OpenCores
www.opencores.org

SDC/MMC Controller

Design Document

Author: [Adam Edvardsson]
[adam@orsoc.se]

Rev. [0.1]
May 7, 2009

WISHBONE COMPATIBLE

This page has been intentionally left blank.

Revision History

Rev.	Date	Author	Description
0.1	4/05/2009	Adam E	First Draft

Contents

1. Introduction

The "sd card

1.1 SD/MMC controller introduction

The "SD/MMC card controller" is an SD/MMC communication controller IP core. It connects to the SD card on one side and to the wishbone SoC bus on the other. The cores has been designed with the goal that a SD card controlled with the controller should be usable as a system disk contain a file system.
Therefore the core has been developed with features a system with operative system will benefit from, as DMA, interrupts and buffered write/readings. The design also include a simplified model of a SD-card to run in test bench.

1.2 Features

The following lists the main features of the SD/MMC controller IP core:

- 32-bit Wishbone Interface
- DMA
- Buffer Descriptor
- Compliant with SD Host Controller Spec version 2.0
- Support SD 4-bit mode
- Interrupt-on-completion of Data and Command transmission
- Write/Read FIFO with variable size
- Internal implementation of CRC16 for data lines and CRC7 for command line

1.3 SD/MMC controller IP Core Directory Structure

Following picture shows the structure of directories of the core.

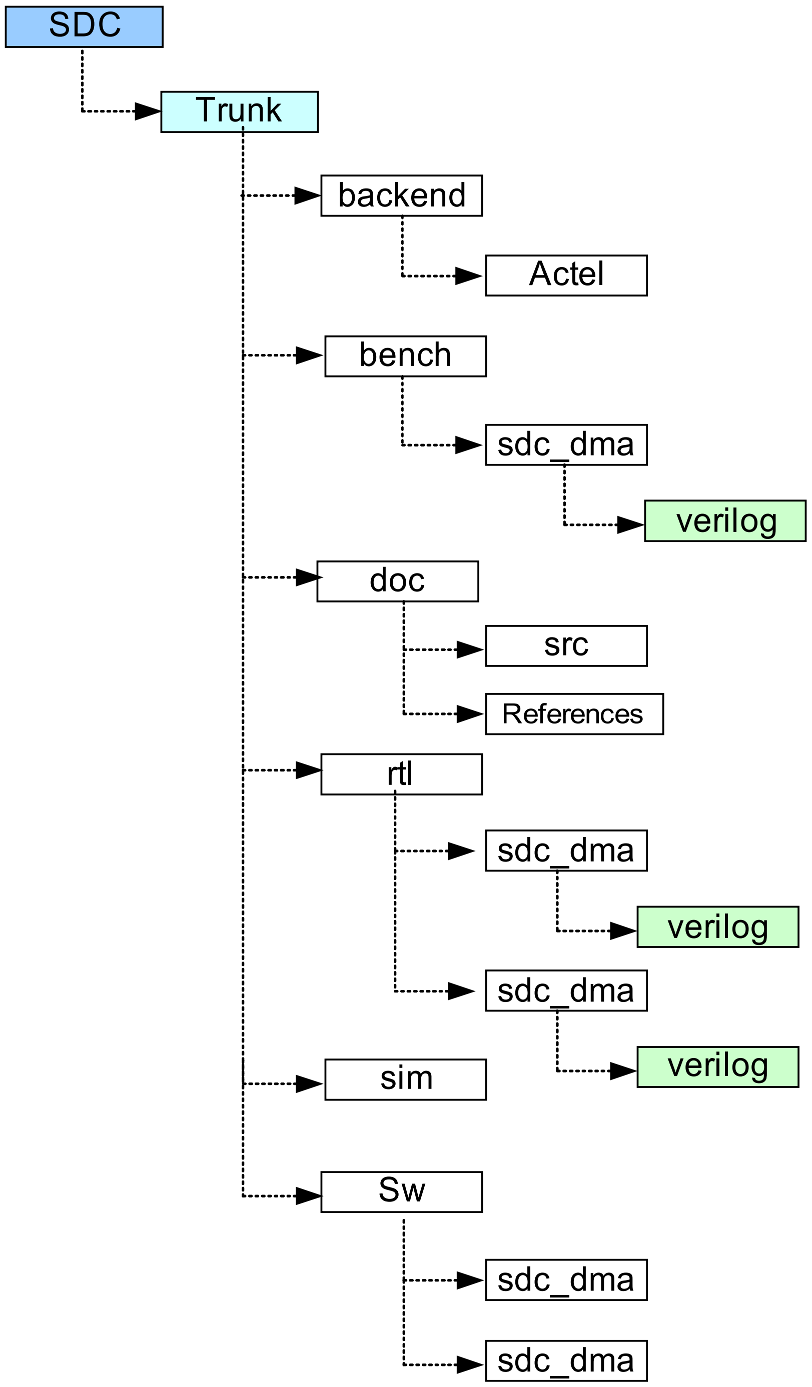

Fig 1: SD/MMC controller Core Directory Structure

There are two major parts of the Verilog code in the sdcard_mass_storage_controller (SDC) directory. First one is the code for the SD/MMC controller IP core.
The Verilog files are in the **trunk\rtl\sdc_dma\verilog** subdirectory. The second one is the code for the Testbench. These files are used together with files for the SD/MMC controller. The Verilog files are in the **trunk\bench\sdc_dma\verilog subdirectory**.

The documentation is in the subdirectory **trunk\doc**. Documentation consists of Ethernet Specification SDC_MMC controller.pdf and Design SDC_MMC controller.pdf

Backend containts Vendor specific floorplan, place and route directory structure

A software example can be found in **trunk\SW\sdc_dma** the folder contains the software for the controller sd_controller.c and sd_controller.h aswell as a testporgram main.c and a comple software to be run on a ORPSoC.

2. SD/MMC controller

2.1 Overview

The SD/MMC controller IP Core consists of 7 Major modules,
Host Interface, CMD Master, CMD Host, Data Master, Data Host, BD structure,FIFO buffer filler. Many of these modules have sub-modules. Module and submodule operations are described later in this section.

2.1.1 Host Interface

Consists of both master and slave interfaces and connects the core to the WISHBONE bus. Master interface is used for storing the received data block to the memory and loading the data that needs to be sent from the memory to the SD/MMC card. Interface is WISHBONE Revision B.2 compatible.

2.1.2 CMD Master

The SD CMD Master module synchronize the communication from the host interface with the physical interface

2.1.3 CMD Host

This module is the interface towards physical SD/MMC cards command pin. This module takes care of the physical sending and receiving of the messages,preamble generation, padding, adding start bits, stop bits, CRC etc..

2.1.4 Data Master

The SD Data Master module synchronize and initiate the data transmission from the host interface with the physical data interface.

2.1.5 Data Host

This is the interface towards physical SD card device Data port, it takes care of the physical sending and receiving of the data, preamble generation, padding, adding start bits, stop bits, CRC etc.

2.1.6 BD

The transmission and the reception processes are based on the descriptors.

2.1.7 FIFO Tx/Rx Filler

This module works as the DMA it manager the receive and transceiver FIFO buffer for the data stream. It keeps track of the status of the FIFO:s if somethings goes wrong, like full receiver FIFO or empty transfer buffer it signals this.

2.2 Description of Core Modules

The module **sd_controller_top.v** consists of sub modules **sd_cmd_master.v, sd_cmd_host.v sd_data_master.v, sd_bd.v, sd_fifo_filer_rx.v, sd_fifo_filer_tx.v** and some logic for synchronizing, multiplexing and registering outputs. All modules and their submodules aswell as their relation can be seen in figure 2.

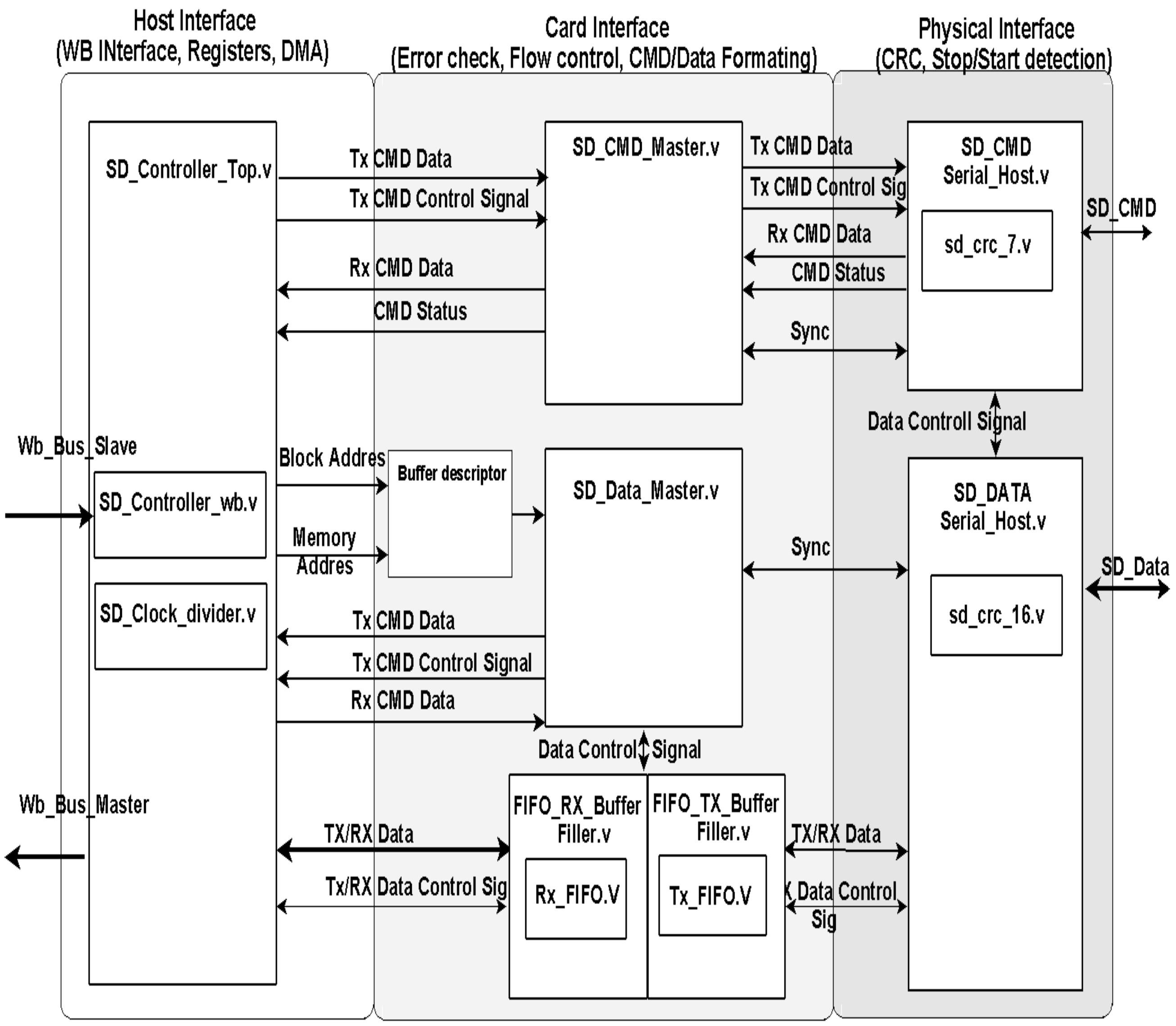

Fig 2: Core modules

2.2.1 Description Sd_CMD_ serial_host.v

The Sd_CMD_ serial_host.v module is an interface to the external SD /MMC card. It is used to read and write command to the SD/MMC card. The external interface consist of two signals clk and a bidirectional signal CMD. The cmd_out_o, cmd_dat_i and cmd_oe_o signals has to be combined in additional module (Preferable the SoC top module).

2.2.1.1 Port declaration

Direction	Width	name	Description
input	1	SD_CLK_IN	CLK
input	1	RST_IN	Synchronous Reset Active high
input	16	SETTING_IN	Define setting for current command
input	1	GO_IDLE	Reset and go to idle state
input	40	CMD_IN	CMD to be sent
input	1	REQ_IN	Request for service
input	1	ACK_IN;	ACK on service completion
input	1	cmd_dat_i	CMD from SD card
output	40	CMD_OUT	CMD reply
output	1	ACK_OUT	ACK on service completion
output	1	REQ_OUT	Request for service
output	16	STATUS	Status of the module
output	1	cmd_oe_o	Tri-state CMD Output enable
output	1	cmd_out_o	CMD to SD card
output	2	st_dat_t	Start data transfer “01” Start Read block “10” Start Write block “11” Stop

Table 1: Sd_CMD_ serial_host.v port declaration

2.2.1.2 Signal Description

Because the data cross a clock domain the signals is synchronized with **REQ** and **ACK** signals. **REQ_IN** is set to high when a service is requested, the module answer with setting **ACK_OUT** to low, as long as **ACK_OUT** is high the **SETTING_IN** and **CMD_IN** should not change. **ACK_OUT** is kept low until a **CMD** cycle is completed. When the Module has data to send (like updated **STATUS** or valid data in **CMD_OUT)** the module assert **REQ_OUT** and wait for **ACK_IN** to go high before doing any further operation.

The **Go_IDLE** signal is used to reset the module and put it into idle state, used to cancel a transfer.

SETTING_IN - Bit Description

Bit	**15**	**[14:13]**	**12**	**11**	**[10:8]**	**7**	**[6:0]**
Width	1	2	1	1	3	1	7
	Reserved	Word Select	Block Read	Block Write	Timing Values	CRC-Check ON/OFF	Response Size

Table 2: CMD_ serial_host Setting_in register

STATUS- Bit Description

Bit	[15:7]	6	5	4	[3:0]
Width	8	1	1	1	4
	Reserved	Data Available	CRC-valid	CMD status L/H	State

Table 3: CMD_ serial_host status register

2.2.1.3 Operation

The module consist of 6 block, FSM_COMBO, REQ_SYNC, ACK_SYNC, COMMAND_DECODER, FSM_OUT and FSM_SEQ.
FSM_COMBO is combinatorial logic to calculate the next state of the FSM, FSM_SEQ is the sequntial part of the FSM and it sets the state synchronized with the clock. REQ_SYNC and ACK_SYNC is 2 flipflop used to reduce chance for metastable state when signals cross clock domains. COMMAND_DECODER read the SETTING_IN vector and store the settings and command to internal register, which is used in the FSM_OUT. The FSM_OUT is the output logic of the FSM as can been seen in figure 3.

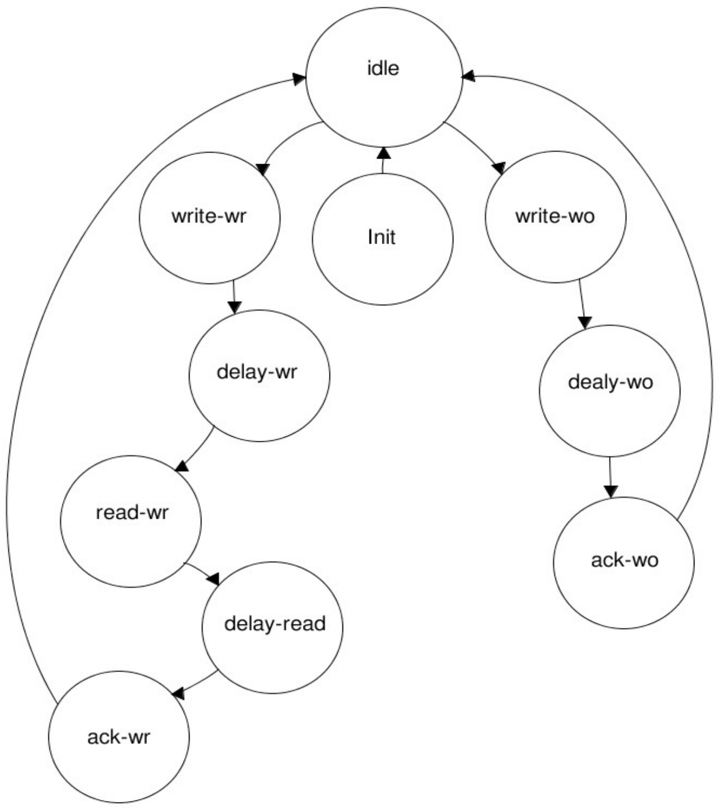

Fig 3: FSM CMD Serial Host

2.2.1.3.1 FSM State transitions

The following state transitions occurs when given condition is fulfilled

Init → Idle

Cmd_ counter >= INIT_DELAY

Idle → Write_WR

Write_Read = 1 (Response size >0)

Idle→Write_WO

Write_Only = 1 (Response size == 0)

Write_WR → Delay_wr

Sent_counter>= 47

Write_WO→Delay_wo

Sent_counter>= 47

Delay_WR → READ_WR

cmd_dat_i == 0

Delay_WO → ACK_WO

Delay_counter>= Delay_cycler

READ_WR → Delay_Read

Receive_counter > Response_size

ACK_WO → IDLE

Any event.

Delay_Read → Ack_WR

ACK_IN == 1

ACK_WR->IDLE

Any event

2.2.1.3.2 FSM State operation

Init:

Keep command line high for 64 cycler.

Idle State:

Reset the CRC and pause it. Reset counters that been used previously.

Write_WR:

1. Enable CRC.
2. Increase Sent_counter for each bit
3. For bit 0 til 39 assign data to OUT_D from the In_Buffer
4. Pause CRC
5. Bit 40 to 46 is assign data to OUT_D from CRC_out
6. The last bit 47 is always 1 (End of transmission)
7. If block_read is enabled set st_dat_t to “10”

Write_W0:

Same as Write_WR

Delay-WR

1. Disable and reset CRC
2. Reset sent counter
3. Put CMD to High'Impedance 'Z' (By setting enable to 0)
4. Increase Delay_counter

Delay-WO

Same as Delay-WR

ACK_WO

1. Set FSM_ACK to 1

Read-WR

1. Enable CRC.
2. Increase Receive_counter for each bit
3. Check Word Select setting bit and put read data to Out_buff
4. Pause CRC unit when (Response size - Receive_counter <=6)
5. Store the next 7 bit to CRC_IN
6. If CRC check enabled compare CRC_IN with the CRC value from CRC unit
7. If not equal set Out_buff to zero and set CRC error status bit
8. If block_write is enabled set st_dat_t to “01”

Dly_read

1. Disable and reset CRC
2. Reset Receive counter
3. Put CMD line to High'Impedance 'Z' (By setting enable to 0)
4. Set REQ_Out to 1
5. Assign CMD_OUT the value from Out_Buff

ACK_WR
1. Set FSM_ACK to 1
2. REQ_Out 0
3. Reset CMD_OUT

2.2.2 Description Sd_data_ serial_host.v

This module is the interface towards physical SD card device Data port. The external interface consist of two signals clk and a bidirectional signal DAT. The DAT_oe_o, DAT_dat_o and DAT_dat_i, signals has to be combined in additional module (Preferable the SoC top module).

The module perform the following actions.
• Synchronized request for write and read data and .
• Adding a CRC-16 checksum on sent data and check for correct CRC-16 on received commands.

2.2.2.1 Port declaration

Direction	Width	name	Description
input	1	sd_clk	CLK
input	1	rst	Reset
input	32	data_in	FIFO data in
input	2	start_dat	Start data transfer “01” Start Read block “10” Start Write block “11” Stop
input	1	ack_transfer	ACK on transm_complete
output	4	data_out	FIFO data out
output	1	we	FIFO WriteEnable
output	1	DAT_oe_o	Tri-state Output enable
output	4	DAT_dat_o	SD Data output
input	4	DAT_dat_i,	SD Data input
output	1	rd	FIFO read enable
output	1	busy_n	Data line Busy Active Low
output	1	transm_complete	Transmission complete

output	1	crc_ok	CRC checksum ok

Table 4: Sd_Data_ serial_host.v port declaration

2.2.2.2 Signal Description

The **data_in** signal is the data from the Tx_FIFO and **data_out** is the data going to the Rx_Fifo. The **we** (write) and **rd(read)** signals is used to control the read and writing to the FIFO:s. The signal **busy_n** is 1 when the module is in idle state else its 0.

When a transfer is completed (block read/block write) the module assert the **transm_complete,** during the assertion of this signal the value of **crc_ok** and **busy_n** is unchanged, the signal stay asserted until a **ack_transfer** is received.

2.2.2.3 Operation

The module consist of 5 blocks, ACK_SYNC, FSM_COMBO, START_SYNC, FSM_OUT and FSM_SEQ. FSM_COMBO is the combinatorial logic to calculate the next state of the FSM, FSM_SEQ is the sequential part of the FSM and it sets the state synchronized with the clock. FSM_OUT is the output logic process for the FSM, ACK_SYNC and START_SYNC dual flipflop to reduce the chances for metastable states when the signals cros the clock domain.

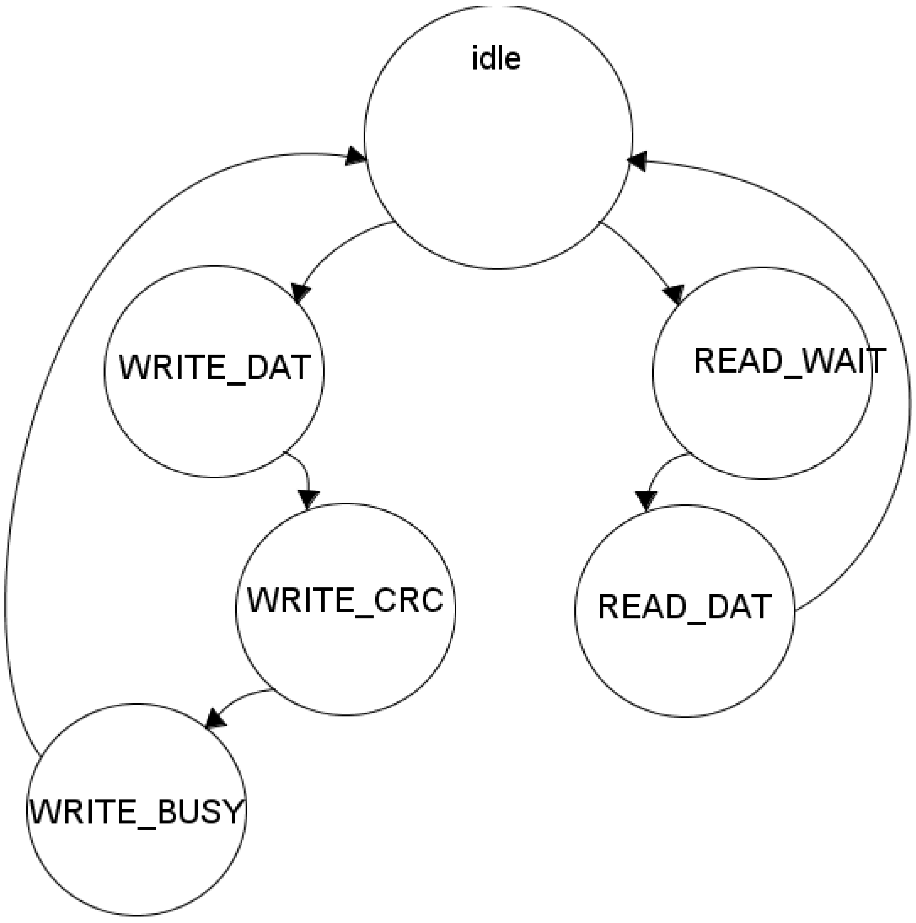

Fig 4: FSM Data Serial Host

2.2.2.3.1 FSM State transitions

Idle → WRITE_DAT
start_dat == 2'b01
Idle→ READ_WAIT
start_dat == 2'b10

READ_WAIT→ READ_DAT
q_start_bit == 0 (gated DAT_dat_o[0])
WRITE_DAT→ WRITE_CRC
transf_cnt >= Bits in a block (+CrC bits)

READ_DAT→ IDLE
ack_transfer_int == 1 (acknowledgment on transm_complete)
or
start_dat == 2'b11
WRITE_CRC→ WRITE_BUSY
crc_status == 0 (Size of CRC Response token)

WRITE_BUSY→ IDLE
dat[0] == 1 and ack_transfer_int == 1 (card not busy and acknowledgment on transm_complete)

2.2.2.3.2 FSM State operation

IDLE:
Reset the CRC and pause it. Reset counters that been used previously.

WRITE_DATA:

1. Fill the inbufferts “write_buf_0” and “write_buf_1” with data from FIFO
2. Set the outputbuffert “sd_data_out” to point at the inbuffert the out_buff_ptr points at.
3. Send Startbit → dat<=0;
4. Read 4 bits from the outputbuffert “sd_data_out” and assign to last_din and crc_in
5. Assign value of last_din to DAT_dat_o, (this makes the card lay 1 step behind CRC unit)
6. When 28 bit have been sent from outputbuffert, increase out_buff_ptr and read in a new value to sd_data_out from a inbuffert.
7. Repeat (1-7) until 512 bytes has been sent
8. Attach a 16 bit CRC to each data line
9. End with stop bit

WRITE_CRC:
Read the CRC response token, 7 cycler. Ignore the 3 first cycler 2 delay and 1 start bit. Save bit 4 to 6 to crc_s. Read bit nr 7 the stopbit.

WRITE_BUSY:

1. Signal for transm_complete
2. Check the CRC response set crc_ok.
3. Poll DAT_dat_i[0] to sense whenever the card is busy

READ_WAIT:
Prepare for data reception, enable crc units, disable output enable, and set up internal control register.

READ_DAT:

1. Read DAT_dat_i and store to FIFO data_out and crc_in
2. Increase the transfercounter

3. Repeat 1-2 until 512 bytes been received
4. Compare receiving bits with all crc_out units
5. When crc mismatch set crc_ok<=0
6. Set transm_complete when 16 CRC bit has been read

2.2.3 Description Sd_crc_ 7.v

Shift register implemented CRC-7 Checksum calculator with the polynomial, x^7+x^3+1

2.2.4 Description Sd_crc_ 16.v

Shift register implemented CRC-7 Checksum calculator with the polynomial, $x^{16}+x^{12}+x^5+1$

2.2.5 Description Sd_cmd_master.v

The SD CMD Master module synchronize the communication from the host interface with the physical interface . perform has three main tasks:
•Read a set of register from the user accessible register in the SD Controller Top to compose a 40 bit command messages to pass to the SD CMD
•Read response messages from the SD CMD Host and forward it to the user accessible register in the SD Controller Top module.
•Keep track of the status of the CMD Host module.

2.2.5.1 Port declaration

Direction	Width	name	Description
input	1	CLK_PAD_IO	CLK
input	1	RST_PAD_I	Reset
input	1	New_CMD	New command incoming
input	1	data_write	Data Write Command
input	1	data_read	Data Read Command
input	32	ARG_REG	Check Specification for details
input	16	CMD_SET_REG	Check Specification for details
input	16	TIMEOUT_REG	Check Specification for details
input	1	ERR_INT_RST	Error interrupt register reset
input	1	NORMAL_INT_RST1	Normal interrupt register reset
input	1	req_in	Request for service
input	1	ack_in	ACK on service completion
input	40	cmd_in	Command to be sent
input	16	serial_status	Status of the Serial_Host
output	16	STATUS_REG,	Check Specification for details
output	32	RESP_1_REG,	Check Specification for details
output	16	ERR_INT_REG,	Check Specification for details
output	16	NORMAL_INT_REG	Check Specification for details

output	16	settings	Settings to be used for the serial host
output	1	go_idle_o,	Put the serial host in idle state
output	40	cmd_out	CMD to be sent
output	1	req_out	Request for service
output	1	ack_out	ACK on a request service

Table 5: Sd_CMD_ master.v port declaration

2.2.5.2 Signal description

The usage of the input and output signals is as following.
The **New_CMD** is asserted when a new command is available in the **ARG_REG.** The both signals **data_write** and **data_read** is set if the incoming command is a block write or block read command. The module read **CMD_SET_REG** and sets up the **settings** signals together with **cmd_out**. Then to initiate a command transfer, the module assert **req_out,** until **ack_in** is received the **req_out, settings** and **cmd_out** is left unchanged during this time. The **req_in** signal is asserted when valid value on **serial_status** and/or **cmd_in** is present, this is acknowledged with **ack_out.** If a timeout occures the **go_idle_o** signal is asserted.

2.2.5.3 Module Operation

The module consist of a FSM with combinatorial, sequential and output logic, and two process for synchronize the req_in and ack_in signals.

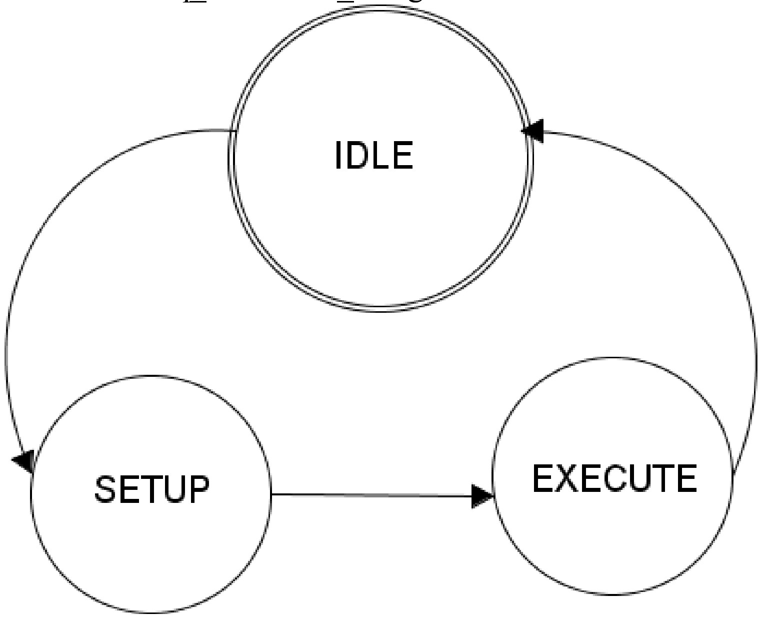

Fig 5: FSM Cmd Master

2.2.5.3.1 FSM State transitions

IDLE→ SETUP
New_CMD == 1

SETUP→ EXECUTE
ack_in_int== 1 (Acknowledged on req_out)

EXECUTE → IDLE
complete==1 (either watchdog timeout or status[6]==1)

2.2.2.3.2 FSM State operation

Idle:

Keep checking for status update in the cmd_host module and reseting the go_idle signal togheter with the req and ack out signals..

SETUP:

1. Compose cmd_out by combining command_setting_reg and arg_reg
2. Sets up the settings output vector
3. resets NORMAL_INT_REG, ERR_INT_REG and STATUS_REG

EXECUTE:

1. Increase watchdog counter and check if (Watchdog_Cnt>TIMEOUT_REG)
2. If timeout occur set EI bit and CTE bit to 1 and reset the CMD_Host module
3. Check if the ACK in line is high (CMD_Host is ready) if so request for service.
4. Check for new serial_status by checking the req_in signal
5. If new status, check it if status [6] is set, then a command cycle is completed
6. If set in command_setting check status for valid CRC and performe Index check
7. Store the data to the Resp_1 register, if response size is >0

2.2.6 Description Sd_data_master.v

Starts to check if there are any new BD thats need to be processed if so the module generate a command by setting up the command and argument register. It then set up the DMA in the FIFO filer to read/write to correct address. If the command line is free the module send the command and wait fore response. If response is valid the module starts the DMA if not valid the CMD is resent again.

During transmission the module keep track for FIFO buffet underflow or overflows, when the transmission is completed it check for valid CRC. If anything goes wrong during a transmission a stop command is sent and the module try to restart the transmission n times before giving up.

2.2.6.1 Port declaration

input	1	clk	Clock
input	1	rst	Reset

input	N:0	dat_in_tx,	Data in from Tx BD
input	M:0	free_tx_bd	NO Free Tx BD
input	1	ack_i_s_tx,	ACK in Read request
output	1	re_s_tx	Read Tx BD
output	1	a_cmp_tx	Free a Tx BD
input	N:0	dat_in_rx,	Data in from Rx BD
input	M:0	free_rx_bd	NO Free Rx BD
input	1	ack_i_s_rx,	ACK in Read request
output	1	re_s_rx	Read Rx BD
output	1	a_cmp_rx	Free a Rx BD
Input	1	cmd_busy	CMD Busy state
output	1	we_req,	Request access to CMD registers
input	1	we_ack	Access granted
output	1	d_write	Block write command
output	1	d_read	Block read command
output	32	cmd_arg	Cmd argugment out
output	16	cmd_set	Cmd setting out
input	1	cmd_tsf_err	Error status of sent cmd
input	5	card_status	Status of card after response
output	1	start_tx_fifo	Start the Tx Fifo Filler
output	1	start_rx_fifo	Start the Rx Fifo Filler
output	32	sys_adr	Memory address for DMA
input	1	tx_empt,	Tx Fifo empty flag
input	1	tx_full,	Tx Fifo full flag
input	1	rx_full	Rx Fifo full flag
input	1	busy_n	Data Busy
input	1	transm_complete	Transmission complete
input	1	crc_ok,	Crc status
output	1	ack_transfer	Ack o n transmission complete
output	8	Dat_Int_Status	Se Specification for register details
input	1	Dat_Int_Status_rst	Reset Dat_Int_Status
output	1	CIDAT	Data inhabit statusbit

Table 6: Sd_Data_ master.v port declaration

2.2.6.2 Signal description

The signal **free_xx_bd** is used to keep track on how many free buffer descriptors (BD) there are. To read a BD the signal **re_s_xx** is asserted when **ack_i_s_xx** is received the BD data is read from **dat_in_xx**. When a data block has been processed the BD is released with **a_cmp_xx** signal set to high.

To send a block write or block read command the first the internal **cmd_arg** and **cmd_set** register is set up. Then the **d_write** or **d_read** is set depending on type of data operation, block read or block write. Then **cmd_busy** is checked so the CMM line is free, if so the **we_req** signal is asserted, an acknowledgment **we_ack** is received when the command has been sent. When the response comes back the **cmd_tsf_err** and **card_status** is checked so the card is in correct state and no errors occurred during transfer.

The signals **start_xx_fifo** is used to start the fifo filler module, **tx_full**, **tx_empty** and **rx_full** is used to keep track on the status of the fifo. The system address the DMA uses is set by the **sys_adr** signal.

When a data block has been transmitted, the **transm_complete** signal asserted, the module resposne with setting **ack_transfer** to 1.

2.2.6.3 Module Operation

The module consist of a FSM with combinatorial, sequential and output logic, and one process for synchronize the transm_complete signal.

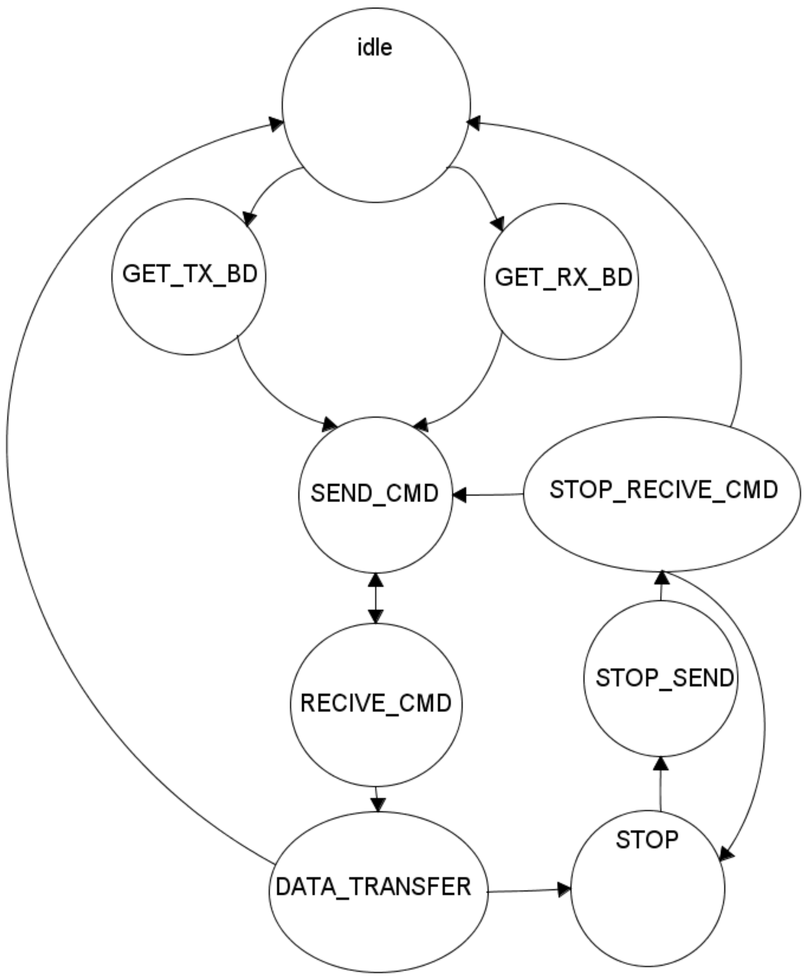

Fig 6: FSM Data Master

2.2.6.3.1 FSM State transitions

IDLE → GET_TX_BD
free_tx_bd !=`BD_EMPTY (BD Buffer not empty)
IDLE → GET_RX_BD
free_rx_bd !=`BD_EMPTY (BD Buffer not empty)

GET_TX_BD → SEND_CMD
bd_cnt > `READ_CYCLE-1 && tx_full==1 (Read two complete BD:s and the Tx FIFO is full)
GET_RX_BD → SEND_CMD
bd_cnt > `READ_CYCLE-1 (Read two complete BD:s)

SEND_CMD → RECIVE_CMD
send_done (we_ack == 1)

RECIVE_CMD → DATA_TRANSFER
rec_done (SD Card in correct state and no transfer error occurred)
RECIVE_CMD → SEND_CMD
rec_failed (SD Card in incorrect state or transfer error occurred)

DATA_TRANSFER → IDLE
trans_done (transm_comple == 1, crc_ok == 1 and busy_n == 1)
DATA_TRANSFER → STOP
trans_failed (CRC Error, FIFO overflowed/under run)

STOP→STOP_SEND
Always

STOP_SEND → STOP_RECIVE_CMD
send_done (we_ack == 1)

STOP_RECIVE_CMD→ RECIVE_CMD
rec_done (SD Card in correct state and no transfer error occurred)
STOP_RECIVE_CMD→IDLE
resend_try_cnt>=`RESEND_MAX_CNT (Still error after N tries)
STOP_RECIVE_CMD→STOP
rec_failed (SD Card in incorrect state or transfer error occurred)

2.2.6.3.2 FSM State operation

IDLE:
Reset all control signals

GET_TX_BD:
1. Enable re_s_Tx to read a BD,
2. When ack==1 save the data from the first BD to system addres
3. When ack==1 Save the data from the second BD to cmd_arg.
4. Set cmd_set to block_write
5. Set tx_cycle flag to 1
6. Start Tx_Ffo_Filer

GET_RX_BD:
1. Enable re_s_rx to read a BD
2. When ack==1 save the data from the first BD to system addres
3. When ack==1 Save the data from the second BD to cmd_arg.
4. Set cmd_set to block_read
5. Set rx_cycle flag to 1

SEND_CMD:
1. Check type of cycle rx or tx
2. set d_read or d_write acordely
3. Check if CMD is free, if so assert we_req
4. Wait for we_ack
5. Set send_done to 1 .

RECIVE_CMD:

1. Start Rx_Fifo_Filer if Rx cycle
2. Wait for command response (indicated by cmd_busy are low)

3. Check so cmd_tsf_err is not asserted, if asserted set command_failed.
4. Check the response data,
5. If card is ready for data card_status[0] == 1
6. If card is in transmission state for data card_status[4:1] == 4,5 or 6

DATA_TRANSFER:

1. If tx_cycle check so tx_fifo doenst get empty
2. If rx_cycle check so rx_fifo doenst get full
3. Wait for command complete
4. Check so crc_ok == 1
5. Free the BD

2.2.7 Description FIFO_RX_FILLER.v

This module works as the DMA it manager the rx_fifo for the data stream. It keeps track of the status of the FIFO:s if the fifo get full it signals this. The module contains the submodule sd_rx_fifo.v.

2.2.7.1 Port declaration

input	1	clk	Clock
input	1	rst	Reset
output	32	m_wb_adr_o	Wishbone master Address Out
output	1	m_wb_we_o	Wishbone master write enable
output	32	m_wb_dat_o	Wishbone master data out
output	1	m_wb_cyc_o	Wishbone master cycle out
output	1	m_wb_stb_o	Wishbone master strobe out
input	1	m_wb_ack_i	Wishbone master ack in
input	1	en	Enable module
input	32	adr	Memory read address
input	1	sd_clk	Write clk to FIFO
input	4	dat_i	Data in to FIFO
input	1	wr	Write enable to FIFO
output	1	full	FIFO Full Flag

Table 7: SD_FIFO_RX_FILLER.v port declaration

2.2.7.2 Signal description

The module contain the master wishbone interface signals. The signal En is used to start the operation of the module. The adr signal is where the module shall start read data from. The **sd_clk, dat_i** and **wr** are signals going to the write side of the sd_rx_fifo.

2.2.7.3 Operation

The module do nothing as long as the **en** signal is low. When the **en** signal is asserted the module do the following

If the **rx_fifo** is'nt empty and **wb_free** is not asserted (**wb_free** is a signal is set to 1 when free and 0 when wishbone transaction is in progres), the module start a wishbone write cycle, by reading 1 word from the fifo buffert and assigning to the **m_wb_dat_o.**

Then when **m_wb_ack_i** is arriving all wishbone signals is deasserted and the memory offset increase. The wb_free register is set to 1 again, signaling the wishbone bus is free again.
It perfromce those action until **en** is deasserted then all the internal register and the FIFO is reseted.

2.2.8 Description FIFO_TX_FILLER.v

This module works as the DMA it manager the tx_fifo for the data stream. It keeps track of the status of the FIFO:s if the fifo get empty it signals this. The module contains the submodule sd_tx_fifo.v.

2.2.8.1 Port declaration

input	1	clk	Clock
input	1	rst	Reset
output	32	m_wb_adr_o	Wishbone master Address Out
output	1	m_wb_we_o	Wishbone master write enable
output	32	m_wb_dat_o	Wishbone master data out
output	1	m_wb_cyc_o	Wishbone master cycle out
output	1	m_wb_stb_o	Wishbone master strobe out
input	1	m_wb_ack_i	Wishbone master ack in
input	1	en	Enable module
input	32	adr	Memory write address
input	1	sd_clk	Read clk to FIFO
input	32	dat_o	Data out frin FIFO
input	1	rd	Read enable to FIFO
output	1	empty	FIFO empty flag
output	1	fe	FIFO full flag

Table 8: SD_FIFO_RX_FILLER.v port declaration

2.2.8.2 Signal description

The module contain the master wishbone interface signals. The signal **En** is used to start the operation of the module. The adr signal is where the module shall start read data from. The **sd_clk, dat_o** and **rd** are signals going to the read side of the sd_tx_fifo.

2.2.8.3 Operation

The module do nothing as long as the **en** signal is low. When the **en** signal is asserted the module do the following

The module initiate a wishbone master read transaction when the **m_wb_ack_i** is not high and the fifo is not full and wishbone is free **(wb_free** asserted**) .** When **m_wb_ack_i** arrives all wishbone master signals is deserted, then the module set the FIFO write (**wr_tx**) signal and assign the **m_wb_dat_o** to the fifo **din.** A delay signal **delay** is then set to 1.

The delay is used to delay the offset increase and wb_free signaling with 1 cycle to prevent a new wishbone transaction to begin before fifo full signal is updated.

2.2.9 Description sd_rx_fifo.v / sd_tx_fifo.v

An ordinary FIFO with 1 read side and one write side, with logic to signaling full and empty state. Its designed to be implemented as register and not ram block so the size of the fifo should be kept low.

2.2.10 Description of sd_bd.v

The transmission and the reception processes are based on the descriptors. Two sequential wrings to this module is required to create one buffer descriptor. First the source address (Memory location) of the data is written then the card block address is written. Depending on the specified RAM width (16, 32 bits) 4 or 2 writings is required to forge a comlpete BD.

2.2.10.1 Port declaration

input	1	clk	Clock
input	1	rst	Reset
input	1	we_m	Write Enable Master side
input	1	re_m	Read Enable Master side
input	N:0	dat_in_m	Data in master side
output	N:0	dat_out_m	Data out master side
output	M:0	free_bd,	NO Free BD
input	1	re_s,	Read Enable slave side
output	1	ack_o_s	ACK slave
input	1	a_cmp	Free one BD
output	N:0	dat_out_s	Data out slave side

Table 9: sd_bd.v port declaration

2.2.10.2 Signals & operation

The signal **we_m** is asserted when the master side (writer) are writing the **dat_in_m** data to the BD ram. Master side can also read a BD by asserting **re_m**, data is then available on the **dat_out_m** port**.** When the reader side wants to read a BD **re_s** is asserted and data will be available on **dat_out_s** when **ack_o_s** is set.

2.2.11 Description of sd_controller_top.v

The host interface connects the IP Core to the rest of the system (RISC, memory) via the WISHBONE bus. The WISHBONE serves to access the configuration registers and the memory. Currently, only DMA transfers are supported for transferring the data from/to the memory

2.2.11.1 Port declaration

Direction	Width	Name	Description
Input	1	wb_clk_i	Slave WISHBONE Clock Input
Input	1	wb_rst_i	Slave WISHBONE Reset Input
Input	4	wb_sel_i	Slave WISHBONE Select Inputs
Input	32	wb_dat_i	Slave WISHBONE Data Inputs
Output	32	wb_dat_o	Slave WISHBONE Data Output
Input	8	wb_adr_i	Slave WISHBONE Address Input
Input	1	wb_we_i	Slave WISHBONE Write Enable
Input	1	wb_cyc_i	Slave WISHBONE Cycle
Input	1	wb_stb_i	Slave WISHBONE Strobe
Output	1	wb_ack_o	Slave WISHBONE Acknowledgment
Output	32	m_wb_adr_o	Master WISHBONE Address
Output	1	m_wb_sel_o	Master WISHBONE Select
Output	1	m_wb_we_o	Master WISHBONE Write Enable
Output	32	m_wb_dat_o	Master WISHBONE Data Output
Input	31	m_wb_dat_i	Master WISHBONE Data Input
Output	1	m_wb_cyc_o	Master WISHBONE Cycle
Input	1	m_wb_ack_i	Master WISHBONE Acknowledgment Input
Output	1	m_wb_cti_o	Master WISHBONE Cti
Output	1	m_wb_bte_o	Master WISHBONE Bte
Input	1	sd_cmd_dat_i	SDC/MMC CMD Input
Output	1	sd_cmd_out_o,	SDC/MMC CMD Output
Output	1	sd_cmd_oe_o	SDC/MMC CMD Output enable
Input	4	sd_dat_dat_i	SDC/MMC Data Input
Output	4	sd_dat_out_o	SDC/MMC Data Output
Output	1	sd_dat_oe_o	SDC/MMC Data Output enable

Direction	Width	Name	Description
Output	1	sd_clk_o_pad	SDC/MMC CLK Output
Input	1	sd_clk_i_pad	SDCLK input
Output	1	int_a,	Interrupt A Output
Output	1	int_b	Interrupt B Output
Output	1	int_c	Interrupt C Output

Table 10: sd_controller_top.v port declaration

2.2.11.2 Signal & Operation

As this is the top module for the the design it contains all the Wishbone signals, SD-Card interface and the interrupt signals.

The module instantiate all the major submodules, the connection between the different modules can be seen in figure 2.

The module also contain some muxes where more then one module want to acces the same recourse. This is the case whith the tx and rx fifo filler where both need to drive the wishbone master signals _**cyc**, _**stb**, _**we** and _**adr_o**. The mux uses the **start_rx_fifo** and **start_tx_fifo** from the data_master module to descried witch module who needs access to the bus, if neither do its set to 0.

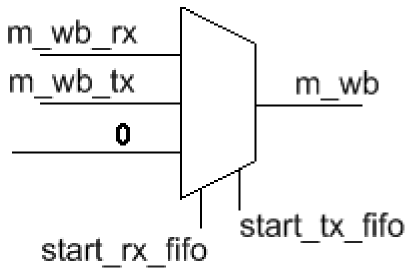

Fig 7: Master Wishbone mux

The module also clock in some wireconnection to registers

2.2.12 Description of sd_controller_wb.v