des Touch-Glases verwendet. Die hardwarenahe Ebene entspricht hier dem `tglass_device`, mit dem ein `touch_channel` verbunden wird (siehe Quelltext). Dadurch wird es möglich auch während der Programmlaufzeit den Touch–Channel z. B. mit einer Maus oder einem anderen Eingabegerät zu verbinden.
Analog zum Abschnitt 2.2 erfolgt das Einrichten des Touch–Screens mit

```
tglass_init( 1, &system_tglas);            /* init touch glas hardware   */
touch_open( &touch_glas, &display, NULL);  /* open touch channel         */
touch_connect( &touch_glas, system_tglas); /* connect channel & hardware */
```

und das spätere Aufräumen durch

```
/* cleanup touch and tglass */
touch_disconnect( &touch_glas);
touch_close( &touch_glas);
tglass_close();
```

Der erste Parameter bei `tglass_init()` gibt die Anzahl der *System-Ticks* vor, die zwischen zwei Abfragen des Touch–Glases liegen sollen. Die Prozedur `touch_open()` erlaubt es ferner auch eine akustische Rückkopplung mit anzusteuern. Hiervon wird aber kein Gebrauch gemacht.

Die Kalibrierroutine gibt bei Ihrem Aufruf noch einen Erklärungstext auf dem Display aus. Die Position des Textes wurde auf ein 320×240 Pixel Display ausgelegt und muß bei kleineren Anzeigen im Quellcode der GUI–Bibliothek geändert werden (Dateien ”tglass.c” und ”text_gui.h”). Nach dem Aufruf von `tglass_calibrate()` werden die gewonnenen Koeffizienten auf dem LCD-Display ausgegeben. Setzen Sie hier einen Breakpoint mit Ihrem Debugger (oder fügen Sie eine Warteschleife ein) und lesen Sie die ermittelten Kalibrierungswerte aus. Die ermittelten Daten weisen Sie den Konstanten _TGLASS_A_DEFAULT_DATA bis _TGLASS_D_DEFAULT_DATA in der Datei “config_hardware_gui.h” zu, damit sie fortan verwendet werden. Zum Schluß des Beispiels werden die verwendeten ”devices” und ”channels” wieder geschlossen.

```
/*****************************************************************************/
/* main.c      calibration of the touch glass                                */
/*                                                                           */
/* (C) SIMPLIFY TECHNOLOGIES GmbH, www.simplify-technologies.de              */
/*                                                                           */
/* last change: 7 Mar 2006                                                   */
/*****************************************************************************/

#include "main.h"               /* includes for this file */
#include "platform_periphery.h" /* includes for special features of the
                                   hardware */

#include "portab.h"      /* includes for operating system functions and defs */
#include "errors.h"      /* error codes */
#include "system.h"      /* includes for the "kernel" */
#include "lcd.h"         /* includes lcd-driver */
#include "display.h"     /* includes for graphical functions */
#include "tglass.h"      /* includes tglass-driver */
#include "touch.h"       /* includes for touch functions */
#include "text_gui.h"    /* defines for calibration explanation */
#include "mono6x8.h"     /* data of a font */

#include <stdio.h>       /* includes standard I/O from ANSI C */


int main( void)
{
   lcd_device      *system_lcd;       /* pointer to lcd driver */
   display_channel display;           /* data structure for the drawing
                                         functions */
   tglass_device   *system_tglas;     /* pointer to tglass driver */
   touch_channel   touch_glas;        /* data structure for the touch
                                         functions */
   const font      *Mono6x8;          /* actual font */
   char            outputstring[50];  /* string for text output */
   uint8           calib_states;      /* current state ID of the calibration
                                         state machine */
   tglass_calib_statevars calib_vars; /* calibration state machine variables */
   err_code        err;               /* error result for the calibration
                                         process */

   platform_periphery_init();   /* init special periphery of your hardware */
   system_init();               /* initialize system timer and font list */

   /* define a screen with your systems resolution */
   lcd_init( _LCD_PHYS_SIZE_X, _LCD_PHYS_SIZE_Y, LCD_NORMAL, &system_lcd);
   disp_open( &display);         /* generate the display and connect it to */
   disp_make_current( &display); /* lcd-driver, which controlls the */
   disp_connect( system_lcd);    /* hardware */

   tglass_init( 1, &system_tglas);           /* init touch glas hardware */
   touch_open( &touch_glas, &display, NULL);  /* open touch channel */
   touch_connect( &touch_glas, system_tglas); /* connect channel & hardware */
```

```
   system_font_init( __Mono6x8);
   system_get_font_pointer("Mono6x8", &Mono6x8);
   disp_set_font( Mono6x8);   /* desired font for text output */

   /* calibration with text from "text_gui.h" */
   calib_states = 0;
   do
   {
      err = tglass_calibration_sequence(
         TEXT_CALIB, TEXT_CALIB_VERIFY, TEXT_CALIB_FAILED,
         10, 10, 80, &calib_vars, &calib_states);
   } while ( ERR_OK_FALSE == err);

   disp_clear();
   disp_set_cursor_position( 10, 40);
   if (ERR_OK == err)
   {
      sprintf( outputstring, "%s%i", "TGLASS_A = ", system_tglas->a);
      disp_text( outputstring);
      disp_set_cursor_position( 10, 30);
      sprintf( outputstring, "%s%i", "TGLASS_B = ", system_tglas->b);
      disp_text( outputstring);
      disp_set_cursor_position( 10, 20);
      sprintf( outputstring, "%s%i", "TGLASS_C = ", system_tglas->c);
      disp_text( outputstring);
      disp_set_cursor_position( 10, 10);
      sprintf( outputstring, "%s%i", "TGLASS_D = ", system_tglas->d);
      disp_text( outputstring);
   }/* if */
   else
   {
      disp_text( "Fatal error. No valid calibration parameters.");
   }/* else */

   /*
   INSERT BREAKPOINT HERE TO READOUT CALIBRATION DATA !!! OR USE
   ENDLESS WHILE LOOP TO STOP.
   */

   while (1) {}

   /* cleanup touch and tglass */
   touch_disconnect( &touch_glas);
   touch_close( &touch_glas);
   tglass_close();

   /* cleanup display and lcd */
   disp_disconnect();
   disp_close( &display);
   lcd_close();

   return( ERR_OK);
}/* main */
```

4.4 Touch–Beispiel

Das Programmbeispiel für Sprites aus Abschnitt 2.2.2 wird nun so verändert, daß der Zeiger nun über das Touch–Glas gesteuert und verschoben wird. An der Position des Sprites wird zusätzlich noch ein kleines Quadrat gezeichnet. Das Projekt dieses Programms finden Sie auf der CD.

```
/*****************************************************************************/
/* main.c      touch demo                                                    */
```

```
/*                                                                           */
/* (C) SIMPLIFY TECHNOLOGIES GmbH, www.simplify-technologies.de              */
/*                                                                           */
/* last change: 7 Mar 2006                                                   */
/*****************************************************************************/

#include "main.h"              /* includes for this file                     */
#include "platform_periphery.h" /* includes for special features of the      */
                               /* hardware                                   */

#include "portab.h"       /* includes for OS functions and defs              */
#include "errors.h"       /* error codes                                     */
#include "system.h"       /* includes for the "kernel"                       */
#include "lcd.h"          /* includes lcd-driver                             */
#include "display.h"      /* includes for graphical functions                */
#include "simplifylogo.h" /* data for the logo                               */
#include "tglass.h"       /* includes tglass-driver                          */
#include "touch.h"        /* includes for touch functions                    */


/* pattern for sprite cursors with their respective mask pattern */
fillstyle arrow_sprite[32] =
{
   0x0600, 0x0900, 0x0900, 0x9200, 0xD200, 0xA400, 0x8400, 0x8780,
   0x8100, 0x8200, 0x8400, 0x8800, 0x9000, 0xA000, 0xC000, 0x8000,
   0x0600, 0x0F00, 0x0F00, 0x9E00, 0xDE00, 0xEC00, 0xFC00, 0xFF80,
   0xFF00, 0xFE00, 0xFC00, 0xF800, 0xF000, 0xE000, 0xC000, 0x8000
};


int main( void)
{
   lcd_device      *system_lcd;   /* pointer to lcd driver                  */
   display_channel display;       /* data structure for the drawing funcs.  */
   tglass_device   *system_tglas; /* pointer to tglass driver               */
   touch_channel   touch_glas;    /* data structure for the touch funcs.    */

   int16 x, y;                    /* cursor position                        */
   uint32 ticks_end, days_end;    /* time stamp of touch contact            */


   platform_periphery_init();   /* init special periphery of your hardware  */
   system_init();               /* initialize system timer and font list    */

   /* define a screen with your systems resolution                          */
   lcd_init( _LCD_PHYS_SIZE_X, _LCD_PHYS_SIZE_Y, LCD_NORMAL, &system_lcd);

   disp_open( &display);         /* generate the display and connect it to  */
   disp_make_current( &display); /* lcd-driver, which controlls the         */
   disp_connect( system_lcd);    /* hardware                                */

   tglass_init( 1, &system_tglas);            /* init touch glas hardware   */
   touch_open( &touch_glas, &display, NULL);  /* open touch channel         */
   touch_connect( &touch_glas, system_tglas); /* connect channel & hardware */

   disp_set_cursor_position( 55, 80); /* set the bitmap of the logo         */
   disp_bmp( simplify_logo);          /* on the LCD                         */

   disp_set_sprite_pattern( arrow_sprite, 0, 15); /* activate sprite        */
   disp_set_sprite_position( _LCD_PHYS_SIZE_X/2, 50);
   disp_sprite_on( DRAW_EXOR);

   system_wait_ticks( 100);
   while (1)
   {
      /* read out touch glass */
      touch_get_contact_end( &touch_glas, &x, &y, &ticks_end, &days_end);
```

```
      /* set sprite at this position, if inside LCD */
      if ((x >= 0) && (x < _LCD_PHYS_SIZE_X) &&
         (y >= 0) && (y < _LCD_PHYS_SIZE_Y))
      {
         disp_set_cursor_position( x, y);
         disp_rectangle( 3, 3);
         disp_set_sprite_position( x, y);
      }/* if */
   }/* while */

   /* clean up is not necessary, because never reached */
   /*
   disp_sprite_off();

   touch_disconnect( &touch_glas);
   touch_close( &touch_glas);
   tglass_close();

   disp_disconnect();
   disp_close( &display);
   lcd_close();
   */

   return (ERR_OK);
}/* main */
```

Die Initialisierung des Touch–Glases erfolgt genauso wie im Kalibrierungsprogramm aus dem vorherigen Abschnitt. Die Positionierung der Bitmap und des Sprites entspricht dem Demo aus Abschnitt 2.2.2. Lediglich eine Funktion zum Auslesen der Touch–Position `touch_get_contact_end()` wird neu eingeführt. Diese ermittelt die Koordinaten und die Zeit, zu der der letzte Kontakt auf dem Touch–Channel endete.

5 Sonstiges

5.1 Lautsprecherunterstützung

Bei der Bedienung von Geräten mit Touch–Display ist es hilfreich für den Anwender, wenn er eine akustische Rückmeldung erhält sobald er mit dem Stift oder Finger irgendwelche Aktionen (z. B. Buttons drückt oder Touch–Tastaturen benutzt) ausführt. Hierfür stellt die GUI–Bibliothek einige Funktionen zur Verfügung (siehe auch die Funktion `touch_open()` im Abschnitt 4.3.

5.1.1 Konfiguration

In der Datei "config_hardware_gui.h" müssen Sie folgenden `#define`–Anweisungen aktivieren, um die Tonfunktionen zu nutzen:

```
/*****************************************************************************/
/* defines for piezo loudspeaker                                             */
/*****************************************************************************/
#define _PIEZO_PRESENT         1    /* set 1 if piezo loudspeaker is present */
#define _PIEZO_VOLUME_ADJUST   2    /* set 2 if volume can be adjusted */
#define _VOLUME_DEFAULT_DATA 128    /* volume is considered to be an */
#define _VOLUME_LOWER_LIMIT    0    /* 8-bit value */
#define _VOLUME_UPPER_LIMIT  255
/* These parameters determine which functions are needed to access the piezo
loudspeaker. Functions and data for adjusting the volume are ommited if
_PIEZO_VOLUME_ADJUST is not set to 2.
If _PIEZO_PRESENT is not set to 1 the whole piezo.c module (and the depending
sound channel module sound.c) will be ommited. */
```

Ebenso enthält die Datei "config_gui.h" einige Schalter, die auskommentiert werden müssen:

```
/*****************************************************************************/
/* customizing sound functions                                               */
/*****************************************************************************/

/* if you want to use sound devices and channels (e. g. for button or keyboard
click) define: */

#define _SOUND_WANTED


/* if the volume of your sound device can be adjusted define: */

#define _SOUND_VOLUME_ADJUST_WANTED
```

5.1.2 Treiberfunktion für den Lautsprecher

Für die Ansteuerung eines Lautsprechers müssen Sie die Funktion `c_sound()` in den Datei "piezodriver.h/c" an Ihre Hardware anpassen:

void c_sound(uint16 frequency, unit16 duration, uint8 volume): Ausgabe eines Tones mit der Frequenz `frequency`, der Dauer `duration` in Systemticks und der Lautstärke `volume`.

5.1.3 Anwendung des Lautsprechers

Analog zu den anderen Geräten wie LCD und Touch–Glas, wird wieder ein *Lautsprechergerät* mit einem *Soundkanal* verbunden:

```
piezo_device  *piezo;              /* piezo loudspeaker    */
sound_channel loudspeaker;

piezo_init( &piezo);               /* init piezo and sound */
sound_open( &loudspeaker);
sound_connect( &loudspeaker, piezo);
```

Ist dies geschehen, so können mit dem Befehl `sound_make()` einige Töne ausgegeben werden.

```
sound_make( &loudspeaker, TONE_D*3, 14);
sound_make( &loudspeaker, TONE_FIS*3, 14);
sound_make( &loudspeaker, TONE_A*3, 20);
```

Zum Schluß wird mit diesen Funktionen aufgeräumt:

```
sound_disconnect( &loudspeaker);
sound_close( &loudspeaker);
piezo_close();
```

Um die Bedienbarkeit für den Anwender weiter zu verbessern, ist es empfehlenswert, das Drücken eines Buttons oder einer Taste mit einem Tastatur-Klick zu verknüpfen. Sie können auf die folgende Art und Weise `Touch-Klicks` einrichten, so daß bei der Berührung des Buttons oder der Taste ein kurzer Ton erzeugt wird. Initialisieren Sie die Tonerzeugung wie im Sound-Beispiel oben und ersetzen Sie den Befehl `touch_open()`:

```
touch_open( &touch_glas, &display, &loudspeaker);
```

Das komplette Beispiel finden Sie auf der CD.

5.2 Wie geht es weiter ?

Die Simplify Technologies GUI–Bibliothek bietet Ihnen noch weitere Möglichkeiten zur Gestaltung Ihrer Benutzeroberfläche, insbesondere die Möglichkeit, Buttons und Tastaturen zu konfigurieren. Im Handbuch und auf der CD zur Bibliothek finden Sie ausführliche Programmbeispiele, die alle diese Möglichkeiten demonstrieren.