LCD Component

REV A

Publication Date: 2012/10/15

XMOS

Table of Contents

1 Overview

IN THIS CHAPTER

The LCD component is used to drive a single graphics LCD module up to 800 * 600 pixels with pixel clocks of up to 25MHz.

1.1 Features

- Standard component to support different LCD displays with RGB 565.
- Different color depths 32 bpp, 16 bpp, etc. based on user configuration.
- Resolution of up to 800 * 600 pixels. See table below for different screen configurations.
- Outputs to a CMOS interface.
- Configurability of * LCD pixel dimensions, * clock rate, * horizontal and vertical timing requiremnts, * port mapping of the LCD.
- Requires a single core for the server. * The function `lcd_server` requires just one core, the client functions, located in `lcd.h` are very low overhead and are called from the application.

1.2 Memory requirements

Resource	Usage
Stack	92 bytes
Program	2168 bytes

1.3 Resource requirements

Resource	Usage
Channels	1
Timers	0
Clocks	1
Logical Cores	1

1.4 Performance

The achievable effective bandwidth varies according to the avaliable XCore MIPS. The maximum pixel clock supported is 25MHz.

2 Hardware Requirements

IN THIS CHAPTER

2.1 Recommended Hardware

2.1.1 Slicekit

This module may be evaluated using the Slicekit Modular Development Platform, available from digikey. Required board SKUs are:

- XP-SKC-L2 (Slicekit L2 Core Board) plus XA-SK-SCR480 plus XA-SK-XTAG2 (Slicekit XTAG adaptor)

2.2 Demonstration Applications

2.2.1 LCD Demo Application

The LCD demo application shows how a buffer of image data can be written to the 480x272 LCD screen that is supplied with the XA-SK-SCR480 Slice Card.

- Package: sc_lcd
- Application: app_lcd_demo

2.2.2 Text Display Application

This application demonstrates how the `module_text_display` can be used to put text into the LCD image buffer for display to the Slice Card screen.

- Package: sc_lcd
- Application: app_text_display

2.2.3 Display Controller Application

This combination demo employs the `module_lcd` along with the `module_sdram` and the `module_display_controller` framebuffer framework component to implement a 480x272 display controller.

Required board SKUs for this demo are:

- XP-SKC-L2 (Slicekit L2 Core Board) plus XA-SK-XTAG2 (Slicekit XTAG adaptor)
- XA-SK-SCR480 for the LCD
- XA-SK-SDRAM for the SDRAM
- Package: sw_display_controller
- Application: app_display_controller

3 API

IN THIS CHAPTER

The component `sc_lcd` includes the module `module_lcd` and the `module_text_display`.

3.1 module_lcd

3.1.1 Configuration Defines

The `module_lcd` includes device support defines, each support header, located in the `devices` directory defines a number of parameters. It is sufficient for the user to specify which device to support in the `lcd_conf.h` for the device to be correctly supported. To do this `lcd_conf.h` must include the define: :: #define LCD_PART_NUMBER p

- AT043TN24V7
- K430WQAV4F

3.1.2 Implementation Specific Defines

It is possible to override the default defines when a part number is selected. The defines avaliable are:

LCD_WIDTH
This define is used to represent the width of the LCD panel in pixels.

LCD_HEIGHT
This define is used to represent the height of the LCD panel in terms of lines.

LCD_BITS_PER_PIXEL
Count of bits used to set a pixels colour, i.e. if the screen was wired for rgb565 then the LCD_BITS_PER_PIXEL would be 16, rgb888 would be 24. This is independant of the actual bit depth of the lcd.

LCD_HOR_FRONT_PORCH
The horizontal front porch timing requirement given in pixel clocks.

LCD_HOR_BACK_PORCH
The horizontal back porch timing requirement given in pixel clocks.

LCD_VERT_FRONT_PORCH
The vertical front porch timing requirement given in horizontal time periods.

LCD_VERT_BACK_PORCH
The vertical back porch timing requirement given in horizontal time periods.

LCD_HOR_PULSE_WIDTH
The horizontal pulse width timing requirement given in pixel clocks. This is the duration that the hsync signal should go low to denote the start of the horizontal frame. Set to 0 when hsync is not necessary.

LCD_VERT_PULSE_WIDTH
The vertical pulse width timing requirement given in vertical time periods. This is the duration that the vsync signal should go low to denote the start of the vertical frame. Set to 0 when vsync is not necessary.

LCD_FREQ_DIVIDEND
The defines FREQ_DIVIDEND and FREQ_DIVISOR are used to calculate the frequency of the clock used for LCD. The frequency configured = (FREQ_DIVIDEND / FREQ_DIVISOR) in MHz

LCD_FREQ_DIVISOR
The defines FREQ_DIVIDEND and FREQ_DIVISOR are used to calculate the frequency of the clock used for LCD. The frequency configured = (FREQ_DIVIDEND / FREQ_DIVISOR) in MHz

3.1.3 API

- `lcd.xc`
- `lcd.h`
- `lcd_defines.h`
- `/devices`

Where the following functions can be found:

`void lcd_init(chanend c_lcd)`
LCD init function.

This sets the lcd into a state where it is ready to accept data.

This function has the following parameters:

`c_lcd` The channel end connecting to the lcd server.

`static void lcd_req(chanend c_lcd)`
Receives the request for data from the LCD server.

This function has the following parameters:

`c_lcd` The channel end connecting to the lcd server.

`static void lcd_update(chanend c_lcd, unsigned buffer[])`
LCD update function.

This sends a buffer of data to the lcd server to to sent to the lcd.

Note, no array bounds checking is performed.

This function has the following parameters:

`c_lcd` The channel end connecting to the lcd server.

`buffer[]` The data to to emitted to the lcd screen, stored in rgb565.

```
static void lcd_update_p(chanend c_lcd, unsigned buffer)
```

C interface for LCD update function.

This sends a buffer of data to the lcd server to to sent to the lcd.

Note, no array bounds checking is performed.

This function has the following parameters:

`c_lcd` The channel end connecting to the lcd server.

`buffer` A pointer to data to to emitted to the lcd screen, stored in rgb565.

```
void lcd_server(chanend client, lcd_ports &ports)
```

The LCD server thread.

This function has the following parameters:

`client` The channel end connecting to the client.

`ports` The structure carrying the LCD port details.

4 Programming Guide

IN THIS CHAPTER

This section provides information on how to program applications using the LCD module.

4.1 Source code structure

Project	File	Description
module_lcd	`lcd.h`	Header file containing the APIs for the LCD component
	`lcd.xc`	File containing the implementation of the LCD component
	`lcd_defines.xc`	Header file containing the user configurable defines for the LCD
	`/devices`	Folder containing header files of configurations for LCDs

Figure 1: Project structure

4.2 How to select the LCD target

The module has been designed to support multiple LCD targets. Each target has a specific configuration and have been provided with the component int the `/devices` directory. The module only supports a single LCD target per XCore.

To select the target the following should be done:

- Create a header in the application project called `lcd_conf.h`
- In the `lcd_conf.h` add the define `#define LCD_PART_NUMBER AT043TN24V7`. This will include the "lcd_defines_AT043TN24V7.h" required for the selected target.
- Any specific overrides should be added to the `lcd_conf.h`. For example, to override the `LCD_HEIGHT` to 600 pixels add the line `#define LCD_HEIGHT 600`.
- The application should also include the port mapping for the LCD as per the hardware used. A variable of the type structure `lcd_ports` should be created and must include the port information

Example: In the application file

```
struct lcd_ports lcd_ports = {
        XS1_PORT_1G,
        XS1_PORT_1F,
        XS1_PORT_16A,
        XS1_PORT_1B,
        XS1_PORT_1C,
        XS1_CLKBLK_1
};
```

The declared variable `lcd_ports` is used by the LCD server call to address these ports. A core should have the `lcd_server` running on it and it should be connected by a channel to the application, for example:

```
chan c_lcd;
par {
      lcd_server(c_lcd, lcd_ports);
      application(c_lcd);
}
```

4.3 Executing The Project

The module by itself cannot be build or executed separately. It must be linked in to an application which needs LCD display. Once the module is linked to the application, the application can be built and tested for driving a LCD screen.

1. The module name `module_lcd` should be added to the list of MODULES in the application project build options.
2. Now the module is linked to the application and can be directly used

4.4 Software Requirements

The module is built on XDE Tool version 12.0 The module can be used in version 12.0 or any higher version of xTIMEcomposer.

5 Example Applications

IN THIS CHAPTER

This tutorial describes the demo applications included in the XMOS LCD software component. §2.1 describes the required hardware setups to run the demos.

5.1 app_lcd_demo

This application demonstrates how the module is used write image data to the LCD screen. The purpose of this application is to show how data is passed to the `lcd_server`

5.2 Application Notes

1. `lcd_server` requires a single logical core.
2. `lcd_init` must be called before any of `lcd_update`, `lcd_update_p` or `lcd_update` are called. This puts the LCD server into a state ready to accept data.
3. `lcd_update` and `lcd_update_p` are used to send an array of pixel data to the LCD server. There is a real-time requirement that this function is called often enough to maintain the display. `lcd_update_p` is the C interface to the LCD server, it takea a pointer to an array rather than the array itself.
4. `lcd_req` is a function (also a select handler) that acknoledges the LCDs request for the next line of pixel data.
5. The LCD server does no buffering of pixel line arrays, therfore, for every `lcd_req` there must be only one `lcd_update` or `lcd_update_p`. Likewise for every `lcd_update` or `lcd_update_p` there must be only one `lcd_req`.
6. The pixel array must be on the same tile as the `lcd_server`.

5.2.1 Getting Started

1. Plug the XA-SK-LCD Slice Card into the ‘STAR’ slot of the Slicekit Core Board
2. Plug the XA-SK-XTAG2 Card into the Slicekit Core Board.
3. Ensure the XMOS LINK switch on the XA-SK-XTAG2 is set to “off”.
4. Ensure the jumper on the XA-SK-SCR480 is bridged if the back light is required.

5. Open `app_lcd_demo.xc` and build the project.
6. run the program

The output produced should look like a bouncing “X” on the LCD screen.