MicroSD_Controller Specification

Author: Christopher Casebeer

Christopher.casebee1@msu.montana.edu

Rev. 1.0

June 26, 2014

Revision History

Rev.	Date	Author	Description
1.0	06/13/2014	[Casebeer]	Initial Release

Introduction

MicroSD_controller and its sub entities are a system to write serial amounts of data to a microSDXC card. The controller is written in VHDL for use with an FGPA and has been developed with the Altera toolchain. Development utilized a Cyclone V FPGA.

MicroSD_controller accepts data from the host and writes that data to an attached SD card. The user does not need knowledge of the SD card protocol as this is wrapped in abstraction. The user only needs to clock data to the component appropriately and the data will be readable off the SD card. The microSD_controller was written specifically for a data logging application where an FPGA was the center of the platform.

Through use of this component the user can dump blocks data to the SD card beginning at a desired address. The user will notify the card how much data the host will send to the SD card. The component will indicate to the host success on write.

The component does not have any utility to function with or operate on a filesystem. The component strictly dumps data onto an attached SD card in a linear fashion from a presented start address. Data presented to the component will show up on the card.

SD Card Intro

The SD card has 8 physical connections of interest. The card must receive 3.3V power and ground. The card must also communicate with the host. SD card communication is done through a custom command response protocol defined by the SD Group in the SD Specifications document. The lines which are involved in communications are the clock, command, and data lines. The clock line provides a clock to the SD card which is used to time command and response alignment as well as clock the card's own internal logic. The command line is a bidirectional line. It handles all command and response transfers. Commands handle aspects such as requesting a write, read or erase. It is not involved in any data movement however. All data bits flow over the data bidirectional data lines. There are four data lines. The lines are often referred to as clk, cmd, and dat0 through dat3. Data flows in parallel over the four data lines bi-directionally in the case of reading or writing.

Data is addressed and handled in blocks. A block is 512 bytes. The SD card uses blocks as the minimum amount of addressable data. Thus the user of the component sends data to the card in 512 byte chunks. The SD Protocol consists of numerous commands designated by number. In this design the multiblock write command 25 is central to writing data to the card. To write data to the card the component must send commands to the SD card over the command line appropriately, getting the card past initialization and into the appropriate state. Users of this component must only clock data to the component appropriately and the data will be written to the card.

SD card communications can take many forms at many different speeds. The SD protocol allows up to 208Mhz along with double data rate configurations. This evolution of the component

targets a 50 Mhz single data rate mode where all data lines are used and signaling occurs at 1.8V. The component operates in 4 bit mode and at 1.8V signaling levels by default. The 1.8V signaling level requires external circuitry detailed later.

Architecture

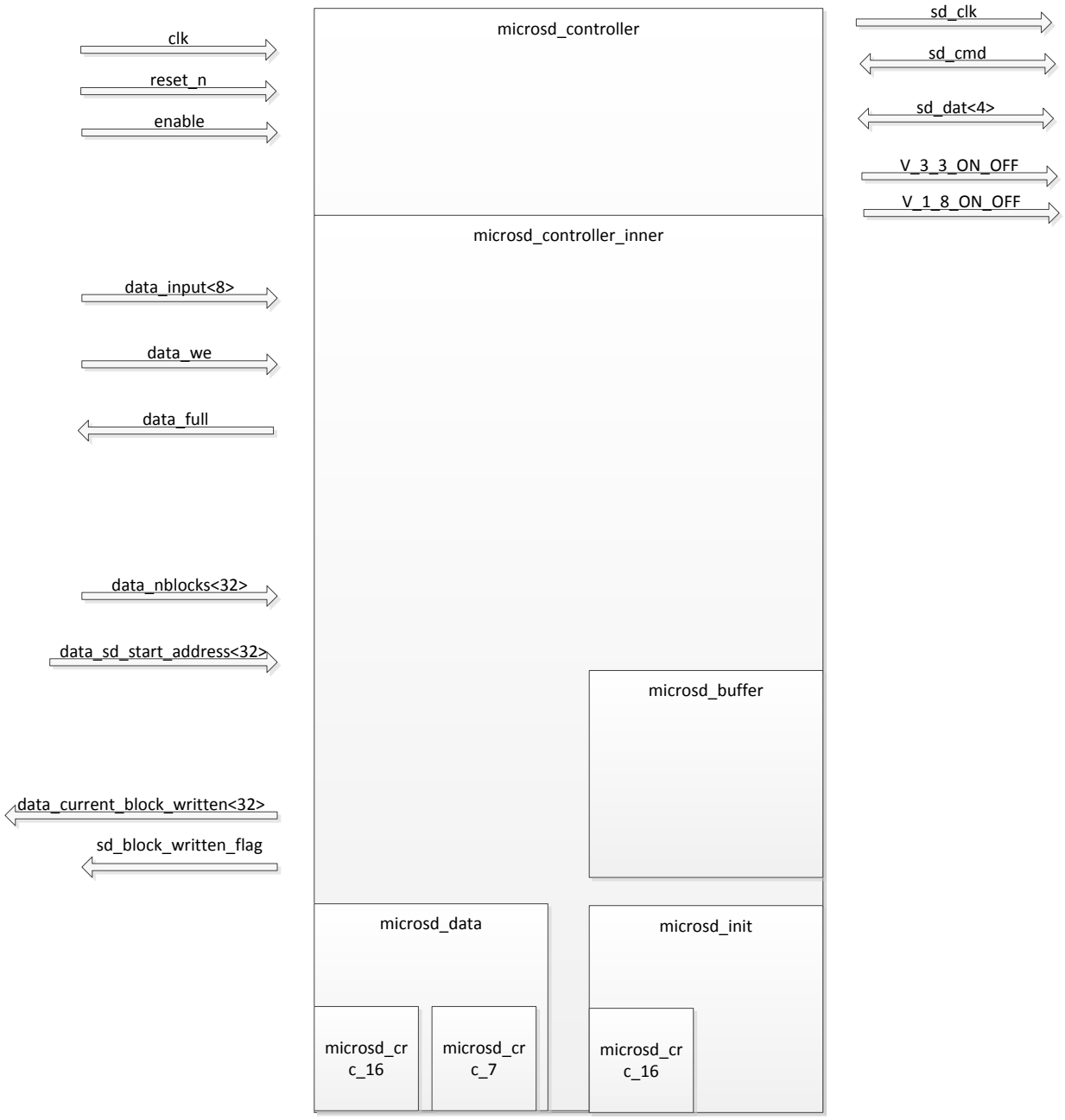

Figure 1 microsd_controller

Microsd_controller

Microsd_controller is the top level component the user will interact with. This top component instantiates microsd_controller_inner. It also tri-states the cmd and data lines depending on the state of write enable signals coming from either microsd_init or microsd_data. This level also handles the microsd_data core control in relation to the internal data buffer. For example, if the buffer has data ready to be written, microsd_controller engages microsd_data to write that data.

Microsd_controller_inner

This inner component has two jobs. One is to create the 400 kHz initialization clock through a clock divider. Microsd_controller_inner also instantiates the two lower components microsd_data and microsd_init and switches between them depending on if initialization has completed.

Microsd_init

This core is responsible for issuing all commands related to SD card initialization. This component is responsible for all commands up through issuing CMD3 to the SD card. The SD card physical specification contains diagrams partitioning what falls into initialization. The microsd_init core also makes decisions on initialization steps based on the desired operating modes. During initialization is when 1.8V signaling switch occurs.

Microsd_crc_7

This component contains the logic required for generating the crc7 checksum that is associated with all commands issued to the SD card over the command line. The code herein is a vhdl port of Kay Gorontzi's online crc generator and LGPL code.

Microsd_data

This component is responsible for all data related communications with the SD card. The core is currently capable of exercising a CMD25 multiblock write to steam continuous 512 byte blocks to the card. The number of blocks in any multiblock stream has been coded to be 128 blocks or less depending on the data_nblocks top I/O. Other requirements of writing data such as switching into 4 bit mode and tracking data crc16 appends is handled here. The core also contains partial read and erase functionality.

Microsd_crc_16

All data blocks sent to and received from the card over the data lines are protected by crc16 checksum. This engine generates the checksums attached to blocks of data sent to the card. The code herein is a vhdl port of Kay Gorontzi's online crc generator and LGPL code.

Microsd_buffer

This component is a data buffer where data is buffered by the host while the card is busy writing other data in the buffer. The host will buffer data here quicker than the card can write the data. The microsd_data core takes its data from microsd_buffer. This buffer is expandable in size via top level generic. The default size is 2048 bytes or 4 blocks. An Altera 2 port ram is used here. Level mark and read and write pointers track the buffer status. The component tracks the number of blocks that have moved through the buffer in relation to the number of blocks expected.

Data_buffer uses an Altera 2 port ram megafunction. Users of other tools chains should substitute this 2 port ram with one from their own vendor. This is the only Altera specific code in the core project.

Operation

Flow:

- Clock the design with a 400kHz to 50Mhz clock on clk input port
- Set data_sd_start_address and data_nblocks inputs to appropriate values
- Begin clocking in data on data_input on the rising edge of data_we
- Halt data_we and data_input on data_full high flag
- Repeat previous steps for another data series upon having sent data_nblocks and data_full returning low.

Figure 1 shows the top level abstraction of the design and the ports mentioned in further descriptions. The user should clock the design with a 400kHz to 50Mhz clock. This clock rate will be the rate at which internal logic is clocked and the rate at which data is transferred to the card. The user should present the component with the number of blocks (512 bytes) to be written, data_nblocks, and the start address on the card where the blocks will begin to be written, data_sd_start_address.

The user can then clock in data bytes on data_input on the rising edge of data_we. The user supplies data_we in relation to their input data. Data_full flag will go high when the internal buffer is full and the component can no longer accept data. The user should stop presenting data immediately and gate data_we. Upon receiving the number of blocks data_nblocks, the component will keep the data_full line high until all data is flushed from buffer and written to the card. The component relies on the idea that data_nblocks amount of data will come from the host. Upon finishing writing all of data_nblocks the component will reset the internal buffer and wait for another set of inputs and first rising edge of data_we. The component samples data_sd_start_address and data_nblocks near the first rising edge of data_we after a previously successful transfer or upon first power up.

The enable input going low will gate clk upon completion of writing the current command 25 stream of data. The reset input going low will cause the card to reinitialize and wait for another set of data_input ,data_nblocks, and data_sd_start_address. The card will rerun its initialization procedure upon reset.

The microSD card uses 6 lines to communicate with the host, in this case an FPGA. The lines are clock, command, and Ddata 0 through 3. The associated I/O of the component are labeled sd_clk,sd_cmd and sd_dat respectively. The microSD card also requires power (3.3V) and ground. The appropriate data, clock, and command lines are tri-stated internally in this component as these lines are bidirectional. The user of this component must take these inout lines and tie them to bidirectional pins at the top of their design. The physical pins must then be connected to the equivalent physical pins of the SD card through a bidirectional voltage level translator. This is detailed later.

The component was tested from 400kHz to 50Mhz successfully. An example synopsis design constraint file (.sdc) is included that was used to constrain the timing of the design successfully. Timing constraint is critically important when using this component on an FPGA. In particular is input and output port constraint correctness.

The component writes data via CMD25 of the SD Card command set. CMD25 is a streaming write command. It is called the multiblock write. It was found that write speed and efficiency went up dramatically as the number of blocks streamed after any CMD25 increased. This was tested up to 128 blocks which resulted in the greatest write speed of ~11MB/s. Thus if the user specifies greater than 128 blocks to write to the card, the data will be sent in 128 block sets. The component will decrease the blocks sent in the CMD25 stream if data_nblocks is less than 128. However, this will result in a slower write time.

The component makes use of an internal buffer. The buffer allows temporary storage of data to be written before it is written to the SD card. The buffer size can be specified. The buffer makes use of Altera specific 2 port ram.

The operation of data_current_block_written and sd_block_written_flag allow the user to log the last block (512 bytes) which was written successfully. On the rising edge of output sd_block_written_flag the data_current_block_written can be sampled.
Data_current_block_written signifies the last block written successfully to the card.

Three generics exist for microsd_controller. BUFSIZE specifies the size of the internal buffer. It must be a multiple of 512 bytes. HS_SDR25_MODE is a single bit which should correspond to the frequency of the input clock. The generic should be set 1 when operating above 25Mhz. This bit changes the initialization process of the card via CMD6, setting the card into a higher speed mode. CLK_DIVIDE specifies a divide by value used to divide the clock input down to ~400kHz. In the case of a 50Mhz clock, a 128 CLK_DIVIDE is used.

The SD card has several other functional parameters. One of these is the signaling level of the data, command and clock lines. The signaling level can be either 1.8V or 3.3V. This component in its default state is set to switch into 1.8V communication. This will require a level translator IC and voltage switches to accomplish. This is necessary as the SD card always begins communication at 3.3V during the very start of initialization. 1.8V is also required for communication above 50Mhz.

The components used for the 1.8V switch were as follows:
TPS22966 Dual-Channel, Ultra-Low Resistance Load Switch
http://www.ti.com/lit/ds/symlink/tps22966.pdf
TXB0106 6-Bit Bidirectional Voltage-Level Translator
http://www.ti.com/lit/ds/symlink/txb0106.pdf

The microsd_controller top level component provides 1.8V and 3.3V OnOff output signals. These are for ultimately controlling the voltage level translator. On the TI translator used, the V_{cca} pin voltage controls the destination voltage levels. V_{cca} in this configuration controls the voltage levels between the level translator and SD card. In this implementation, V_{cca} of the level

translator was tied to both outputs of a dual switch. This switch was then used to enable either 1.8V or 3.3V to the V_{cca} port of the level translator, effectively switching from 3.3V signaling to 1.8V signaling to the SD card. The FPGA output pins stayed at 3.3V signaling levels throughout. The following diagram schematically shows the design which was used.

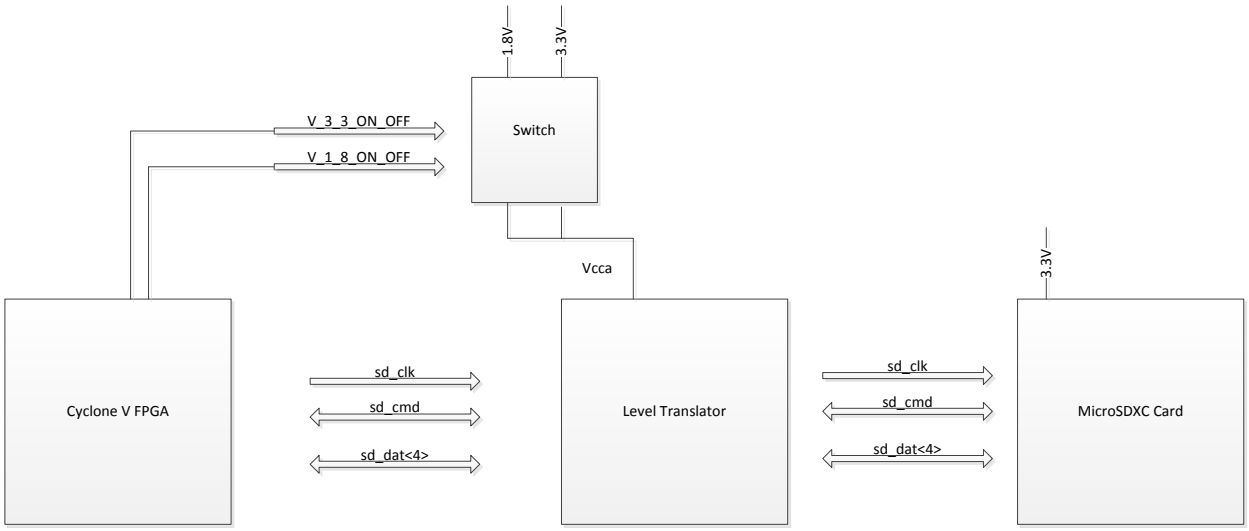

Figure 2 Level Translation Setup

The SD card can communicate with 1 data line or 4 data lines. By default the 4 bit mode is used for increased throughput.

This component was developed on a SanDisk Ultra microSDXC UHS-1 64GB card.

Registers

This design has no addressable registers. Data which is presented to the component will be written to the card. A software or bus interfacing register set is not part of the design.

Clocks

Name	Source	Rates (MHz)		Description
		Max	Min	
clk	Input	50	.4	Data system and data transmit clock
clk_400k_signal	Divide on clk	~.4	~.4	Init system clock

Table 1: List of clocks

IO Ports

Port	Width	Direction	Description
clk	1	Input	Clock Input
reset_n	1	Input	Reset Input
clock_enable	1	Input	Clock Enable
data_input	8	Input	Byte to be written to SD card
data_we	1	Input	Data_input write enable
data_sd_start_address	32	Input	Starting address on SD card for data write
data_nblocks	32	Input	Number of blocks(512bytes) to be written
data_current_block_written	32	Output	Address of last block successfully written
sd_block_written_flag	1	Output	Data_current_block_written is valid
sd_clk	1	Output	SD card clock line
sd_cmd	1	InOut	SD card command line
sd_dat	4	InOut	SD card data lines
V_3_3_ON_OFF	1	Output	3.3 V On/Off control to level shifter
V_1_8_ON_OFF	1	Output	1.8 V On/Off control to level shifter
init_start	1	Input	Debug: Init on button press
user_led_n_out	4	Output	Debug: FSM states to leds

Table 2: List of IO ports

Verification

The verification process uses a top level vhdl file to exercise microsd_controller. Two verification vhdl files are included with the design. One file writes 2048 blocks to the card in one large data_nblock. The other writes 128*16 nblocks through repeated interaction with microsd_controller. This is done by issuing a 16 data_nblock input to the component 128 times. Both tests uses a 512 byte ram initialized with random hex values. The mif file used to initialize this ram was created with Matlab. The Matlab script generates both a mif file for use with Quartus and a bin file for use with the unix compare function for verification. The top level test files write a total of 1 MB of data to the SD card. The bin file for use with cmp is also 1MB in size.

The data written to the card was verified by using the dd command at the termianl inside of a virtual machine running Debian linux. The command “dd if=/dev/sdb of=/home/chrisc/desktop/sd_test count= 2048”, was used to dump the first 2048 blocks for examination. The command “cmp sd_test 512bytecount_rand.bin” returns 0 if the two files are the same and the write was successful.

To Do

The CMD25 multiblock write pathway is the major focus of development so far. The CMD25 4 bit pathway is the only portion of the design currently utilized. However a framework and partial design exists for reading from the card and erasing the card. The single block read, single block write and erase have all been implemented and tested working previously. The finite state machine paths and associated processes for these functions need maintenance to bring them back to working order. Multiblock read however has never been implemented and would need more work than the others.

Currently the component does not have error handling capabilities. The design assumes a specific SDXC card and an error free transmission environment. An error logging system could also be implemented to log certain errors in SD communications.

To Do:

- Erase
- Single Block and Multiblock Read
- Single Block Write
- Programming Interface
- Error and Retry Handling
- Multi SD Card Bus Management
- Communication at over 50Mhz

References

SD Group. *SD Specifications Part 1 Physical Layer Simplified Specification Version 3.01 May 18, 2010.*

SandDisk Corporation. *SanDisk SD Card Product Manual Version 2.2 Document No. 80-13-00169 November 2004*

Matt Ownby. *The mysterious SD card "CRC status response"*
http://my-cool-projects.blogspot.com/2013/02/the-mysterious-sd-card-crc-status.html. 2013.

Bacchi Raffaele, Xcore SD Card Library, (2012), https://github.com/xcore/sc_sdcard.

Kay Gorontzi. CRC Generation Unit - Linear Feedback Shift Register implementation
(c) Kay Gorontzi, GHSi.de, distributed under the terms of LGPL
http://ghsi.de/CRC/index.php?Polynom=10001001. 2005.

Kay Gorontzi. CRC Generation Unit - Linear Feedback Shift Register implementation
(c) Kay Gorontzi, GHSi.de, distributed under the terms of LGPL
http://ghsi.de/CRC/index.php?Polynom=10001000000100001. 2005.