16x2 Character Display for Altera DE2-Series Boards

For Quartus II 14.0

1 Core Overview

The 16x2 Character Display core facilitates communication with the 16×2 Liquid Crystal Display (LCD) on Altera's DE2-115 boards.

2 Functional Description

A block diagram of the 16x2 Character Display core is shown in Figure 1. It includes an Avalon slave port for connecting to Qsys systems, and a separate interface that is connected to the 16×2 character display on the DE2-115 board. The core communicates with the display through the *Instruction* and *Data* registers shown in Figure 1, which are described in section 4. As indicated in the figure, the core includes circuitry that automatically intializes the 16×2 character display when the Qsys system is reset.

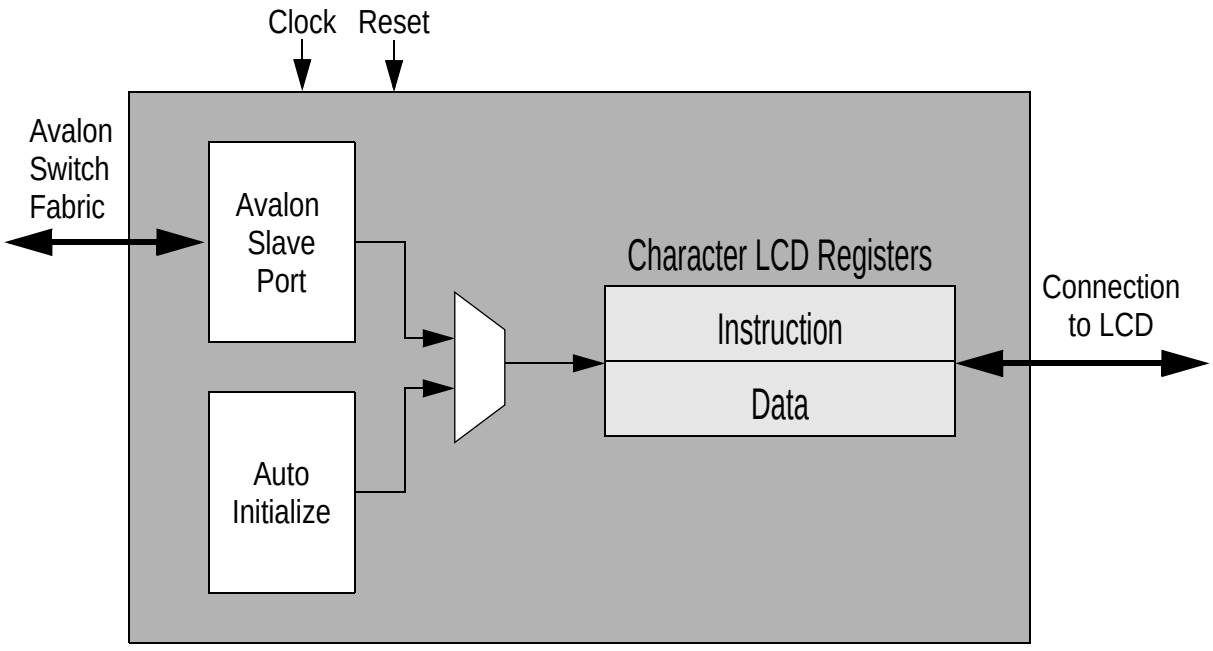


Figure 1. Block Diagram for 16x2 Character Display Core

The 16x2 Character Display core provides a memory-mapped interface for sending instructions and data to the 16×2 character display. The core supports a clock frequency of 50 MHz, which is readily available on the DE2-115 boards.

3 Instantiating the Core

The 16x2 Character Display core can be instantiated in a system using Qsys or as a standalone component from the IP Catalog within the Quartus II software.

3.1 Configuration Wizard

The configuration wizard sets the cursor type. Other settings of the 16x2 Character Display core are automatically initialized.

3.1.1 Display Cursor

Choose the desired cursor display. The 16x2 Character Display core supports Normal, Blinking, Both (Normal & Blinking) and None (for no cursor) options.

4 Software Programming Model

The programming interface for the 16x2 Character Display core consists of the two registers that were shown in Figure 1. The *Instruction* register is used to control the 16×2 character display, and the *Data* register is used to send character data to the display. Data can be sent as ASCII character codes which are automatically converted by a character-generator ROM into bit patterms by the display. The display also supports other non-ASCII characters as described in the LCD Datasheet.

4.1 Register Map

When instantiated in an Qsys system, the 16x2 Character Display core is assigned a base address, as a memory-mapped device. As Table1 shows, each of the two registers in the core is one byte in width. The *Instruction* register has the offset 0 from the base address, and the *Data* register has the offset 1.

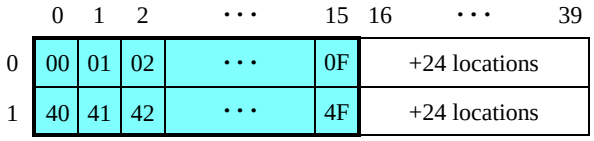
Table 1. 16x2 Character Display core register map

Offset in bytes	Register Name	Read/Write	7...0
0	`Instruction`	R/W	Instruction bits; used to read and write to the display where $rs = 0$. See the 16x2 Display's Datasheet for details.
4	`Data`	R/W	Index for the display's character generator ROM. Used to read and write to the display where $rs = 1$. See the 16x2 Display's Datasheet for details.

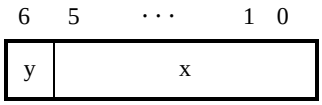
The *Instruction* and *Data* registers can be used together to store character data into each location in the display. Figure 2a shows that the 16×2 character display includes memory locations for storing two rows of 40 characters.

The first 16 locations in each row are visible on the display and the remaining are not visible at any given time. As shown in Figure 2a, the addresses of the visible locations in the top row are $(00)_{16}$ ... $(0F)_{16}$, and in the bottom row are $(40)_{16}$ ... $(4F)_{16}$.

The *Instruction* register is used to send commends to the 16×2 character display as defined in the 16x2 Display's Datasheet. Some of the instructions supported by the display are listed in Table 2. The first instruction, which is identified by $b_7 = 1$, is used to set the cursor location in the display to a specific adderss. The address is specified in the bits b_{6-0}, and follows the addressing scheme illustrated in Figure 2. Part b of the figure shows how the address of each location can be formed from its x,y coordinates, in which $y = 0$ for the top row and $y = 1$ for the bottom. After the location of the cursor has been set, a character can be loaded into this location by writing its value into the *Data* register.



(a) 16 x 2 character display



(b) 16 x 2 character display addresses

Figure 2. The 16×2 character display

When data is written into the cursor location, the 16×2 character display automatically advances the cursor one position to the right. Multiple characters can be loaded into the display by writing each character in succession into the *Data* register. As we showed in Figure 2, the 16×2 character display includes 40 locations in each row. When the cursor is advanced past address $(0F)_{16}$ in the top row, the next 24 characters are stored in locations that are not visible on the display. After 40 characters have been written into the top row, the cursor advances to the bottom row at address $(40)_{16}$. At the end of the bottom row, the cursor advances back to address $(00)_{16}$.

The 16×2 character display has the capability to shift its entire contents one position to the left or right. As shown in Table 2, the instruction for shifting left is $(18)_{16}$ and the instruction for shifting right is $(1C)_{16}$. These instructions cause both rows in the display to be shifted in parallel; when a character is shifted out of one end of a row, it is rotated back into the other end of that same row. It is possible to turn off the blinking cursor in the display by using the instruction $(0C)_{16}$, and to turn it back on using $(0F)_{16}$. The display can be erased, and the cursor location set to $(00)_{16}$, by using the instruction $(01)_{16}$.

Instruction	b_7	$b6-0$
Set cursor location	1	*Address*
Shift display left	0	0011000
Shift display right	0	0011100
Cursor off	0	0001100
Cursor blink on	0	0001111
Clear display	0	0000001

Table 2. 16x2 Character Display instructions.

4.2 Device Drivers for the Nios II Processor

A set of device driver functions for the 16x2 Character Display core are described below. These functions are intended to be used as part of Altera's Hardware Abstraction Layer (HAL) system and the Nios II processor. An example of C code that shows how to use these functions is provided at the end of this section.

To use the functions, the C code must include the statement:

```
#include "altera_up_avalon_character_lcd.h"
```

The functions provided are listed below.

4.2.1 alt_up_character_lcd_init

Prototype: `void alt_up_character_lcd_init(alt_up_character_lcd_dev *lcd)`
Include: `<altera_up_avalon_character_lcd.h>`
Parameters: `lcd` – struct for the LCD Controller device
Description: Initialize the LCD by clearing its display.

4.2.2 alt_up_character_lcd_open_dev

Prototype: `alt_up_character_lcd_dev* alt_up_character_lcd_open_dev(const char *name)`
Include: `<altera_up_avalon_character_lcd.h>`
Parameters: `name` – the character LCD name. For example, if the character LCD name in Qsys is "character_lcd_0", then *name* should be "/dev/character_lcd_0"
Returns: The corresponding device structure, or NULL if the device is not found
Description: Open the character LCD device specified by *name* .

4.2.3 alt_up_character_lcd_write

Prototype: `void alt_up_character_lcd_write(alt_up_character_lcd_dev *lcd, const char *ptr, unsigned int len)`
Include: `<altera_up_avalon_character_lcd.h>`
Parameters: `lcd` – struct for the LCD Controller device
`ptr` – the pointer to the char buffer
`len` – the length of the char buffer
Returns: nothing
Description: Write the characters in the buffer pointed to by *ptr* to the LCD, starting from where the current cursor points to.

4.2.4 alt_up_character_lcd_string

Prototype: `void alt_up_character_lcd_string(alt_up_character_lcd_dev *lcd, const char *ptr)`
Include: `<altera_up_avalon_character_lcd.h>`
Parameters: `lcd` – struct for the LCD Controller device
`ptr` – the pointer to the char buffer
Returns: nothing
Description: Write the characters in the NULL-terminated string to the LCD.

4.2.5 alt_up_character_lcd_write_fd

Prototype: `void alt_up_character_lcd_write_fd(alt_fd *fd, const char *ptr, unsigned int len)`
Include: `<altera_up_avalon_character_lcd.h>`
Parameters: –
Description:

4.2.6 alt_up_character_lcd_set_cursor_pos

Prototype: `int alt_up_character_lcd_set_cursor_pos(alt_up_character_lcd_dev *lcd, unsigned x_pos, unsigned y_pos)`
Include: `<altera_up_avalon_character_lcd.h>`
Parameters: `lcd` – struct for the LCD Controller device
`x_pos` – x coordinate (0 to 15, from left to right)
`y_pos` – y coordinate (0 for the top row, 1 for the bottom row)
Returns: 0 for success
Description: Set the cursor position.

4.2.7 alt_up_character_lcd_shift_cursor

Prototype: `void alt_up_character_lcd_shift_cursor(alt_up_character_lcd_dev *lcd, int x_right_shift_offset)`
Include: `<altera_up_avalon_character_lcd.h>`
Parameters: `lcd` – struct for the LCD Controller device
`x_right_shift_offset` – the number of spaces to shift to the right. If the offset is negative, then the cursor shifts to the left.
Returns: nothing
Description: Shift the cursor to left or right.

4.2.8 alt_up_character_lcd_shift_display

Prototype: `void alt_up_character_lcd_shift_display(alt_up_character_lcd_dev *lcd, int x_right_shift_offset)`
Include: `<altera_up_avalon_character_lcd.h>`
Parameters: `lcd` – struct for the LCD Controller device
`x_right_shift_offset` – the number of spaces to shift to the right. If the offset is negative, then the display shifts to the left.
Returns: nothing
Description: Shift the entire display to left or right.

4.2.9 alt_up_character_lcd_erase_pos

Prototype: `int alt_up_character_lcd_erase_pos(alt_up_character_lcd_dev *lcd, unsigned x_pos, unsigned y_pos)`
Include: `<altera_up_avalon_character_lcd.h>`
Parameters: `lcd` – struct for the LCD Controller device
`x_pos` – x coordinate (0 to 15, from left to right)
`y_pos` – y coordinate (0 for the top row, 1 for the bottom row)
Returns: 0 for success
Description: Erase the character at the specified coordinate.

4.2.10 alt_up_character_lcd_cursor_off

Prototype: `void alt_up_character_lcd_cursor_off(alt_up_character_lcd_dev *lcd)`
Include: `<altera_up_avalon_character_lcd.h>`
Parameters: `lcd` – struct for the LCD Controller device
Returns: nothing
Description: Turn off the cursor.

4.2.11 alt_up_character_lcd_cursor_blink_on

Prototype: `void alt_up_character_lcd_cursor_blink_on(alt_up_character_lcd_dev *lcd)`
Include: `<altera_up_avalon_character_lcd.h>`
Parameters: `lcd` – struct for the LCD Controller device
Returns: nothing
Description: Turn on the cursor.

```
#include "altera_up_avalon_character_lcd.h"

int main(void)
{
	alt_up_character_lcd_dev * char_lcd_dev;

	// open the Character LCD port
	char_lcd_dev = alt_up_character_lcd_open_dev ("/dev/Char_LCD_16x2");
	if ( char_lcd_dev == NULL)
		alt_printf ("Error: could not open character LCD device\n");
	else
		alt_printf ("Opened character LCD device\n");

	/* Initialize the character display */
	alt_up_character_lcd_init (char_lcd_dev);

	/* Write "Welcome to" in the first row */
	alt_up_character_lcd_string(char_lcd_dev, "Welcome to");

	/* Write "the DE2 board" in the second row */
	char second_row[] = "the DE2 board\0";
	alt_up_character_lcd_set_cursor_pos(char_lcd_dev, 0, 1);
	alt_up_character_lcd_string(char_lcd_dev, second_row);
}
```

Figure 3. An example of C using HAL for the Character LCD core.