Display controller component

REV A

Publication Date: 2013/11/12

XMOS®

Table of Contents

1 Overview

IN THIS CHAPTER

The display controller module is used to drive a single graphics LCD screen up to 800 * 600 pixels incorporating a managed double buffer.

1.1 Features

- Non-blocking SDRAM management.
- Real time servicing of the LCD.
- Touch interactive display
- Image memory manager to simplify handling of images.
- No real time constraints on the application.

1.2 Memory requirements

Resource	Usage
Stack	6198 bytes
Program	11306 bytes

1.3 Resource requirements

Resource	Usage
Channels	3
Timers	0
Clocks	0
Threads	1

1.4 Performance

The achievable effective bandwidth varies according to the available 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)
- XA-SK-SCR480 plus XA-SK-XTAG2 (sliceKIT xTAG adaptor)

2.2 Demonstration applications

2.2.1 Display controller application

- Package: sw_display_controller
- Application: app_display_controller

This combination demo employs the `module_lcd` along with the `module_sdram`, `module_touch_controller_lib`, `module_i2c_master` and the `module_display_controller` framebuffer framework component to implement a 480x272 display controller.

Required board SKUs for this demo are:

- XP-SKC-L16 (sliceKIT L16 Core Board) plus XA-SK-XTAG2 (sliceKIT xTAG adaptor)
- XA-SK-SDRAM
- XA-SK-SCR480 (which includes a 480x272 color touch screen)

3 API

IN THIS CHAPTER

- component: sc_sdram_burst which handles the SDRAM
- component: sc_lcd which handles the LCD

The below section details the APIs in the application. For details about the LCD and SDRAM APIs please refer to the respective repositories.

3.1 Configuration defines

The `module_display_controller` can be configured via the header `display_controller_conf.h`. The module requires nothing to be additionally defined however any of the defines can be overridden by adding the header `display_controller_conf.h` to the application project and adding the define that needs overridding. The possible defines are:

DISPLAY_CONTROLLER_MAX_IMAGES
: This defines the storage space allocated to the display controller for it to store image metadata. When an image is registered with the display controller its dimensions and location in SDRAM address space are stored in a table. The define specifies how many entries are allowed in that table. Note, there is no overflow checking by default.

DISPLAY_CONTROLLER_VERBOSE
: This define switches on the error checking for memory overflows and causes verbose error warnings to be emitted in the event of an error.

3.2 API

- `display_controller_client.xc`
- `display_controller_internal.h`
- `display_controller.xc`
- `display_controller.h`
- `transitions.h`
- `transitions.xc`

The display controller handles the double buffering of the image data to the LCD as a real time service and manages the I/O to the SDRAM as a non-real time service.

The display controller API is as follows:

```
void display_controller(chanend c_client, chanend c_lcd, chanend c_sdram)
```

Function to manage the LCd c_server and SDRAM server whilst maintaining image buffers.

This function has the following parameters:

`c_client` The channel from the display_controller to the client application.

`c_lcd` The channel from the display_controller to the LCd c_server.

`c_sdram` The channel from the display_controller to the SDRAM server.

```
void display_controller_image_read_line(chanend c_server,
                                        unsigned line,
                                        unsigned image_no,
                                        unsigned buffer[])
```

The function reads a line of pixel data from the SDRAM.

This function has the following parameters:

`c_server` The channel from the client application to the display_controller.

`line` The image line number to be read.

`image_no` The image number whose line is to be read.

`buffer[]` The buffer which is to be filled with the read data.

```
void display_controller_image_read_line_p(chanend c_server,
                                          unsigned line,
                                          unsigned image_no,
                                          intptr_t buffer)
```

The function reads a line of pixel data from the SDRAM.

This function has the following parameters:

`c_server` The channel from the client application to the display_controller.

`line` The image line number to be read.

`image_no` The image number whose line is to be read.

`buffer` A pointer to the buffer which is to be filled with the read data.

```
void display_controller_image_write_line(chanend c_server,
                                         unsigned line,
                                         unsigned image_no,
                                         unsigned buffer[])
```

The function writes a line of pixel data to the registered image in SDRAM.

This function has the following parameters:

`c_server` The channel from the client application to the display_controller.

`line` The image line number to be written.

`image_no` The image number whose line is to be written.

`buffer[]` The buffer which is to be written to the image.

```
void display_controller_image_write_line_p(chanend c_server,
                                           unsigned line,
                                           unsigned image_no,
                                           intptr_t buffer)
```

The function writes a line of pixel data to the registered image in SDRAM.

This function has the following parameters:

`c_server` The channel from the client application to the display_controller.

`line` The image line number to be written.

`image_no` The image number whose line is to be written.

`buffer` A pointer to the buffer which is to be written to the image.

```
void display_controller_image_read_partial_line(chanend c_server,
                                                unsigned line,
                                                unsigned image_no,
                                                unsigned buffer[],
                                                unsigned line_offset,
                                                unsigned word_count,
                                                unsigned buffer_offset)
```

The function writes a partial line of pixel data to the registered image in SDRAM.

This function has the following parameters:

`c_server` The channel from the client application to the display_controller.

`line` The image line number to be written.

`image_no` The image number whose line is to be written.

`buffer[]` The buffer which is to be written to the image.

`line_offset` The offset in pixels to begin the write of the image line from.

`word_count` The number of words to write to the image line.

`buffer_offset`
The offset from the begining of the buffer to write from in words.

```
void display_controller_image_read_partial_line_p(chanend c_server,
                                                   unsigned line,
                                                   unsigned image_no,
                                                   intptr_t buffer,
                                                   unsigned line_offset,
                                                   unsigned word_count,
                                                   unsigned buffer_offset)
```

The function writes a partial line of pixel data to the registered image in SDRAM.

This function has the following parameters:

`c_server` The channel from the client application to the display_controller.

`line` The image line number to be written.

`image_no` The image number whose line is to be written.

`buffer` A pointer to the buffer which is to be written to the image.

`line_offset` The offset in pixels to begin the write of the image line from.

`word_count` The number of words to write to the image line.

`buffer_offset`
The offset from the begining of the buffer to write from in words.

```
unsigned display_controller_register_image(chanend c_server,
                                           unsigned img_width_words,
                                           unsigned img_height_lines)
```

Registers an image with the display controller.

Returns an image handle to refer to the image from then on.

This function has the following parameters:

`server` The channel from the client application to the display_controller.

`img_width_words`
The width of the image in words.

`img_height_lines`
The height of the image in lines(pixels).

```
void display_controller_wait_until_idle(chanend c_server,
                                        unsigned buffer[])
```

Makes the display controller wait until the current SDRAM service has completed.

This function has the following parameters:

`server` The channel from the client application to the display_controller.

`buffer` The buffer which is to be written to the image.

```
void display_controller_wait_until_idle_p(chanend c_server,
                                          intptr_t buffer)
```

Makes the display controller wait until the current SDRAM service has completed.

This function has the following parameters:

`c_server` The channel from the client application to the display_controller.

`buffer` A pointer to the buffer which is to be written to the image.

```
void display_controller_frame_buffer_commit(chanend c_server,
                                            unsigned image_no)
```

Commits the image to the display controller to be displayed on the LCD screen when the current image is completly displayed.

The display controller contains a single next image number buffer meaning that if the buffer is empty (the previously commited image is already on the LCD screen) then the command will return immediatly. If the buffer is full then this function will block until the current image is on the LCD screen and the buffer is ready for a new entry. This behaviour ensures that frame commits will not overwrite.

This function has the following parameters:

`c_server` The channel from the client application to the display_controller.

`image_no` The image handle of the image to be displayed as per the described behaviour.

```
void display_controller_frame_buffer_init(chanend c_server,
                                          unsigned image_no)
```

Commits the image to the display controller to be displayed on the LCD screen and initialises the display controller.

This must only be called once at the begining of the display controllers use.

This function has the following parameters:

`server` The channel from the client application to the display_controller.

`image_no` The image handle of the image to be displayed as per the described behaviour.

The transition API is as follows:

```
unsigned transition_wipe(chanend c_server,
                         unsigned frame_buffer[2],
                         unsigned image_from,
                         unsigned image_to,
                         unsigned frames,
                         unsigned cur_fb_index)
```

Transition effect: A -> B as a wipe, i.e.

B wipes to A from the right.

This function has the following parameters:

`c_server` The channel from the client application to the display_controller.

`frame_buffer` An array of the frame buffer image handles.

`image_from` The image handle of the image to transition from.

`image_to` The image handle of the image to transition to.

`frames` The number of frame to take over the course of the transisiton.

`cur_fb_index` The initial index of the current frame in the frame_buffer .

This function returns:

The final index of the current frame in the frame_buffer .

```
unsigned transition_slide(chanend c_server,
                          unsigned frame_buffer[2],
                          unsigned image_from,
                          unsigned image_to,
                          unsigned frames,
                          unsigned cur_fb_index)
```

Transition effect: A -> B as a slide, i.e.

B slides over A from the right.

This function has the following parameters:

`c_server` The channel from the client application to the display_controller.

`frame_buffer`
: An array of the frame buffer image handles.

`image_from`
: The image handle of the image to transition from.

`image_to`
: The image handle of the image to transition to.

`frames`
: The number of frame to take over the course of the transisiton.

`cur_fb_index`
: The initial index of the current frame in the frame_buffer .

This function returns:

The final index of the current frame in the frame_buffer .

```
unsigned transition_roll(chanend c_server,
                         unsigned frame_buffer[2],
                         unsigned image_from,
                         unsigned image_to,
                         unsigned frames,
                         unsigned cur_fb_index)
```

Transition effect: A -> B as a roll, i.e.

B rolls on from the right and A rolls off to the right.

This function has the following parameters:

`c_server`
: The channel from the client application to the display_controller.

`frame_buffer`
: An array of the frame buffer image handles.

`image_from`
: The image handle of the image to transition from.

`image_to`
: The image handle of the image to transition to.

`frames`
: The number of frame to take over the course of the transisiton.

`cur_fb_index`
: The initial index of the current frame in the frame_buffer .

This function returns:

The final index of the current frame in the frame_buffer .

```
unsigned transition_dither(chanend c_server,
                           unsigned frame_buffer[2],
                           unsigned image_from,
                           unsigned image_to,
```

```
                    unsigned frames,
                    unsigned cur_fb_index)
```

Transition effect: A -> B as a dither, i.e.

B is revealed in 2 pixel chunks until A is gone.

This function has the following parameters:

`c_server` The channel from the client application to the display_controller.

`frame_buffer`
An array of the frame buffer image handles.

`image_from` The image handle of the image to transition from.

`image_to` The image handle of the image to transition to.

`frames` The number of frame to take over the course of the transisiton.

`cur_fb_index`
The initial index of the current frame in the frame_buffer .

This function returns:

The final index of the current frame in the frame_buffer .

```
unsigned transition_alpha_blend(chanend c_server,
                                unsigned frame_buffer[2],
                                unsigned image_from,
                                unsigned image_to,
                                unsigned frames,
                                unsigned cur_fb_index)
```

Transition effect: A -> B as a fade, i.e.

A fades away as B fades in.

This function has the following parameters:

`c_server` The channel from the client application to the display_controller.

`frame_buffer`
An array of the frame buffer image handles.

`image_from` The image handle of the image to transition from.

`image_to` The image handle of the image to transition to.

`frames` The number of frame to take over the course of the transisiton.

`cur_fb_index`
: The initial index of the current frame in the frame_buffer .

This function returns:

The final index of the current frame in the frame_buffer .

The transitions use the display controller API.

4 Programming guide

IN THIS CHAPTER

4.1 Shared memory interface

The display controller uses a shared memory interface to move the large amount of data around from tile to tile efficiently. This means that the `display_controller`, `sdram_server` and `lcd_server` must be one the same tile.

4.2 Source code structure

Project	File	Description
module_display_controller	`display_controller.h`	Header file containing the APIs for the display controller component.
	`display_controller.xc`	File containing the implementation of the display controller component.
	`display_controller_client.xc`	File containing the implementation of the display controller client functions.
	`display_controller_internal.h`	Header file containing the user configurable defines for the display controller component.
	`transitions.h`	Header file containing the APIs for the display controller transitions.
	`transitions.xc`	File containing the implementation of the display controller transitions.

Figure 1: Project structure

4.3 Executing the project

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

1. `module_display_controller`
2. `module_lcd`
3. `module_sdram`
1. `module_touch_controller_lib` or `module_touch_controller_server`
2. `module_i2c_master`

should be added to the list of MODULES.

4.4 Software requirements

The module is built on xTIMEcomposer 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 a demo application that uses the display controller module. §2.1 describes the required hardware setup to run the demos.

5.1 app_display_controller_demo

This application demonstrates how the `lcd_module` is used to write image data to the LCD screen whilst imposing no real time constraints on the application. The purpose of this demonstration is to show how data is passed to the `display_controller`. This application also demonstrates an interactive display using `touch_controller_lib` module.

5.2 Application notes

5.2.1 Getting started

1. Plug the XA-SK-LCD Slice Card into the 'TRIANGLE' slot of the sliceKIT Core Board
2. Plug the XA-SK-SDRAM Slice Card into the 'STAR' slot of the sliceKIT Core Board
3. Open `app_display_controller_demo.xc` and build the project.
4. Run the program ensuring that it is run from the project directory where the TGA images are.

The output produced should look like a series of images transitioning on the LCD when the screen is touched.