Graphic Dot Matrix LCD Driver Library

Application note
for
single chip processor applications
using
KS07xx

READ THIS MANUAL BEFORE YOU START

RAMTEX

Disclaimer

The information in this document is subject to change without notice. While the information contained herein is assumed to be accurate, RAMTEX assumes no responsibility for any errors or omissions.

Copyright Notice

Trademarks

STIMGATE is a trademark of RAMTEX International
StreamWindows is a trademark of RAMTEX International

Contact information, sale and service

RAMTEX *International ApS*
Box 84, Skodsborgvej 346,
DK-2850, Nærum
Denmark
Phone: +45 4550 5357 Fax: +45 4550 5390
Email: sale@ramtex.dk Web: http://www.ramtex.dk

Application note revision

Revision 1.0 Apr. 2002
The LCD simulator is produced and maintained by RAMTEX International ApS
The Dot Matrix LCD Driver Library is produced and maintained by RAMTEX Engineering ApS

Table of content

Introduction

This is an application note for how to use the Graphic Dot Matrix LCD driver library with single-chip processors. It describes the special consideration which have to be made to when an LCD (Liquid Crystal Display) module using the KS07xx LCD controller is connected to the Input/Output (I/O) ports of a processor chip.

With single-chip processors the signals of the normal external bus interface used by the LCD controller chip must be simulated by software controlling I/O ports. The address and bus control signals of the LCD controller must be connected to output ports and the data bus of the LCD controller must be connected to an 8 bit I/O port which can be programmed to be bi-directional.

Required background knowledge

Processor architecture

The different processor architectures use different architectures for I/O ports. You must know in advance exactly how the I/O ports of your processor are working. Especially you must know how to switch the direction of the "data bus" I/O pins from input to output and back.

Typically one of these 3 concepts is used:

1. When logical high the I/O pin has a week pull-up. When logical low the I/O pin has a strong pull-down. Here all pins of the data bus register must be programmed to logical high when used as an input port. (Ex the 8051 processor family).

2. The processors are using 2 registers. One for controlling the direction of the I/O port and one Read/Write port for either reading the input pin status or writing the output level. Here the direction register must be updated before each read or write operation.

3. The processors are using 3 registers. One for controlling the direction of the I/O port and one read-only port for reading the pin status and one write-only port for writing the output level. Here the direction register must be updated before each read or write operation. (Ex the H8S processors)

Consult the relevant chapters of your processor hardware manual first to find out exactly how the I/O ports are working in your target system.

Compiler syntax

Nearly all compilers for embedded processors support I/O operations from the C source level.

However, because the syntax for doing I/O operations is not standardized by the international "ISO/IEC Programming language C" standard (successor of the ANSI-C standard) each individual compiler vendor uses their own proprietary syntax. You must know in advance exactly how I/O port access should be handled with your target C compiler. Especially you must know how to define I/O registers at fixed addresses and wether the C compiler (and processor) supports direct addressing of I/O port bits.

Consult the relevant chapters of your C compiler manual first to find out exactly how I/O ports are defined for your C compiler and how to access I/O ports in the source code. Typically you find I/O port access under

Intrinsic features or *Intrinsic functions* (compiler vendor specific features and functions).

See also the chapter : *How to implement your own SG header files* in the main manual for the LCD library.

Bus interface for a LCD module

The different LCD modules vendors are using somewhat different bus pin interfaces. A typical microprocessor interface for a KS07xx based module is described in the following table, together with the symbolic port names used by the software examples last in this application note. The KS07xx bus interface can be 6800a like (R/W and E clock) or 8080 like (/WR and /RD clocks). The following description assumes that a 8080 bus is used.

PIN NAME	LCD LEVEL	SYMBOLIC NAME IN C	FUNCTIONALITY AND COMMENTS ABOUT SOFTWARE CONTROL
VSS	-	-	Power supply ground (and logical 0 for bus signals) (Not relevant for the LCD driver library.)
VDD	-	-	Power supply +5V (and logical 1 for bus signals) (Not relevant for the LCD driver library.)
C86	Input	-	C86 = “H”: 6800 Series MPU interface. C86 = “L”: 8080 MPU interface. Here 8080 mode (L) is assumed
P/S	Input	-	Parallel / Serial select. Here parallel mode (H) is assumed.
/WR	Input	NWR	Negative write strobe. Used for clocking data into the LCD controller. All other signals must be stable when this signal goes high. Always a processor output signal. The processor data bus must be an output while this signal is active.
/RD	Input	NRD	Negative read strobe. Used for enabling LCD controller data to the databus. All other signals must be stable while this signal is active low. Always a processor output signal. The processor data bus port must be programmed as an input port before this signal goes low.
/CS	Input	NCE	Negative Chip Enable. Must be low while /RD or /WR is low. Always a processor output signal.
CS	Input	-	Is assumed to be hardwired to 1.
A0	Input	CD	Address bus” for selecting either the Control / status registers or the Data bus registers. Always a processor output signal.
/RST	Input	NRST	Negative reset input. Resets the LCD controller. Should be toggled low at least once after the +5V power supply has stabilized.. May be connected to a processor output port or hardwired to the processors /RESET input signal.
DB0	I/O	DATAWR DATARD	LCD Data bit 0 - 7. Should typically be connected to an 8 bit input port on the processor. The data direction on the bus is controlled by /RD and /WR. The data direction of the processor must always be opposite of the data direction of the LCD display. Two different symbolic names are used for bus read and bus write as some processors uses different registers for reading port input and writing port output. Some processors uses a third internal register to control the direction of the port pins (i.e. to enable or tri-state the port output register). This direction control register must be updated before each bus read or write operation.
DB1	I/O		
DB2	I/O		
DB3	I/O		
DB4	I/O		
DB5	I/O		
DB6	I/O		
DB7	I/O		

Adapting the LCD library for use with single chip processors

Most I/O operations in the Dot Matrix LCD library is concentrated in the file *ghwinit*.c. Normally the library uses a subset of the portable SG syntax for all I/O access operations. You can read more about the SG syntax at *www.ramtex.dk/standard/sgsyntax.htm*.

In short the LCD library only uses the following functions for direct I/O operations on 8 bit registers.

```
void sgwrby(portname, SGUCHAR data);
SGUCHAR sgrdby(portname);
```

To use the LCD library with a single chip processor the SG (macro) functions for direct port access must be replaced by two corresponding driver function which simulates the signals of an external bus for the LCD module using ordinary I/O ports on the processor.

When the preprocessor switch GHW_SINGLE_CHIP is defined in the compiler the *sgwrby()* and *sgrdby()* functions are replaced by calls of the corresponding bus simulator functions:

```
void simwrby(SGUCHAR LCD_address, SGUCHAR data);
SGUCHAR simrdby(SGUCHAR LCD_address);
```

These new functions must be implemented so they simulate the signal sequences of a normal external bus. Template examples are shown last in this application note.

Files for use with single chip processors located on the `\GLCD\controller\SINGLCHP\` directory.

bussim.c	Template file with comment which describes how the bus simulator functions should be implemented. (Can also be used as a stub for compilation tests)
bussimbi.c	Template file using bit operations of the individual control signals.
bussimby.c	Template file using only byte operations on a (8 bit) control signal port.

A LCD reset function in *bussimxx.c* is invoked by *ginit()* via *ghw_io_init()* before any I/O addressing takes place. Place LCD reset activation and any bus initialization required in the reset function:

```
void sim_reset(void);
```

You must do a little work your self.

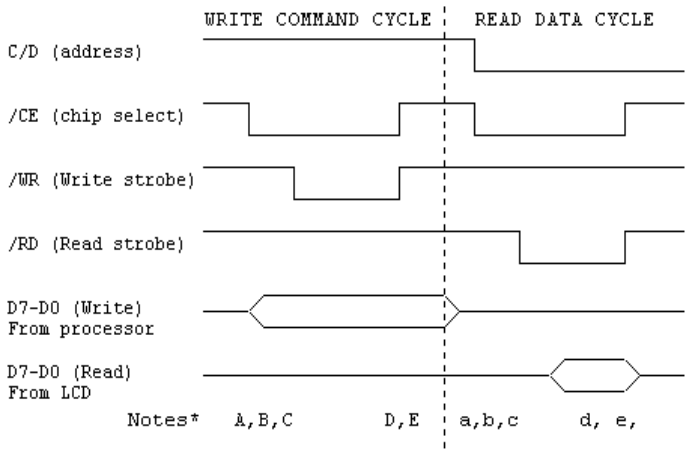
As every target system is different the body of the *bussim.c* functions must be implemented (or adjusted) to fit the processor chip architecture and the way you have designed the port layout in your target hardware. To be able to successfully full-fill this task there is no way around reading the relevant hardware manuals for the processor chips. If you do not already have an intimate know-ledge about the single-chip processor hardware and the architecture of on-chip I/O registers it is time to look into the manuals NOW.

The template files *bussimbi.c* and *bussimby.c* shows implementation examples using symbolic names for the LCD control signal bits and the I/O registers, and the SG notation for I/O port operations.
You must be able to make the right definitions for these symbolic names so they matches the actual processor and port configuration in your hardware (and the I/O access and definition syntax of your C compiler). The LCD control signals has to be modified in the right logical sequence.
The necessary logical behavior is described in the template files and documented in the following chapter.

Simulated bus signal sequences

It is assumed that the KS07xx based display is configured for an Intel like interface with separate /WR and /RD signals. The logical signal sequences for a write operation and a read operation is shown in figure 1.

The `simwrby(address,data)` function must simulate the write sequence, and the `simrdby(address)` function must simulate the read sequence.
In figure 1 the notes A-E and a-e refer to places where the software must do some operations. These points are also referred in the comment of the *bussimxx.c* template files. See below.



Figur 1 Logical signal sequences for bus simulator

Template example for external LCD bus simulator functions (located in *bussim.c*):

```
void simwrby(SGUCHAR adr, SGUCHAR dat)
   {
   /* A: Set C/D line according to adr (0 or 1), Set /CE line active low */
   /* B1: Make data port an output (if required by port architecture) */
   /* B2: Write data to data port */
   /* C:  Set /WR active low */
   /* D:  Set /WR passive high */
   /* E:  Set /CE passive high (could be ignored */
   }

SGUCHAR simrdby(SGUCHAR adr)
   {
   SGUCHAR dat = 0;
   /* a:  Set C/D line according to adr (0 or 1). Set /CE line active low */
   /* b:  Make data port an input (if required by port architecture) */
   /* c:  Set /RD active low */
   /* d:  Read data from data port */
   /* e1: Set /RD passive high */
   /* e2: Set /CE passive high (could be ignored )*/
   return dat;
  }

void sim_reset( void )
   {
   /* 1. Init data port setup (if required by port architecture) */
   /* 2. Make C/D,/RD,/WR,/CE to outputs (if required by port architecture), and high */
   /* 3. Set LCD reset line /RST active low (if /RST is connected to a port bit) */
   /* 4. Set LCD reset line /RST passive high (if /RST is connected to a port bit) */
   }
```

Timing

The only critical timing is the active low time for /RD and /WR (pkt C-D and pkt c-d). They must obey come minimum requirements given by the KS07xx controller. Typically:

- The /RD and /WR low time must be 70 ns or longer.
- The /RD signal must be low for 70 ns before data is read.

The actual software timing for a given implementation will depend on the processor type, the clock oscillator frequency used, your implementation method, and the code generation efficiency of your C compiler. You can test your implementation by calling the *simrdby* or *simwrby* function repeatedly in a short loop and measure the timing with an oscilloscope. If the timing in the actual hardware turns out to be critical then stretch the timing by repeating the code lines for operation C and c respectively. Step (E) and (e2) respectively could be ignored as it just will cause the LCD control signals to be stable until the next LCD read or write operation.

Template example using bit and byte operations

The source code is located in *bussimbi.c*

```
/***************************** bussimbi.c ***********************************

   Generic template file for external bus simulator drivers.

   The functions in this module is called by single-chip-processor version
   of the ghwinit.c module.

   This implementation assumes that the processor port architecture supports
   BIT addressing capabilities and that BIT addressing is also supported by
   the target compiler.

   The compiler independent SG syntax notation is used in this example for
   clarification.
   (Alternatively these functions could be implemented using the compiler
    specific syntax for I/O port operations.)

   The following symbolic names for bit and byte ports must be declared for
   the actual processor architecture and target system:

   Bit ports:
      CD     (LCD address line)
      NWR    (LCD /WR line)
      NRD    (LCD /RD line)
      NCE    (LCD /CE line)
      NRST   (LCD /RST line, if connected to a port)

   Byte ports:
      DATAWR (LCD data write)
      DATARD (LCD data read. May be equal to DATAWR, depends on the processor)
      DATADIR (Processor data port direction setup, depends on the processor).
      CTRLDIR (Processor ctrl port direction setup, depends on the processor).

   Copyright (c) RAMTEX Engineering Aps 1998

*****************************************************************************/
#ifndef GHW_NOHDW
#ifdef  GHW_SINGLE_CHIP

#include <bussim.h>
#include <gdispcfg.h> /* Include GDISPCW declaration */

#include <sgio.h>   /* the above port names must be defined in sgio_ta.h
                       using the appropriate compiler syntax. Use the SGSETUP
                       tool when ever possible */

/*#include < Port declaration file for your compiler >
/* Is included instead of sgio.h if the compiler specific syntax for port
   access is used */

/*
   Update these defined values according to processor port architecture
   (if port direction control is needed)
*/
#define DATABUS_IS_INP  0x00  /* 8 signal lines is used as input */
#define DATABUS_IS_OUTP 0xFF  /* 8 signal lines is used as outputs */
#define CTRLBUS_IS_OUTP 0x3F  /* 4-6 signal lines is used as output */

/*
   Simulate a bus write operation for a LCD controller with an Intel
   like bus interface (i.e. use of separate /RD and /WR strobes).

   The address parameter adr is assumed to be either 0 or 1.
*/
void simwrby(SGUCHAR adr, SGUCHAR dat)
   {
   /* A: Set C/D line according to adr, Set /CE line active low */
   sgwrbi(CD, (adr != 0));
   sgwrbi(NCE, 0);
   /* B1: Make data port an output (if required by port architecture) */
   sgwrby(DATADIR,DATABUS_IS_OUTP);
   /* B2: Write data to data port */
   sgwrby(DATAWR,dat);
   /* C: Set /WR active low, (Delay min 80 ns), */
   sgwrbi(NWR, 0);     /* repeat this instruction if more delay is needed*/
   /* D: Set /WR passive high */
```

```
   sgwrbi(NWR, 1);
   /* E: Set /CE passive high */
   sgwrbi(NCE, 1);
   }

/*
   Simulate a bus read operation for a LCD controller with an Intel
   like bus interface (i.e. use of separate /RD and /WR strobes).

   The address parameter adr is assumed to be either 0 or 1.
*/
SGUCHAR simrdby(SGUCHAR adr)
   {
   SGUCHAR dat;
   /* a: Set C/D line according to adr. Set /CE line active low */
   sgwrbi(CD, (adr != 0));
   sgwrbi(NCE, 0);
   /* b: Make data port an input (if required by port architecture) */
   sgwrby(DATADIR,DATABUS_IS_INP);
   /* c: Set /RD active low */
   sgwrbi(NRD, 0);     /* repeat this instruction if more delay is needed*/
   /* d: Read data from data port */
   dat = sgrdby(DATARD);
   /* e1: Set /RD passive high */
   sgwrbi(NRD, 1);
   /*, Set /CE passive high (could be ignored) */
   sgwrbi(NCE, 1);
   return dat;
   }

/*
  Initialize and reset LCD display.
  Is called before simwrby() and simrdby() is invoked for the first time

  The KS07xx reset line is toggled here if it connected to a bus port.
  (it may alternatively be hard-wired to the reset signal to the processors
  in the target system).

  The sim_reset() function is invoked automatically via the ginit() function.
*/
void sim_reset( void )
   {
   /* 1. Init data port setup (if required by port architecture) */
   sgwrby(DATADIR,DATABUS_IS_INP);
   /* 2. Make C/D, /RD, /WR, /CE (/RST) to outputs (if required by port architecture) */
   sgwrby(CTRLDIR,CTRLBUS_IS_OUTP);
   sgwrbi(NCD, 1);
   sgwrbi(NWR, 1);
   sgwrbi(NRD, 1);
   sgwrbi(NCE, 1);
   /* 3. Set LCD reset line /RST active low   (if /RST is connected to a port bit) */
   sgwrbi(NRST, 0);
   /* 4. Set LCD reset line /RST passive high (if /RST is connected to a port bit) */
   sgwrbi(NRST, 1);
   }

#endif /* GHW_SINGLE_CHIP */
#endif /* GHW_NOHDW */
```

Template example using only byte operations

The source code is located in *bussimby.c*

```
/***************************** bussimby.c ***********************************

   Generic template file for external bus simulator drivers.

   The functions in this module is called by single-chip-processor version
   of the ghwinit.c module.

   This implementation assumes that the processor port architecture only
   supports BYTE wide port access. Unused bits in the bus CTRL port is left
   unchanged.

   The compiler independent SG syntax notation is used in this example for
   clarification.
   (Alternatively these functions could be implemented using the compiler
    specific syntax for I/O port operations.)

   The following symbolic names for bit and byte ports must be declared for
   the actual processor architecture and target system:

   Byte ports:
      DATAWR (LCD data write)
      DATARD (LCD data read. May be equal to DATAWR, depends on the processor)
      CTRLWR (LCD ctrl port write)
      CTRLRD (LCD ctrl port read, May be equal to CTRLWR, depends on the processor)

      DATADIR (Processor data port direction setup, depends on the processor).
      CTRLDIR (Processor ctrl port direction setup, depends on the processor).

   Copyright (c) RAMTEX Engineering Aps 1998-2001

*****************************************************************************/
#ifndef GHW_NOHDW
#ifdef  GHW_SINGLE_CHIP

#include <bussim.h>
#include <gdispcfg.h> /* Include GDISPCW declaration */

#include <sgio.h>   /* the above port names must be defined in sgio_ta.h
                       using the appropriate compiler syntax. Use the SGSETUP
                       tool when ever possible */

/*#include < Port declaration file for your compiler >
/* Is included instead of sgio.h if the compiler specific syntax for port
   access is used */

/*
   Update these defined values according to processor port architecture
   (if port direction control is needed)
*/
#define  DATABUS_IS_INP  0x00  /* 8 signal lines is used as input */
#define  DATABUS_IS_OUTP 0xFF  /* 8 signal lines is used as outputs */
#define  CTRLBUS_IS_OUTP 0x3F  /* 4-6 signal lines is used as output */

/* Define bits for control port (modify these definements to fit target) */
#define CD     0x01   /* LCD address line) */
#define NWR    0x02   /* LCD /WR line) */
#define NRD    0x04   /* LCD /RD line) */
#define NCE    0x08   /* LCD /CE line) */
#define NRST   0x10   /* LCD /RST line, if connected to a port) */
#define CTRLMSK  (CD|NWR|NRD|NCE)  /* Mask of port bits used dynamically */


/*
   Simulate a bus write operation for a LCD controller with an Intel
   like bus interface (i.e. use of separate /RD and /WR strobes).

   The address parameter adr is assumed to be either 0 or 1.
*/
void simwrby(SGUCHAR adr, SGUCHAR dat)
   {
   /* A: Set C/D line according to adr, Set /CE line active low */
   sgwrby(CTRLWR, (sgrdby(CTRLRD) | CTRLMSK) & ((adr == 0) ? ~(NCE|CD) : ~NCE));
   /* B1: Make data port an output (if required by port architecture) */
   sgwrby(DATADIR,DATABUS_IS_OUTP);
   /* B2: Write data to data port */
```

```
   sgwrby(DATAWR,dat);
   /* C: Set /WR active low, (Delay min 80 ns), */
   sgwrby(CTRLWR, (sgrdby(CTRLRD) & ~NWR)); /* repeat this instruction if more delay is needed*/
   /* D: Set /WR passive high */
   /* E: Set /CE passive high (Could be ignored, but does not cost anything here) */
   sgwrby(CTRLWR, (sgrdby(CTRLRD) | NWR | NCE));
   }

/*
   Simulate a bus read operation for a LCD controller with an Intel
   like bus interface (i.e. use of separate /RD and /WR strobes).

   The address parameter adr is assumed to be either 0 or 1.
*/
SGUCHAR simrdby(SGUCHAR adr)
   {
   SGUCHAR dat;
   /* a: Set C/D line according to adr. Set /CE line active low */
   sgwrby(CTRLWR, (sgrdby(CTRLRD) | CTRLMSK) & ((adr == 0) ? ~(NCE|CD) : ~NCE));
   /* b: Make data port an input (if required by port architecture) */
   sgwrby(DATADIR,DATABUS_IS_INP);
   /* c: Set /RD active low, (Delay min 150ns), */
   sgwrby(CTRLWR, (sgrdby(CTRLRD) & ~NRD)); /* repeat this line if more delay is needed*/
   /* d: Read data from data port */
   dat = sgrdby(DATARD);
   /* e: Set /RD passive high, Set /CE passive high (could be ignored but the cost is zero) */
   sgwrby(CTRLWR, (sgrdby(CTRLRD) | NRD | NCE));
   return dat;
   }

/*
  Initialize and reset LCD display.
  Is called before simwrby() and simrdby() is invoked for the first time

  The KS07xx reset line is toggled here if it connected to a bus port.
  (it may alternatively be hard-wired to the reset signal to the processors
  in the target system).

  The sim_reset() function is invoked automatically via the ginit() function.
*/
void sim_reset( void )
   {
   /* 1. Init data port setup (if required by port architecture) */
   sgwrby(DATADIR,DATABUS_IS_INP);
   /* 2. Make C/D, /RD, /WR, /CE (+ FS and /RST) to outputs (if required by port architecture) */
   sgwrby(CTRLDIR,CTRLBUS_IS_OUTP);
   sgwrby(CTRLWR, (sgrdby(CTRLRD) | CTRLMSK));
   /* 3. Set LCD reset line /RST active low   (if /RST is connected to a port bit) */
   sgwrby(CTRLWR, (sgrdby(CTRLRD) & ~NRST));
   /* 4. Set LCD reset line /RST passive high (if /RST is connected to a port bit) */
   sgwrby(CTRLWR, (sgrdby(CTRLRD) | NRST));
   }

#endif /* GHW_SINGLE_CHIP */
#endif /* GHW_NOHDW */
```