SD card component

REV A

Publication Date: 2013/12/4

XMOS

Table of Contents

1 Overview

IN THIS CHAPTER

The SD card module is used to talk to the SD card through the SPI interface.

1.1 Features

- Low level functions for accessing SD card
- Uses SPI interface

1.2 Memory requirements

Resource	Usage
Stack	220 bytes
Program	5 KB

1.3 Resource requirements

Resource	Usage
Clocks	1
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-XTAG2 (sliceKIT xTAG adaptor)
- XA-SK-FLASH 1V0 Slice Card

2.2 Demonstration applications

2.2.1 Display controller application

- Package: sc_sdcard
- Application: app_sdcard_test

This demo uses the `module_FatFs` along with the `module_sdcardSPI`, `module_spi_master`.

Required board SKUs for this demo are:

- XP-SKC-L16 (sliceKIT L16 Core Board) plus XA-SK-XTAG2 (sliceKIT xTAG adaptor)

3 API

IN THIS CHAPTER

- Configuration defines
- API

- module: module_spi_master

The below section details the APIs in the SD card module. For details about the FatFS APIs please refer to the respective repositories.

3.1 Configuration defines

The `module_sdcardSPI` requires a configuration defined in spi_conf.h. The module requires nothing to be additionally defined.

SPI_MASTER_SD_CARD_COMPAT
This defines the SPI modifications needed for SD card usage. This needs to be set to 1 for SD cards to use SPI interface.

3.2 API

- `SDCardHostSPI.xc`
- `spi_conf.h`

The SD card module provides APIs to read/write data to SD card.

The SD card APIs are as follows:

`DSTATUS disk_initialize(BYTE drv)`
Function to initialize disk.

This function has the following parameters:

`drv` Physical drive nmuber (0).

`DSTATUS disk_status(BYTE drv)`
Function to open or create a file.

This function has the following parameters:

`drv` Drive number (always 0).

`DRESULT disk_read(BYTE drv, BYTE buff[], DWORD sector, BYTE count)`

Function to read data from a disk.

This function has the following parameters:

`drv`	Physical drive nmuber (0).
`buff`	Pointer to the data buffer to store read data.
`sector`	Start sector number (LBA).
`count`	Sector count (1..128).

`DRESULT disk_write(BYTE drv, const BYTE buff[], DWORD sector, BYTE count)`

Function to write data to the disk.

This function has the following parameters:

`drv`	Physical drive nmuber (0).
`buff`	Pointer to the data to be written.
`sector`	Start sector number (LBA).
`count`	Sector count (1..128).

`DRESULT disk_ioctl(BYTE drv, BYTE ctrl, BYTE buff[])`

Function to control device specific features and miscellaneous functions.

This function has the following parameters:

`drv`	Physical drive nmuber (0).
`ctrl`	Control code.
`buff`	Buffer to send/receive control data.

The SD card APIs use the module_spi_master APIs.

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 reads and writes files on SD card using the SPI interface.

5.1 app_sdcard_test

- Demonstrate read and write files on SD card using the SPI interface. At the end, it prints the read/write performances of the filesystem on SD card using the SPI.

5.2 Application notes

5.2.1 Getting started

1. Connect XA-SK-FLASH 1V0 Slice Card to the XP-SKC-L16 sliceKIT Core board using the connector marked with the `TRIANGLE`.
2. Connect the xTAG Adapter to sliceKIT Core board, and connect xTAG-2 to the adapter.
3. Connect the xTAG-2 to host PC. Note that the USB cable is not provided with the sliceKIT starter kit.
4. Set the `XMOS LINK` to `OFF` on the xTAG Adapter(XA-SK-XTAG2).
5. Make sure the SD card slot in XA-SK-FLASH slice has a Class-4 SD card in it.
6. Switch on the power supply to the sliceKIT Core board.

5.2.2 Import and Build the Application

1. Open xTIMEcomposer and check that it is operating in online mode. Open the edit perspective (Window->Open Perspective->XMOS Edit).
2. Locate the `'SD card demo'` item in the xSOFTip pane on the bottom left of the window and drag it into the Project Explorer window in the xTIMEcomposer. This will also cause the modules on which this application depends to be imported as well.

3. Click on the app_sdcard_test item in the Explorer pane then click on the build icon (hammer) in xTIMEcomposer. Check the console window to verify that the application has built successfully.

5.2.3 Run the Application

Now that the application has been compiled, the next step is to run it on the sliceKIT Core Board using the tools to load the application over JTAG (via the xTAG-2 and xTAG Adapter card) into the xCORE multicore microcontroller.

1. Select the file `app_sdcard_test.xe` in the `app_display_controller_demo` project from the Project Explorer.

2. Click on the `Run` icon (the white arrow in the green circle).

3. At the `Select Device` dialog select `XMOS xTAG-2 connect to L1[0..1]` and click `OK`.

4. The application starts executing and reads/writes contents into SD card.