Display controller library

The XMOS display controller library provides the service of removing the real-time constraint of maintaining the LCDs line buffer from the application and provides a managed frame buffering service. It does this by using an SDRAM as a storage for the frame buffers.

Features

- Asynchronous non-blocking interface for modifying frame buffers,
- User configurable number of frame buffers.

Components

- Display controller

Resource Usage

This following table shows typical resource usage in some different configurations. Exact resource usage will depend on the particular use of the library by the application.

Configuration	Pins	Ports	Clocks	Ram	Logical cores
Display controller server, 2 frame buffers of 480x272 pixels	0	0	0	~10.8K	0
Display controller server, 4 frame buffers of 480x272 pixels	0	0	0	~10.9K	0
Display controller server, 8 frame buffers of 480x272 pixels	0	0	0	~11.0K	0

Software version and dependencies

This document pertains to version 3.0.0 of this library. It is known to work on version 14.0.1 of the xTIMEcomposer tools suite, it may work on other versions.

This library depends on the following other libraries:

- lib_sdram (>=3.0.0)
- lib_lcd (>=3.0.0)

Related application notes

The following application notes use this library:

- AN00169 - Using the display controller library

1 Hardware characteristics

The display controller requires use of an SDRAM and an LCD. The respective hardware requirements of these are covered in:

- SDRAM library (see XM-004440-PC),
- LCD library (see XM-004440-PC).

2 Display Controller API

All display controller functions can be accessed via the `display_controller.h` header:

```
#include <display_controller.h>
```

You will also have to add `lib_display_controller` to the `USED_MODULES` field of your application Makefile.

The display controller server and client are instantiated as parallel tasks that run in a `par` statement. The client (application on most cases) can connect via a streaming channel.

The display controller uses distributed tasks to implement bi-directional asynchronous decoupling of the commands between the application(client) and the display controller. This means that the asynchronous command buffering is handled by the interfaces:

```
interface app_to_cmd_buffer_i
interface cmd_buffer_to_dc_i
interface dc_to_res_buf_i
interface res_buf_to_app_i
```

There is one other interface that connects to the application, the vertical synchronization interface. This interface allows the application to know when the frame is at the start of a new scan, i.e. line zero is about to be written.

As the display controller uses some of the SDRAM, the memory address allocator is used to allocate an amount of the SDRAM to the display controller. See (... TODO) to find out more about how to use the memory address allocator.

For example, the following code instantiates a display controller server and connects an application to it (the SDRAM and LCD declarations has been shortened for simplicity):

```
on tile[1] : out buffered port:32   lcd_rgb                    = XS1_PORT_16B;
on tile[1] : out port               lcd_clk                    = XS1_PORT_1I;
on tile[1] : out port               ?lcd_data_enabled          = XS1_PORT_1L;
on tile[1] : out buffered port:32   ?lcd_h_sync                = XS1_PORT_1J;
on tile[1] : out port               ?lcd_v_sync                = XS1_PORT_1K;
on tile[1] : clock                  lcd_cb                     = XS1_CLKBLK_1;

on tile[1] : out buffered port:32   sdram_dq_ah                = XS1_PORT_16A;
on tile[1] : out buffered port:32   sdram_cas                  = XS1_PORT_1B;
on tile[1] : out buffered port:32   sdram_ras                  = XS1_PORT_1G;
on tile[1] : out buffered port:8    sdram_we                   = XS1_PORT_1C;
on tile[1] : out port               sdram_clk                  = XS1_PORT_1F;
on tile[1] : clock                  sdram_cb                   = XS1_CLKBLK_2;

int main() {
  interface app_to_cmd_buffer_i     app_to_cmd_buffer;
  interface cmd_buffer_to_dc_i      cmd_buffer_to_dc;

  interface dc_to_res_buf_i         dc_to_res_buf;
  interface res_buf_to_app_i        res_buf_to_app;

  interface dc_vsync_interface_i    vsync_interface;

  interface memory_address_allocator_i to_memory_alloc[1];

  streaming chan c_sdram[2], c_lcd;

  par {
      on tile[1]: [[distribute]] memory_address_allocator( 1, to_memory_alloc, 0, 1024*1024*8);

      on tile[1]: [[distribute]] command_buffer(app_to_cmd_buffer, cmd_buffer_to_dc);
      on tile[1]: display_controller(
              cmd_buffer_to_dc, dc_to_res_buf, vsync_interface,
              DISPLAY_CONTROLLER_IMAGE_COUNT,
              LCD_HEIGHT,
              LCD_WIDTH,
              LCD_BYTES_PER_PIXEL,
              to_memory_alloc[0], c_sdram[0], c_sdram[1], c_lcd);
      on tile[1]: [[distribute]] response_buffer(dc_to_res_buf, res_buf_to_app);

      on tile[1]: app(app_to_cmd_buffer, res_buf_to_app, vsync_interface);

      on tile[1]:lcd_server(c_lcd,... );
      on tile[1]:sdram_server(c_sdram, 2, ... );

  }
  return 0;
}
```

Note that the client application, display controller, LCD server and SDRAM server must be on the same tile as the line buffers are transfered by moving pointers from one task to another.

The display controller library uses movable pointers to pass buffers between the client and the server. This means that when the client passes a buffer to the display controller the client cannot access that buffer while the server is processing the command . To handle this the client sends commands using `display_controller_read` and `display_controller_write`, both of which take a movable pointer as an argument. To return the pointer to the client, the client must call the interface from the display controller (res_buf_to_app_i) using the `pop()` method which will take back ownership of the pointer when the display controller server is finished processing the command.

2.1 Client/Server model

The display controller server must be instantiated at the same level as its clients. For example:

```
on tile[1]: display_controller(
              cmd_buffer_to_dc, dc_to_res_buf, vsync_interface,
              DISPLAY_CONTROLLER_IMAGE_COUNT,
              LCD_HEIGHT,
              LCD_WIDTH,
              LCD_BYTES_PER_PIXEL,
              to_memory_alloc[0], c_sdram[0], c_sdram[1], c_lcd);
on tile[1]: app(app_to_cmd_buffer, res_buf_to_app, vsync_interface);
```

2.2 Command buffering

The display controller server implements a single slot command buffer. This means that the client can queue up a command to the display controller server through calls to `display_controller_read` or `display_controller_write`. A successful call to `display_controller_read` or `display_controller_write` will return 0 and issue the command to the command buffer. When the command buffer is full then a call to `sdram_read` or `sdram_write` will return 1 and not issue the command. Commands are completed (i.e. a slot is freed) when `sdram_complete` returns. Commands are processed as in a first in first out ordering.

2.3 Initialization

The display controller will start by displaying the image with handle 0. There is no need to initialize this frame as it will be set to all black (0x0). The application should then begin by writing to any other registered frames (image handle 1 and up).

2.4 Safety through the use of movable pointers

The API makes use of movable pointers to aid correct multi-threaded memory handling. See[1] to know more about movable pointers.

[1] https://www.xmos.com/download/public/XMOS-Programming-Guide-(documentation)(E).pdf

3 API

Function	**display_controller**
Description	The display controller server task.

Type	`void`
	`display_controller(client interface cmd_buffer_to_dc_i to_dc,`
	`  client interface dc_to_res_buf_i from_dc,`
	`  server interface dc_vsync_interface_i vsync,`
	`  static const unsigned num_frame_buffers,`
	`  static const unsigned height,`
	`  static const unsigned width,`
	`  static const unsigned bytes_per_pixel,`
	`  client interface memory_address_allocator_i mem_alloc,`
	`  streaming chanend c_sdram_lcd,`
	`  streaming chanend c_sdram_client,`
	`  streaming chanend c_lcd)`

Parameters	`to_dc` The interface for the command buffering to send commands to the display controller
	
	`from_dc` The interface for thedisplay controller to send responses to the command buffering
	
	`vsync` The interface used to indicate when a vertical restart has happened
	
	`num_frame_buffers`
	The number of frame bufferes required by the application
	
	`height` The width of each of the frame buffers(they are all be the same)
	
	`width` The height of each of the frame buffers(they are all be the same)
	
	`bytes_per_pixel`
	The bytes per pixel
	
	`mem_alloc` The interface to the memory address allocator
	
	`c_sdram_lcd`The
	streaming channel to the SDRAM server (high priority)
	
	`c_sdram_client`The
	streaming channel to the SDRAM server (low priority)
	
	`c_lcd` The streaming channel to the LCD server

Function	display_controller_read
Description	This issues a read command to the display controller.
Type	void display_controller_read(client interface app_to_cmd_buffer_i app_to_cmd_buf, unsigned *movable buffer, unsigned image_no, unsigned line, unsigned word_count, unsigned word_offset)
Parameters	app_to_cmd_buf The interface for the application to send commands to the command buffering. buffer A pointer to an array where the data should be saved to. image_no The image number to be read from. line The line number of the image to be read from. word_count The number of words to be read. word_offset The number of words from the begining of the line to begin the read.

Function	display_controller_write
Description	This issues a write command to the display controller.
Type	void display_controller_write(client interface app_to_cmd_buffer_i app_to_cmd_buf, unsigned *movable buffer, unsigned image_no, unsigned line, unsigned word_count, unsigned word_offset)

Continued on next page

Parameters	`app_to_cmd_buf`
	The interface for the application to send commands to the command buffering.
	
	`buffer` A pointer to an array where the data should be read from.
	
	`image_no` The image number to be written to.
	
	`line` The line number of the image to be written to.
	
	`word_count`
	The number of words to be written.
	
	`word_offset`
	The number of words from the begining of the line to begin the write.

Function	**display_controller_frame_buffer_commit**
Description	This schedules the given image to be displayed on the LCD at the next vertical refresh.

Type	`void`
	`display_controller_frame_buffer_commit(`
	`  client interface app_to_cmd_buffer_i from_dc,`
	`  unsigned image_no)`

Parameters	`from_app` The interface for the application to send commands to the command buffering `image_no` The image number to be commited to the display.

APPENDIX A - Known Issues

There are no known issues with this library.

APPENDIX B - Display controller library change log

B.1 3.0.0

- Consolidated version, major rework from previous display controller components