Graphical LCD User Interface

Auszüge aus der Bakkalaureatsarbeit von

Mario Gartner
mario.gartner@gmx.net

Wien, Februar 2007
Abstract

This work is covering the development of a graphical LCD user interface on an Infineon C167 microcontroller using the colour display of a mobile phone.

As one part, a driver to access the LCD was adopted. Additionally, a library that offers routines for comfortable drawing of geometrical figures and text output was implemented.

The user interface API is logically separated part and operates on the next higher level, but it uses the functionalities of the driver library for the output on the LCD. It offers sufficient utilities to quickly construct a menu structure. Therefore, the functions of any system can be easily accessed by navigating through the menu.

For developing, testing and demonstration issues, a hardware user interface module was built as a prototype. It is equipped with a graphical 65k colour LCD and 8 pushbuttons for operation input and therefore ready for various applications.

Kurzfassung

Diese Arbeit behandelt die Entwicklung eines grafischen LCD User Interfaces auf einem Infineon C167 Mikrocontroller. Dabei wird das Farbdisplay eines Mobiltelefons verwendet.

Zunächst wurde für diese Aufgabe ein Treiber angepasst, der den Zugriff auf das LCD ermöglicht. Zusätzlich wurde eine Funktionsbibliothek entwickelt, die das komfortable Zeichnen von geometrischen Figuren und die Textausgabe auf dem Display ermöglicht.

Das User Interface API ist ein davon logisch getrennter Teil und operiert auf einer höher liegenden Ebene, verwendet allerdings die Funktionalitäten der Treiberbibliothek für die Ausgabe am LCD. Es bietet praktische Werkzeuge für die rasche Erstellung einer Menü-Struktur. Dadurch können die Funktionen eines beliebigen Systems sehr einfach mittels navigieren durch das Menü erreicht werden.

Für die Entwicklung sowie für Test- und Demonstrationszwecke wurde ein User Interface Modul als Hardware Prototyp aufgebaut. Es ist mit einem 65.536 Farben Display und 8 Eingabetasten ausgestattet und somit für vielfältige Anwendungen verwendbar.
# Contents

1  **Introduction**  
   1.1 Motivation  
   1.2 Assignment  

2  **Implementation**  
   2.1 The UI Hardware Module  
      2.1.1 Schematics  
      2.1.2 Arrangement Layout  
   2.2 The Graphical LCD Driver  
      2.2.1 Capabilities of the GLCD Library  
      2.2.2 Colour Coding  
      2.2.3 Screen Orientation  
      2.2.4 Function Overview  
   2.3 The User Interface API  
      2.3.1 Menu Content Structure  
      2.3.2 Navigation  
      2.3.3 Function Overview  

3  **Using the User Interface API**  
   3.1 Operating  
   3.2 Customisation  
      3.2.1 Predefined Constant Parameters  
      3.2.2 Changeable Standard Parameters  
      3.2.3 Runtime Parameters  

4  **Porting the UI to other µC-Systems**  
   4.1 Serial Data Transmission  
   4.2 Button Input  
   4.3 Timer Functions  

5  **Conclusion**
1 Introduction

1.1 Motivation

The idea to use a graphical colour LCD of a mobile phone originally came up in a project of designing a mp3-player. The fundamental advantages of those displays are their easy availability as spare parts and the low price. On the other hand they have a considerable drawback because they are proprietary developments and so there are no datasheets available. This is compensated by the experience already made with the chosen LCD by the µC hobby developer scene.

Due to the manifold field of applications, which could benefit from a flexible user interface based on a graphical display with menu-functions, offering a generally useable API (Application Programming Interface) became the central idea behind this work.

1.2 Assignment

The scope of this work is set to design a feasible API for implementing a graphical LCD user interface on a C167 microcontroller. Its functions are limited to the LS020 display, which was used in the Siemens S65 mobile phone.

The work is based on existing GLCD (Graphical LCD) drivers for Atmel’s AVR controllers by Hagen Reddmann, who wrote a driver for the Nokia 6100 display. The library was later adapted to the S65 display by Christian Kranz, mainly in assembler.

Their API offers a variety of functions to comfortable write chars or draw geometric figures to the LCD, which got ported to work on the C167 µC. This driver is named C167 GLCD. From a programmer’s point of view, it should make as little difference as possible, if he is writing an application for the AVR or the C167 version of the GLCD library.

The features of the GLCD driver will be extended with the possibility to easily build a user interface with menu navigation using direction keys and soft-function buttons. This is done on a higher level and is compatible to any of the GLCDs. The API is simply called User Interface (UI) all through this work.
2 Implementation

This chapter is about how the general ideas about the user interface were implemented in hard- and software.

2.1 The UI Hardware Module

The UI prototype board was built for development, testing and demonstration in laboratory and educational environment. It is attached to the Phytec KitCON-167KR board by plugging it on connector X3. This link provides the required supply voltages, GND and all necessary pins of the C167’s ports P3 and P8.

2.1.1 Schematics

The LCD’s power supply of 2.9V is generated by a linear regulator. The voltage of the interface lines is reduced by potential dividers using resistor. The LCD’s DAT input is connected to the controllers MTSR (Master Tx, slave Rx) pin, CLK to the SCLK (Serial Clock). The other signal lines are operated on any standard parallel port pins in output mode. The buttons are connected to port P8 and use the alternate input function, the CAPCOM (Capture/Compare) input channels.

![Fig. 1: Schematics of the hardware module.](image-url)
2.1.2 Arrangement Layout

Equipped with a graphical 65k colour LCD and 8 pushbuttons for operation input, the UI-Module is ready for different demo-applications. Four of the buttons are preferably used for directional navigation (up, down, left and right). Directly below the LCD are two soft-functional buttons, which prompt the currently associated function shown on the display. The additional two buttons on the right hand side are used as optional input, depending on the application. Currently the upper right button is used to turn of the LCD, but every button can be reassigned to another functionallity. Fig. 2 shows the arrangement of the input and output devices on foto capture.

Fig. 2: Photograph of the UI-Module wit LCD and pushbuttons.
2.2 The Graphical LCD Driver

One part of the work is the C167 version of the original GLCD library by Hagen Reddmann, which is an independent driver for the 16-bit colour TFT display type LS020 with 132x176 pixel resolution. The LCD was used in Siemens S65, M65, CX65 and SK65 mobile phones. The library acts as an API between the I/O hardware-devices and the user interface, but it is not mandatory to use them both. For applications that not require a menu based UI, the graphic routines of the GLCD may offer all needed functionalities. In that case, the user directly operates the display driver.

2.2.1 Capabilities of the GLCD Library

- All operations with 16 bit = 65,536 colours
- Set a single pixel
- draw lines with any angle to the axis
- Rectangles, filled with/without boarder or boarder or only
- Ellipses and circles, filled and with border
- All outputs might be cut by a clipping rect
- All routines support transparent writing
- Text output with different fonts, proportional and mono-spaced
- Design fonts in an editor by Hagen Reddmann

Fig. 3: Flow chart of application structure and functional dependencies.
2.2.2 Colour Coding

The GLCD driver uses 16-bit colours, encoded with a scheme using 5-6-5 bits for R-G-B. The first 5 bits, starting on MSB, represent the intensity of red, followed by 6 bits green and finally 5 bits blue. [Kr05]

The driver uses a table to store the currently used colours. Many functions use this colours for their drawing routines. The four standard colours are glcdSetBkColor for the background, glcdSetFgColor for the foreground, glcdSetFrColor for frames and glcdSetShColor for the frames shadow.

The most important colour-values are defined as macros (e.g. RED, YELLOW, WHITE …). The predefined colour NONE is used for transparent drawing.

2.2.3 Screen Orientation

The colour condition of every single pixel on the screen is stored in the displays memory, which is organised as a matrix with the same dimensions as the LCD. The driver operates the display in 90 degree orientation mode, which means the screen is in portrait format. Many functions use a coordinate system for their drawing routines. The most upper left pixel has the coordinates (0/0), the most lower right (131/175), always measured in pixel. The pixels are drawn from the upper left corner down to lower right corner, each with every transmitted 16 bit data. The area is filled from Y1 to Y2 in positive Y-direction and from X1 to X2 column by column, as depicted in figure Fig. 4. [Kr05]

![Fig. 4: Coordinate orientation and order of written pixel.](image-url)
2.2.4 Function Overview

This section describes the C167-GLCD driver and function library. The interface offers all necessary routines to draw geometrical figures or text on the screen. All function-calls are compatible to the AVR-GLCD driver for the Siemens S65 mobile phone display. For a more detailed description of all functions, please see the html documentation in the folder documentation at the project’s root directory.

Initialize LCD:

```c
void glcdDisplayInit(void)
```

The display needs to get initialized before it can be used. The function also calls the init routines `timerInit()` of the timer and the `serialInit()` of the data transmission part.

Shutdown LCD:

```c
void glcdDisplayOff(void)
```

The display is shut down by calling this function. The initialisation must be repeated for using the LCD again after the shutdown.

Clear LCD:

```c
void glcdClearScreen(uint8_t ignoreclipping)
```

If called with `ignoreclipping == 1`, the whole display gets filled with the default screen colour. If `ignoreclipping == 0`, the screen gets filled only inside of the clipping rectangle.

Draw a pixel:

```c
void glcdSetPixel(glcdCoord_t x, glcdCoord_t y, glcdColor_t color)
```

A pixel with the argumented colour gets drawn to the point (x/y).

Draw a line.

```c
void glcdLine(glcdCoord_t x1, glcdCoord_t y1, glcdCoord_t x2, glcdCoord_t y2)
```

This function draws a line in the standard foreground colour. The line is constructed between the points (x1/y1) and (x2/y2).

Draw a rectangle with coloured border:

```c
void glcdRectangle( glcdCoord_t x1, glcdCoord_t y1, glcdCoord_t x2, glcdCoord_t y2)
```
This function writes a rectangular area of pixels. The point (x1/y1) determines the first corner of the rectangle, point (x2/y2) marks the opposite corner. The most outer pixels on the boarders are drawn in the standard foreground colour (glcdFgColor), the inner pixels are filled with the background colour (gpcdBkColor).

**Draw a filled rectangle without frame:**

```c
void glcdFillRect( glcdCoord_t x1, glcdCoord_t y1,
                  glcdCoord_t x2, glcdCoord_t y2, glcdColor_t color)
```

This function writes a single-colour rectangular area of pixels. The colour is determined by the argument `color`.

**Draw an ellipse**

```c
void glcdEllipse( glcdCoord_t x, glcdCoord_t y,
                  glcdCoord_t a, glcdCoord_t b)
```

An ellipse with its centre on the point (x/y) gets drawn. The arguments a and b determine the length of the semimajor and semiminor axes. Foreground colour is used for the most outer pixel as a boarder and the circle is filled with the background colour.

**Select a font:**

```c
void glcdSelectFont(const uint8_t *data, glcdFontFunc_t callback)
```

Before text can be written to the LCD, its font must get initialised by calling this function. The first argument is a pointer to the font-array, the second is a the address of the callback function of type `glcdFontFunc_t`.

**Move cursor to coordinates:**

```c
#define glcdMoveTo(x, y)
```

This makro sets the cursor position for text output to the point (x/y)

**Print a string:**

```c
uint16_t glcdPrint(const char *text, uint8_t flashstored)
```

The function prints a 0 terminated string sored in a char[] at the cursor position on the screen. It returns the total pixelwith of the drawn letters.

**Get width of a string:**

```c
uint16_t glcdCharsWidth(const char *text, uint8_t flashstored)
```

Only returns the total pixelwith of the drawn letters. Does not write anything on the screen.
2.3 The User Interface API

Based on the provided routines of the GLCD library, the User Interface API offers feasible utilities to quickly construct a menu structure for navigation through any systems functions. As depictured in Figure 4, the User Interface is set on a higher level, but directly based on the GLCD diver and therefore the usage without it is not possible.

2.3.1 Menu Content Structure

Before a menu is ready to be used, it is necessary to describe its style and content. The programmed API offers a function to logically create a menu instance and load data into its properties-structure. That contains of the Menu Title, used colours, the type of font for title and items, the window size and a table of all associated items. Furthermore, the structure holds the displayed names for the soft functions \textit{func A} and \textit{func B}.

The menu’s contents shown on the LCD are text-based elements, each represented by a Menu Item in the software and need to be initialised too. This init-function links an item to a certain menu and saves its displayed name in a zero terminated char array. Each Menu Item also contains two function pointers. Those allow to proceed callbacks initiated by pushing one of the soft-function buttons. Figure Fig. 5 shows the arrangement of the used elements on the screen. The UI is designed to show five items in each menu. Chapter 3 describes how to customize and operate the user interface in more details.

![Menu Design](image)

**Fig. 5:** Menu design (PC draft), shows actual arrangement of the elements on the LCD.
2.3.2 Navigation

If the menu gets accessed, the cursor menu items are listed and the first element gets selected. By browsing through the menu by the up and down buttons, the cursor moves to the according direction or wraps around on top or bottom. Items are marked as selected by surrounding the text with a frame. Highlighting with a filled rectangle, e.g. combined with inverted font colour might be more attractive, but require an extensive redrawing.

NOTE: highlighting by a rectangle with a colour gradient is supported in the latest version, but not documented in this paper! The redrawing of the text plus the gradient colour does not slow down the navigation in a recognisable amount.

The soft-functions can be used for changing the menu level. The func A would call the prior menu level or the desktop, whereas func B continues with the next submenu or starts an application, depending on the selected Item.

2.3.3 Function Overview

To get a quick overview of the offered functions, the most important ones are briefly described below. The html documentation in the folder documentation at the project’s root directory gives more information about all functions available.

Create a Desktop

```c
int8_t-uiNewDesktop (const char *uiSoftA, const char *uiSoftB,
                     glcdFontData_t font, uiCall_t uiCallA, uiCall_t uiCallB)
```

By calling the function, a new desktop will be set up. It behaves like a menu without items, hence it has both soft-function areas printed on the screen and their functionality. The passed arguments describe the text of the soft-functions, the used font and the callback functions. The returned value is the number of the created menu for a later access.

Create a Menu

```c
int8_t-uiNewMenu (const char *uiTitle, const char *uiSoftA, const char *uiSoftB,
                   glcdFontData_t font, glcdFontData_t titleFont)
```

This function initializes a new menu. The passed arguments describe the title, the text of the soft-functions, and the fonts for title and the menu items. The returned value is the number of the created menu for a later access.

Create a Menu Item
int8_t uiNewMenuItem ( uint8_t uiMenuNumber, const char *uiItemText,
uiCall_t uiCallA, uiCall_t uiCallB)

To fill a menu with items, this function is used. The order of creating will be the order in which the items get displayed on the screen.

Open Menu

int8_t  uiOpenMenu (uint8_t menuNumber)

If called, the content of the specified menu gets drawn to the LCD. Returns the number of the opened menu, or a negative value if an error occurred.

Close all Menus

void uiCloseMenu ()

The menus get closed if this function gets called. All menu-related navigation functions of the button input is turned off and the screen will be cleared in the routine. If a desktop is used, opening it instead may be preferred.

Set colors.

void uiSetColors( uint8_t uiMenuNumber, glcdColor_t uiFgColor,
glcdColor_t uiBgColor, glcdColor_t uiFontColor,
glcdColor_t uiHighlightColor, glcdColor_t uiHlFontColor)

To change the colours of a menu, this function can be used after the menu was created. If it does not get called, the default colours are used.

Set fonts.

void uiSetFonts ( uint8_t uiMenuNumber, glcdFontData_t font,
glcdFontData_t titleFont)

This function might be used to change the fonts after the menu was created.
3 Using the User Interface API

3.1 Operating

To run the User Interface API, the proper GLCD display library must be used. The client software which uses the API functions has to take a couple of actions before the UI can be used. For demonstrating the functionality, the file start_ui.c is included at the ui directory.

The first step to do is to initialise the GLCD by calling the accordant function. The button input and other optional init routines should also be called in advance.

It is now possible to immediately create the complete menu structure, but it is not obligatory. A menu can be initialized at any time, as long as the maximum count was not exceeded. To use a desktop, it must be created by calling uiNewDesktop at first to keep a clean structure of menus. Other menus can be added afterwards with the uiNewMenu function. The example in start_ui.c links the corresponding menu items right after that (uiNewMenuItem) and uses the returned ID for identification.

By creating the menu items, the callback functions for soft function A and B need to be passed. Those functions must be implemented in the user’s code with the right signature. They need to be void funcXY(int8_t, int8_t) typed. It is possible to assign two different functions to each menu item. Sometimes, it can be more effective to reuse some functions and decide which action to take by analyse the arguments currentMenu and currentItem. The demo application uses both ways.

3.2 Customisation

The UI API allows to customise the behaviour and the appearance of the menus. Several parameters of the user interface and the menu are defined in the ui.h header file as macros. Those can be changed before the code gets compiled and are valid for all menus. Others are previously defined as standard values but can be changed in runtime. Some additional parameters must always be passed when the menu gets initialised.
3.2.1 Predefined Constant Parameters

Since all structures representing a menu are stored in an array, it is necessary to specify its size and therefore the maximum number of menus including the desktop (UI_MAX_MENUS). Other values stored in arrays are the maximum items per menu (UI_MAX_ITEMS), the menu-title length (UI_TITLE_LEN) and the length of a menu item’s text (UI_ITEM_TXT_LEN).

Furthermore, the arrangement of the items are specified in advance. The distance from the left screen border to the first letter of a menu item (item 0) is defined in UI_ITEM_X and the distance from the top in y-direction in UI_ITEM0_Y. Other items are positioned relatively to item 0 with the offset UI_ITEM_OFFSET, which is the spacing between two items in y-direction.

3.2.2 Changeable Standard Parameters

The standard colours for all menu’s elements are stored as pre-defined values. A menu generally gets initialized with the standard values, but they can be changed at any time by calling the uiSetColors function. The menu just needs to be reopened afterwards. Figure 7 shows the names of the colours and in which way they are used in a menu.

Fig. 6: Menu design (PC draft), shows the usage of the different colours.

3.2.3 Runtime Parameters

Some parameters strictly depend on the application, consequently it is not helpful to use standard values in that cases. Those parameters get passed as arguments when the menu or the menu item is created by calling the accordant function. Please have a look a the function description or the html documentation for a detailed explanation of the functions arguments.
4 Porting the UI to other µC-Systems

The originally GLCD v1 by Hagen Reddmann was a driver for the display of the Nokia 6100 mobile phone for Atmel AVR controllers. The core routines to draw geometrical figures and to write characters in different fonts could be reused for the C167 GLCD.

All display-hardware related operations to write the Siemens S65 LCD were programmed by using research work and an AVR driver, both done by Christian Kranz. Because his main attention was to built a small and high-performance code especially for the AVR architecture, all functions were written in strictly hardware-dependant assembler language.

As an important objective, the C167 GLCD should be more flexible and as hardware independent as possible. Due to the usage of standard C code and the structural design of the whole User Interface, it is possible to easily port it to other architectures. All changes needed to be done are described in the following paragraphs and are limited to the files highlighted in Fig. 7.

For the usage of the User Interface, a successfully working GLCD driver is mandatory. The API was developed and tested with the S65 mobile phone display, but since it does not directly access the hardware, it should be easy to run it with other LCD types too. In that case, the arrangement of the menu elements need to be reconfigured to fit to the actual screen size. Also the button input has to be adapted to the controller hardware.

![Fig. 7: Structure of files; selected items need to be changed for other hardware](image)
4.1 Serial Data Transmission

The hardware related function of sending data via C167’s SSC are encapsulated to the serial.c file and need to be exchanged. A new serial.c file must provide a function to send a 16 bit wide data packet by SPI compatible communication with this signature:

```
serialSend(uint16_t data, uint8_t cmd)
```

The arguments passed represent the data to send and a command identifier. The cmd flag represents the status of the RS pin, which signals if a command (RS = high) or data is sent (RS = low). Both data and cmd can be stored to a ringbuffer and get sent if the controller is ready to obtain a good data flow.

The µC acts as master and must therefore generate an appropriate clock signal. Originally, the Siemens S65 uses a frequency of 13MHz, but the C167 can maximally use a fourth of the system clock. Hence, the GLCD driver for the C167 operates the display with a data rate of 5 mega baud. A quite wide range of transmitting speed seems to be supported and therefore it should be as fast as possible, somewhere between 5 and 13 MBd. The data to the LCD is transmitted with MSB first and the data is latched at the display’s controller with the rising clock edge. Those settings are done in an initialisation routine that gets called by the GLCD driver init function. Thus, the new serial.c must provide this function with the signature:

```
void serialInit();
```

4.2 Button Input

For the C167 user interface, the input buttons are attached to port P8.0 to p8.7. The asynchronous events are captured by using the pin’s alternate function, the CAPCOM input, and are handled as external interrupts with a sample rate of 16 TCL. This has performance advantages over polling the pins status, particularly if the key input should be debounced and a button that is held down should be recognized differently to a single tap.

Generally, handling the user input buttons is not very time-critically, therefore they have the lowest interrupt level (ILVL). Since all buttons are assumed as to be equally important and the pushed order should not get shuffled, none of their ISRs should interrupt each other. This is granted by using a software controlled interrupt class that contents all button’s ISRs.

The button input module in the buttons.c file must capture and validate the input event. It is not mandatory how to provide this, as long as the appropriate function of the user interface gets called. These are always named in the manner
void UIButton_<name>();

with the matching buttons signature for the `<name>` mark. Those are respectively up, down, left, right, softA, softB, 1 and 2. Currently, there is also a initialisation function, but it does not get called by any internal function of the UI API. The user has to make sure to correctly initialise his input functions somewhere in his code.

4.3 Timer Functions

One of the C167 µC’s timers is used to provide accurate timing for the LCD initialisation sequences. The driver calls a function to wait an integer number of milliseconds, specified by the passed argument:

    void ms_sleep(unsigned int ms)

The settings for the timer are performed in a routine called by the GLCD driver init function:

    void init_timer();

Again, it is not mandatory how to generate the delay, as long as the user provides a function with the right signature that pauses the advance of the program for the passed number of milliseconds.
5 Conclusion

The previously aimed objectives could be achieved satisfactorily. All key functionalities of the original idea of a menu based UI were implemented and have been successfully demonstrated on hardware.

Because the LCD’s memory can not be read, functions that need a directly link to its content can not be done. Such operations would therefore need a dedicated memory for the displays content. In that case, the way of communication would be needed to get improved, to achieve the required speed.

An additional improvement of the UI would be to offer advanced functions for user input. This could be a pop-up window for value input, message dialogs or some standard components for a grid based form drawing. Input by rotary switches could be supported to enter text, numbers etc.

As a possible enhancement of the Graphical LCD User Interface, the whole API could be implemented to run on an operating system. That would be well suitable because of its characteristics of being a part of a bigger application.

Literature:

[Kr05] Using the Siemens S65-Display, Christian Kranz, 2005
http://www.superkranz.de/christian/S65_Display/DisplayIndex.html

Figures:

Figure 5: [Kr05], edited with permission.

All other figures: own work

Special thanks to Hagen Reddmann and Christian Kranz.