SD Memory Slave Controller Integration Manual

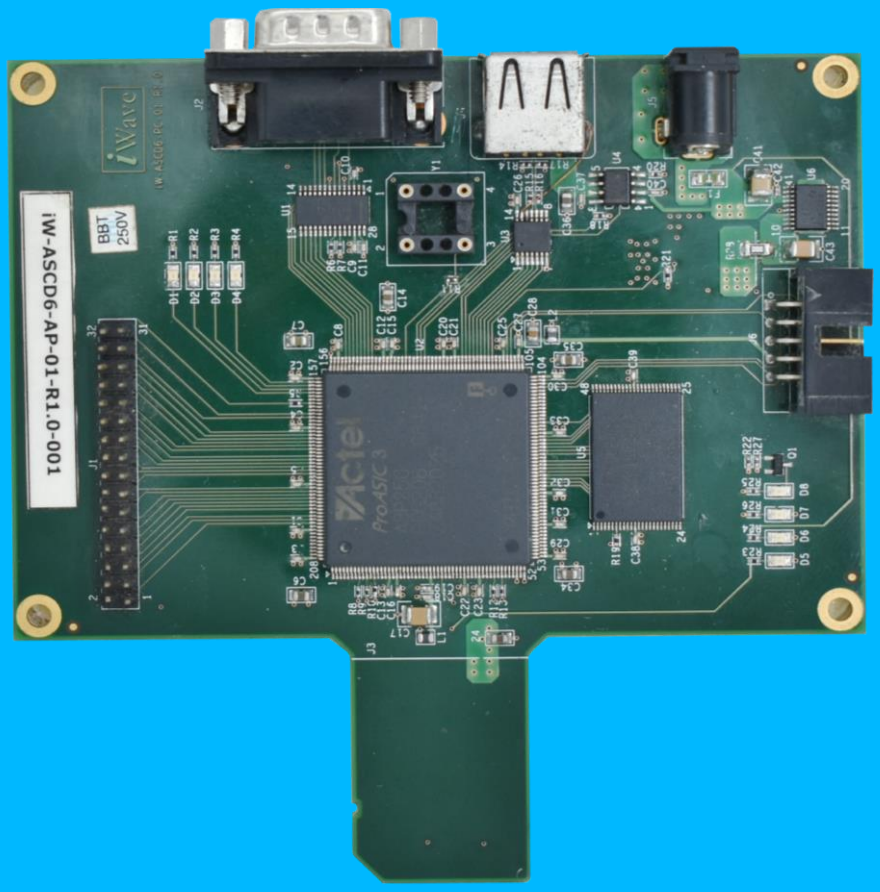

Table of Contents

List Of Figures

List Of Tables

1 Introduction

1.1 Purpose

The purpose of this document is to describe SD Memory Slave Controller Integration manual.

1.2 Scope

This document describes the how to integrate the SD Memory Slave Controller. This document contains information about IP interface ,synthesis and verification details.

1.3 Reference Document

- SD Physical Specification Version 2.00
- SD Memory Card Test specification version 2.0.
- SD Memory Slave Controller design document (iW-ASCDS-DD-01-R1.0.doc).

1.4 Overview

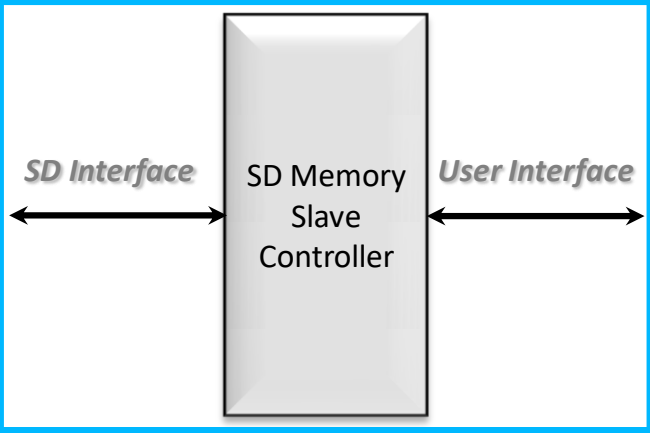

Figure 1: SD Memory Slave Controller Block Diagram

SD Memory Slave Controller can be used evaluate SD to any custom logic/interface. The SD memory interface supported by the bridging applications enables a low-cost and small size implementation. A typical application includes a communication link between SD memory host interface and memory devices.

User interface can be used to access the SD memory application interface.

SD Memory Slave Controller is a rigorously verified ,reliable and easy-to-integrate core. Ease of integration is served by a complete verification environment.

The deliverables for the SD Memory Slave Controller include the synthesizable RTL code (Verilog), the standalone testbench, synthesis and simulation as well as full documentation.

The top level component supplied is the sd_mem_slave component. The sd_mem_slave component includes the instantiation of RAM modules, and it is synthesizable only for FPGA technologies. ASIC users should use the appropriate RAM modules.

1.5 Acronyms and Abbreviations

Table 1: Acronyms & Abbreviations

Term	Meaning
FPGA	Field Programmable Gate Array
MMC	Multi-Media Card
OS	Operating System
PC	Personal Computer
SD	Secure Digital
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

2 IP Configuration and Instantiation

2.1 IP Configuration

The Verilog source version of the sd_mem_slave is configured before synthesis through the package header file sd_mem_define.v.

2.1.1 sd_mem_define.v

```
// Uncomment to configure the SD memory Slave controller is capable of high
 // Capacity card
//~ `define  HIGH_CAP_CARD
 `define VER2_0

 // CSR Register Configuration
`ifdef HIGH_CAP_CARD // High Capacity Card (4GB Card)
   `define CCS                1'b1               // Card Capacity Status
   `define CSD_STRUCTURE      2'b01              // High Capacity (Version 2.0)
   `define TAAC               8'b0_0001_110      // Read Data Access Time 1ms
   `define NSAC               8'b0000_0000       // Read Data Access time in CLK cycles
   `define TRAN_SPEED_DEF     8'h32              // 12.5 MB/s
   `define TRAN_SPEED_HIGH    8'h5A              // 25 MB/s
   `define CCC                12'b0101_0011_0101 // Speed Class 0,2,4,5,8,10
   `define READ_BL_LEN        4'h9               // 512 Bytes
   `define READ_BL_PARTIAL    1'b0               // Partial Read Block is not allowed
   `define WRITE_BLK_MISALIGN 1'b0               // Crossing physical block boundaries is not valid
   `define READ_BLK_MISALIGN  1'b0               // Crossing physical block boundaries is not valid
   `define DSR_IMP            1'b0               // DSR is not implemented
   `define C_SIZE_HIGH        22'h1FFF           // 8192 * 512KB = 4GB
   `define ERASE_BLK_LEN      1'b1               //
   `define SECTOR_SIZE        7'h7F              // 128
   `define WP_GRP_SIZE        7'h0
   `define WP_GRP_ENABLE      1'b0               // No Write Protect Group
   `define R2W_FACTOR         3'h2               // program time is 4 times slower than read
   `define WRITE_BL_LEN       4'h9               // 512 Bytes
   `define WRITE_BL_PARTIAL   1'b0               // Partial Write Block is not allowed
   `define FILE_FORMAT_GRP    1'b0
   `define COPY               1'b1               // MTP device sold to customer
   `define PERM_WRITE_PROTECT 1'b0               // Permanently not write protected
   `define TMP_WRITE_PROTECT  1'b0               // Temperarily not write protected
   `define FILE_FORMAT        2'h0
   `define MAX_ADDR           32'h800000         // Max Cap = (C_SIZE_HIGH+1) * 512KB = (8191+1) * 512KB = 4GB
                                                 // Max Address = 4GB/512 = 8388608
`else                                            // Standard Capacity Card (32MB Card)
```

```
`define CCS                1'b0              // Card Capacity Status
`define CSD_STRUCTURE      2'b00             // Standard Capacity (Version 2.0)
`define TAAC               8'b0_0001_110     // Read Data Access Time 1ms
`define NSAC               8'b0000_0000      // Read Data Access time in CLK cycles
`define TRAN_SPEED_DEF     8'h32             // 12.5 MB/s
`define TRAN_SPEED_HIGH    8'h5A             // 25 MB/s
`define CCC                12'b0101_0011_0101 // Speed Class 0,2,4,5,8,10
`define READ_BL_LEN        4'h9              // 512 Bytes
`define READ_BL_PARTIAL    1'b1              // Partial Read Block is allowed
`define WRITE_BLK_MISALIGN 1'b1              // Crossing physical block boundaries is valid
`define READ_BLK_MISALIGN  1'b1              // Crossing physical block boundaries is valid
`define DSR_IMP            1'b0              // DSR is not implemented
`define C_SIZE             12'h0             // 2000 original 12'h7CF, changed to 12'h0 to reduce size to 2k
`define VDD_R_CURR_MIN     3'h1              // 100mA original 3'h7, changed to 3'h1 (1 mA) to test
`define VDD_R_CURR_MAX     3'h2              // 200mA original 3'h7, changed to 3'h2 (10 mA) to test
`define VDD_W_CURR_MIN     3'h1              // 100mA original 3'h7, changed to 3'h1 (1 mA) to test
`define VDD_W_CURR_MAX     3'h2              // 200mA original 3'h7, changed to 3'h2 (10 mA) to test
`define C_SIZE_MULT        3'h0              // Multiplication factor is 3 orig 3'h3, changed to 3'h0 to declare 2k mem size
`define ERASE_BLK_LEN      1'b1              //
`define SECTOR_SIZE        7'h7F             // 128
`define WP_GRP_SIZE        7'h0
`define WP_GRP_ENABLE      1'b0              // No Write Protect Group
`define R2W_FACTOR         3'h2              // program time is 4 times slower than read
`define WRITE_BL_LEN       4'h9              // 512 Bytes
`define WRITE_BL_PARTIAL   1'b1              // Partial Write Block is allowed
`define FILE_FORMAT_GRP    1'b0
`define COPY               1'b1              // MTP device sold to customer
`define PERM_WRITE_PROTECT 1'b0              // Permanently not write protected
`define TMP_WRITE_PROTECT  1'b0              // Temperarily not write protected
`define FILE_FORMAT        2'h0
`define MAX_ADDR           32'h800           // Max Cap = Max Address = 800h for 2K memory
```

```
// Card Identification Register Configuration
`define MID    8'h0               // Mabufacturer ID
`define OID    16'h69_57          // ASCII Value of iW
`define PNM    40'h53_44_4D_45_4D // ASCII Value of SDMEM
`define PRV    8'b0010_0000       // Product Revision 2.0
`define PSN    32'h0              // Product Serial Number
`define MDT    12'b0000_1011_0101 // Manufacturing Date (May, 2011)

// SCR Register Configuration
`define DATA_STAT_AFTER_ERASE 1'b1    // Data After Erase will be 1
`define SD_SECURITY           3'h0    // No Security
`define SD_BUS_WIDTHS         4'b0101 // 1bit as well as 4-bit support

// OCR Register
`define OCR    24'hFF8000 // Voltage Range 2.7 to 3.6

// SD Status register
`define SPEED_CLASS      8'hAB // Class 0 (Performance not specified)
`define PERFORMANCE_MOVE 8'h98 // Not defined
`define AU_SIZE          4'h7 // Not Defined
`define ERASE_SIZE       16'h6543 // Erase time-out calculation is not supported
`define ERASE_TIMEOUT    6'h2 // Erase time-out calculation is not supported
`define ERASE_OFFSET     2'h1 // Meaningless when ERASE_TIMEOUT and ERASE_SIZE is 0

`define MAX_CURR_CONS    16'h6 // 64mA (2^6)
```

Figure 2: IP Configuration File

Single header is provided to configure the SD memory Slave controller as either high Capacity card or standard capacity card. Uncomment the `define HIGH_CAP_CARD to configure the SD memory Slave controller as high Capacity card. The card status register CSR, card identification

register CID, SD configuration register SCR, operational condition register OCR and SD status register SSR are defined in the sd_mem_defines.v file.

The card configuration is done through sd_mem_defines.v by changing the attributes. Below example explains few configurations,

1. Configuring sd_mem_slave as high capacity card of size 8GB
   - Uncomment the `define HIGH_CAP_CARD
   - `define C_SIZE_HIGH 22'h3FFF
   - `define MAX_ADDR 32'h1000000
2. Configuring sd_mem_slave as standard capacity card of size 32MB for block length of 512bytes
   - Comment the `define HIGH_CAP_CARD
   - `define C_SIZE 12'h7CF
   - `define MAX_ADDR 32'h2000000
   - `define C_SIZE_MULT 3'h3
3. Configuring sd_mem_slave as standard capacity card of size 2KB for block length of 512bytes
   - Comment the `define HIGH_CAP_CARD
   - `define C_SIZE 12'h0
   - `define MAX_ADDR 32'h800
   - `define C_SIZE_MULT 3'h0
4. Configuring the Manufacturing ID of CID register as x98
   - `define MID 8'h98
5. Configuring OCR register for 2.7V to 3.3V
   - `define OCR 24'h1F8000

2.1.2 Component Instantiation

sd_mem_slave component can be instantiated as follows:

```
sd_mem_top sd_mem_top
(
  // Clock & Reset
  .sys_clk_i            ( sys_clk_i         ),
  .sys_rst_n_i          ( sys_rst_n_i       ),
  .sd_rst_n_i           ( sys_rst_n_i       ),

  // SD Memory Host Interface
  .sd_clk_i             ( sd_clk_i          ),
  .sd_cmd_in_i          ( sd_cmd_in         ),
  .sd_cmd_out_o         ( sd_cmd_out        ),
  .sd_cmd_oen_o         ( sd_cmd_oen        ),
  .sd_data_in_i         ( sd_data_in        ),
  .sd_data_out_o        ( sd_data_out       ),
  .sd_data_oen_o        ( sd_data_oen       ),
  .cd_disable_o         ( cd_disable_o      ),

  // User Interface
  .user_xfer_req_o      ( user_xfer_req     ),
  .user_xfer_addr_o     ( user_xfer_addr    ),
  .user_xfer_cmd_o      ( user_xfer_cmd     ),
  .erase_no_of_blk_o    ( erase_no_of_blk   ),
  .user_rd_ready_o      ( user_rd_ready     ),
  .user_rd_dvalid_i     ( user_rd_dvalid    ),
  .user_rd_data_i       ( user_rd_data      ),
  .user_wr_ready_i      ( user_wr_ready     ),
  .user_wr_dvalid_o     ( user_wr_dvalid    ),
  .user_wr_data_o       ( user_wr_data      ),
  .user_txfer_status_i  ( user_txfer_status ),
  .user_xfer_abort_o    ( user_xfer_abort   ),
  .card_reset_o         ( card_reset        ),
  .command_index_o      ( command_index     ),
  .cmd_arg_o            ( cmd_arg           ),
  .cmd_valid_o          ( cmd_valid         ),
  .block_len_o          ( block_len         ),
  .card_status_o        ( card_status       ),
  .card_hcs_status_o    ( card_hcs_status   ),
  .card_speed_type_o    ( card_speed_type   )
);
```

Figure 3: Component Instantiation

3 Deliverables

3.1 Introduction

The SD Memory Slave Controller Core is a reliable and easy-to-integrate. Ease of integration is served by a complete verification environment.

The deliverables for the SD Memory Slave Controller core include the synthesizable RTL code (Verilog), a standalone test bench, example design for SD memory slave controller synthesis and simulation environments as well as full documentation. A typical release tree is as shown in the figure below.

3.2 Directory Structure

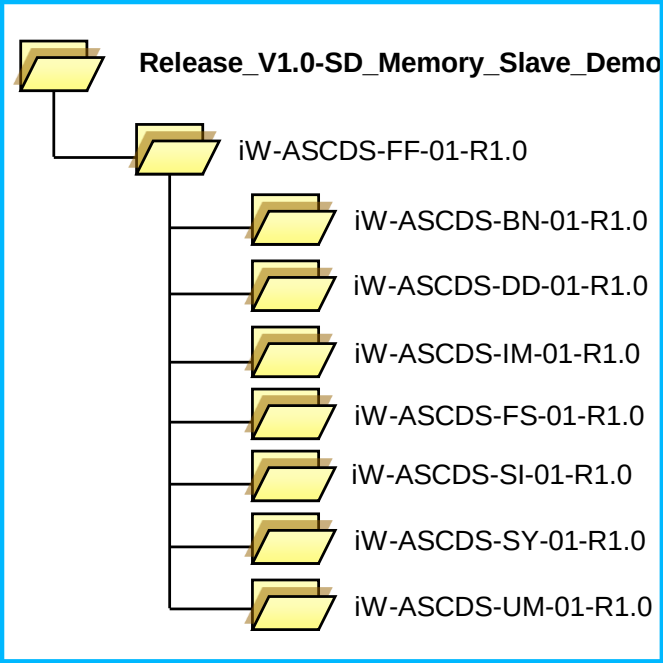

Figure 4: Directory Structure

4 Implementation Details

4.1 Example design for SD Memory Slave Controller

This design illustrates that, how the sd_mem_slave component can be interfaced at the user interface side. example_design is the top level block which integrates SD memory slave controller and example design.

Example design module interfaces to user interface side and generates the controls to access the attached 2KB RAM memory.

4.1.1 sim_wait.v

This header file used only for simulation purpose, so for FPGA implementation using user application example design `define SIM_WAIT should always be commented.

4.2 Clock Domain

sd_mem_slave uses two clock domains sd_clk_i and sys_clk_i.

4.3 Signals Crossing Clock Domains

Table 2: Signal list of crossing clock domains

Path	Source clock	Destination clock
sd_mem_top/sd_cmd_path/sd_bus_fsm/xfer_req_o to sd_mem_top/user_if/xfer_req_d1	sd_clk_i	sys_clk_i
sd_mem_top/sd_cmd_path/sd_bus_fsm/xfer_cmd_o[1:0] to sd_mem_top/user_if/xfer_cmd_d[1:0]	sd_clk_i	sys_clk_i
sd_mem_top/sd_cmd_path/sd_bus_fsm/xfer_block_len_o[9:0] to sd_mem_top/user_if/xfer_block_len_d[9:0]	sd_clk_i	sys_clk_i
sd_mem_top/sd_cmd_path/sd_cmd_fsm/block_mode_o to sd_mem_top/user_if/block_mode_d	sd_clk_i	sys_clk_i
sd_mem_top/sd_cmd_path/sd_cmd_fsm/pre_erase_o to sd_mem_top/user_if/pre_erase_d	sd_clk_i	sys_clk_i
sd_mem_top/sd_cmd_path/sd_bus_fsm/data_read_o to sd_mem_top/user_if/data_read_d	sd_clk_i	sys_clk_i
sd_mem_top/sd_cmd_path/sd_cmd_fsm/data_addr_o[31:0] to sd_mem_top/user_if/data_addr_d[31:0]	sd_clk_i	sys_clk_i
sd_mem_top/sd_cmd_path/sd_cmd_fsm/erase_saddr_o[31:0] to sd_mem_top/user_if/erase_saddr_d[31:0]	sd_clk_i	sys_clk_i
sd_mem_top/sd_cmd_path/sd_cmd_fsm/erase_eaddr_o[31:0] to sd_mem_top/user_if/erase_eaddr_d[31:0]	sd_clk_i	sys_clk_i

sd_mem_top/user_if/erase_no_of_blk_i[22:0] to sd_mem_top/user_if/erase_no_of_blk_d[22:0]	sd_clk_i	sys_clk_i
sd_mem_top/user_if/stop_cmd_rcvd to sd_mem_top/user_if/stop_cmd_rcvd_d1	sd_clk_i	sys_clk_i
sd_mem_top/sd_data_path/sd_data_fsm/wr_blk_done_o to sd_mem_top/user_if/wr_blk_done_d1	sd_clk_i	sys_clk_i
sd_mem_top/sd_data_path/sd_data_fsm/wr_blk_crc_err_o to sd_mem_top/user_if/wr_blk_crc_err_d1	sd_clk_i	sys_clk_i
sd_mem_top/user_if/write_completed_sd to sd_mem_top/user_if/write_completed_sd_d1	sd_clk_i	sys_clk_i
sd_mem_top/user_if/write_buffer/rempty to sd_mem_top/user_if/wrbuf_empty_d1	sys_clk_i	sd_clk_i
sd_mem_top/user_if/write_complete_rcvd to sd_mem_top/user_if/write_complete_rcvd_d1	sys_clk_i	sd_clk_i
sd_mem_top/user_if/usr_if_error_s to sd_mem_top/user_if/usr_if_error_d1	sys_clk_i	sd_clk_i
sd_mem_top/user_if/stop_cmd_rcvd_sys to sd_mem_top/user_if/stop_cmd_rcvd_sys_d1	sys_clk_i	sd_clk_i
sd_mem_top/user_if/read_buffer/wptr[9:0] to sd_mem_top/user_if/read_buffer/rwptr1[9:0]	sys_clk_i	sd_clk_i
sd_mem_top/user_if/write_buffer/wptr[9:0] to sd_mem_top/user_if/write_buffer/rwptr[9:0]	sd_clk_i	sys_clk_i
sd_mem_top/user_if/read_buffer/rptr[9:0] to sd_mem_top/user_if/read_buffer/wrptr1[9:0]	sd_clk_i	sys_clk_i
sd_mem_top/user_if/write_buffer/rptr[9:0] to sd_mem_top/user_if/write_buffer/wrptr1[9:0]	sys_clk_i	sd_clk_i

4.4 Memory Requirements

Table 3: Memory Requirements

Kind of memory	Name	Number	PortA		PortB		Address width (bits)	Number of word	Data width (bits)
			W/R	Clock	W/R	Clock			
Async 1KB FIFO	write_buffer	1	W	sd_clk_i	R	sys_clk_i	NA	1024	8
	read_buffer	1	W	sys_clk_i	R	sd_clk_i	NA	1024	8

4.5 Example Design for sd_mem_slave constraints

4.5.1 example_top.sdc

```
########   False Path Constraints  ########

set_false_path   from [get_cells { sd_mem_top/user_if/xfer_req_d1 }]  -to [get_cells { \
sd_mem_top/user_if/xfer_req_d2 }]

set_false_path  -from [get_cells { sd_mem_top/user_if/stop_cmd_rcvd_d1 }]  -to [get_cells { \
sd_mem_top/user_if/stop_cmd_rcvd_sys }]

set_false_path  -from [get_cells { sd_mem_top/user_if/stop_cmd_rcvd_sys_d1 }]  -to \
[get_cells { sd_mem_top/user_if/stop_cmd_rcvd_sd }]

set_false_path  -from [get_cells { sd_mem_top/user_if/usr_if_error_d1 }]  -to [get_cells { \
sd_mem_top/user_if/usr_if_error_o }]

set_false_path  -from [get_cells { sd_mem_top/user_if/wrbuf_empty_d1 }]  -to [get_cells { \
sd_mem_top/user_if/wrbuf_empty_sd }]

set_false_path  -from [get_cells { *read_buffer/wptr[* }]  -to [get_cells { \
*read_buffer/rwptr1[* }]

set_false_path  -from [get_cells { *write_buffer/wptr[* }]  -to [get_cells { \
*write_buffer/rwptr1* }]

set_false_path  -from [get_cells { *read_buffer/rptr[* }]  -to [get_cells { \
*read_buffer/wrptr1[* }]

set_false_path  -from [get_cells { *write_buffer/rptr[* }]  -to [get_cells { \
*write_buffer/wrptr1[* }]

set_false_path  -from [get_cells { *write_complete_rcvd_d1* }]  -to [get_pins { \
sd_mem_top/user_if/write_completed_sd:D }]

set_false_path  -from [get_cells { sd_mem_top/user_if/write_completed_sd_d1 }]  -to \
[get_cells { sd_mem_top/user_if/write_completed_sys }]

set_false_path  -from [get_cells { sd_mem_top/user_if/read_buffer/wptr[2] \
sd_mem_top/user_if/read_buffer/wptr[3] sd_mem_top/user_if/read_buffer/wptr[0] \
sd_mem_top/user_if/read_buffer/wptr[1] sd_mem_top/user_if/read_buffer/wptr[6] \
sd_mem_top/user_if/read_buffer/wptr[7] sd_mem_top/user_if/read_buffer/wptr[4] \
sd_mem_top/user_if/read_buffer/wptr[5] sd_mem_top/user_if/read_buffer/wptr[8] \
sd_mem_top/user_if/read_buffer/wptr[9] }]  -to [get_cells { \
sd_mem_top/user_if/read_buffer/rwptr1[10] \
sd_mem_top/user_if/read_buffer/rwptr1[7] sd_mem_top/user_if/read_buffer/rwptr1[9] \
sd_mem_top/user_if/read_buffer/rwptr1[6] sd_mem_top/user_if/read_buffer/rwptr1[8] \
sd_mem_top/user_if/read_buffer/rwptr1[1] sd_mem_top/user_if/read_buffer/rwptr1[0] \
sd_mem_top/user_if/read_buffer/rwptr1[3] sd_mem_top/user_if/read_buffer/rwptr1[2] \
sd_mem_top/user_if/read_buffer/rwptr1[5] sd_mem_top/user_if/read_buffer/rwptr1[4] \
}]
```

```
set_false_path  -from [get_cells { sd_mem_top/user_if/write_buffer/wptr[6] \
sd_mem_top/user_if/write_buffer/wptr[5] sd_mem_top/user_if/write_buffer/wptr[8] \
sd_mem_top/user_if/write_buffer/wptr[7] sd_mem_top/user_if/write_buffer/wptr[9] \
sd_mem_top/user_if/write_buffer/wptr[1] sd_mem_top/user_if/write_buffer/wptr[3] \
sd_mem_top/user_if/write_buffer/wptr[2] sd_mem_top/user_if/write_buffer/wptr[4] \
sd_mem_top/user_if/write_buffer/wptr[0] }]  -to [get_cells { \
sd_mem_top/user_if/write_buffer/rwptr1[10] \
sd_mem_top/user_if/write_buffer/rwptr1[1] \
sd_mem_top/user_if/write_buffer/rwptr1[0] \
sd_mem_top/user_if/write_buffer/rwptr1[4] \
sd_mem_top/user_if/write_buffer/rwptr1[3] \
sd_mem_top/user_if/write_buffer/rwptr1[5] \
sd_mem_top/user_if/write_buffer/rwptr1[8] \
sd_mem_top/user_if/write_buffer/rwptr1[7] \
sd_mem_top/user_if/write_buffer/rwptr1[2] \
sd_mem_top/user_if/write_buffer/rwptr1[9] \
sd_mem_top/user_if/write_buffer/rwptr1[6] }]
set_false_path  -from [get_cells { sd_mem_top/user_if/read_buffer/rptr[0] \
sd_mem_top/user_if/read_buffer/rptr[1] sd_mem_top/user_if/read_buffer/rptr[8] \
sd_mem_top/user_if/read_buffer/rptr[9] sd_mem_top/user_if/read_buffer/rptr[6] \
sd_mem_top/user_if/read_buffer/rptr[3] sd_mem_top/user_if/read_buffer/rptr[7] \
sd_mem_top/user_if/read_buffer/rptr[4] sd_mem_top/user_if/read_buffer/rptr[5] \
sd_mem_top/user_if/read_buffer/rptr[2] }]  -to [get_cells { \
sd_mem_top/user_if/read_buffer/wrptr1[2] sd_mem_top/user_if/read_buffer/wrptr1[1] \
sd_mem_top/user_if/read_buffer/wrptr1[3] sd_mem_top/user_if/read_buffer/wrptr1[6] \
sd_mem_top/user_if/read_buffer/wrptr1[5] sd_mem_top/user_if/read_buffer/wrptr1[0] \
sd_mem_top/user_if/read_buffer/wrptr1[7] sd_mem_top/user_if/read_buffer/wrptr1[9] \
sd_mem_top/user_if/read_buffer/wrptr1[4] sd_mem_top/user_if/read_buffer/wrptr1[8] \
sd_mem_top/user_if/read_buffer/wrptr1[10] }]
set_false_path  -from [get_cells { sd_mem_top/user_if/write_buffer/rptr[0] \
sd_mem_top/user_if/write_buffer/rptr[1] sd_mem_top/user_if/write_buffer/rptr[2] \
sd_mem_top/user_if/write_buffer/rptr[7] sd_mem_top/user_if/write_buffer/rptr[8] \
sd_mem_top/user_if/write_buffer/rptr[9] sd_mem_top/user_if/write_buffer/rptr[3] \
sd_mem_top/user_if/write_buffer/rptr[4] sd_mem_top/user_if/write_buffer/rptr[5] \
sd_mem_top/user_if/write_buffer/rptr[6] }]  -to [get_cells { \
sd_mem_top/user_if/write_buffer/wrptr1[10] \
sd_mem_top/user_if/write_buffer/wrptr1[0] \
sd_mem_top/user_if/write_buffer/wrptr1[1] \
sd_mem_top/user_if/write_buffer/wrptr1[4] \
sd_mem_top/user_if/write_buffer/wrptr1[5] \
sd_mem_top/user_if/write_buffer/wrptr1[6] \
sd_mem_top/user_if/write_buffer/wrptr1[7] \
sd_mem_top/user_if/write_buffer/wrptr1[8] \
sd_mem_top/user_if/write_buffer/wrptr1[9] \
sd_mem_top/user_if/write_buffer/wrptr1[2] \
sd_mem_top/user_if/write_buffer/wrptr1[3] }]
set_false_path  -from [get_cells { sd_mem_top/user_if/write_complete_rcvd_d1 }]  -to \
[get_pins { sd_mem_top/user_if/write_completed_sd:D }]
```

Figure 5: Example design for sd_mem_slave IP constraint File

5 FPGA Implementation

5.1 Resource Utilization

The table below shows the resource utilization summary for Actel ProAsic (A3P250–PQG 208) FPGA device for Example design for sd_mem_slave.

Table 4: FPGA Resource Utilization

No. of Core Tiles	5668
No. of RAM/FIFO Block	8
IO	9
Global	6

6 Simulation Environment

6.1 Block Diagram

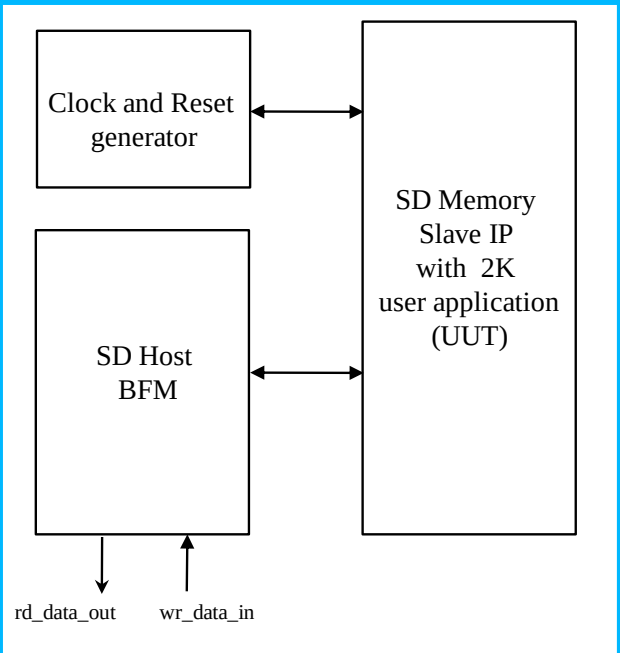

Figure 6 : Test Environnent

6.2 Description of Test Environment

The test environment consists of mainly below units:

- Clock and Reset generator
- SD Host BFM
- Text files

6.2.1 SD memory Slave IP (UUT):

SD memory slave IP acts as the SD memory device which is the unit under test.

6.2.2 Clock and Reset generator:

This module will generate the system clock and system reset signal required for the SD memory IP and EBI interface.

6.2.3 SD Host BFM:

SD host BFM has the different tasks which will generate the different commands to test the IP. This module also receives the response from the IP and checks whether the response is received properly or not. During the write operation this BFM reads the data from the wr_data_in.txt and

passes the data in the data line as the write data to the IP. During read it receives the data from IP and compares with the input data and displays test result depending on the result of comparison.

6.2.4 Test Description:

The defined test case will be used to test the single and multi-block write and read. The test case will first test the single block operation by writing the 512 bytes of the data followed by the read of same. After the successful write and read the host BFM will compare the read data with written data. If there is no data mismatch which means the test is successful, the BFM will initiate the multi-block write with 4 blocks of data followed by the multi-block read. After completion of all block write and read, the BFM compares the read data with written data and depending on the comparison, the test result will be displayed. User can check the waveform and confirm that the test results.

7 Simulation Procedure

The following steps are followed to simulate the design in Modelsim DE 10.1.

1. Open Modelsim DE 10.1 and change directory to the test folder with the following command in the Modelsim prompt.

   ModelSim> cd {path of simulation environment} to change the simulation directory

2. Simulation can be carried out with below mentioned command

   Modelsim> do run.do

3. Verify the design by observing the Modelsim console. The console will display different debugging message

4. The verification can also be done by observing the waveform on the waveform window.