Banerjee and Mukhopadhyay *EURASIP Journal on Embedded Systems* (2016) 2016:24
DOI 10.1186/s13639-016-0060-8

EURASIP Journal on Embedded Systems

RESEARCH Open Access

# A low cost and fast controller architecture for multimedia data storage and retrieval to flash-based storage device

Samiran Banerjee* and Sumitra Mukhopadhyay

**Abstract**

Real-time multimedia data access plays an important role in electronic systems; as time goes by, with decrease in data processing speed and increase in communication time, storage time, and retrieval time, the overall response time increases for real-time applications. Therefore, in this paper, a novel real-time, fast, low-cost, system-on-chip (SoC) controller has been proposed and implemented where large volume of data can be efficiently stored and retrieved from flash memory cards. It is being implemented only using hardware description language (HDL) on a field programmable gate array (FPGA) chip without using any other on-board or external hardware resources or high-level languages. The entire controller architecture, in a single chip, contains five different modules and is designed using finite state machine (FSM)-based approach. The modules are card initialization module (CINM), idle module (IM), card read module (CRM), card write module (CWM), and decision module (DM). The architecture is completely synthesized for Spartan 3E xc3s500e-4-fg320 FPGA with only 5% of the total logic utilization. The experimental results tested for microSD, SD, and SDHC cards of different size, and these show that the architecture uses less hardware and clock cycles for card initialization and single/multiblock read/write procedure.

**Keywords:** Flash memory read/write, Secure Digital High Capacity (SDHC) card, MicroSD card, Serial peripheral interface (SPI), Finite state machine (FSM), Very high speed integrated circuit hardware description language (VHDL), Field programmable gate array (FPGA)

## 1 Introduction

The flash-based memory storage device, introduced by Toshiba in 1984, is basically a non-volatile electronic memory and used whenever a shock resistance is the key requirement of any application [1]. The Secured Digital High Capacity (SDHC) card, for example, is a flash-based memory storage device and is mainly designed to meet certain requirements such as security, capacity, performance, and environmental issues inherent in newly emerging audio and video consumer electronic devices. The Secured Digital (SD) card standard is designed and licensed by SD Card Association [2] and is a collaborative effort of the three manufacturers, namely Toshiba, SanDisk, and MEI [3].

*Correspondence: samiranbanerjee1991@gmail.com
Institute of Radio Physics and Electronics, University of Calcutta, 92, APC Road, Kolkata, India

The SD card includes an on-card intelligent controller to manage the interface protocol, security algorithms, data storage and retrieval, error handling and corresponding error correction code (ECC) algorithms, defect handling and diagnostics, power management, and clock control [1]. However, to interface the SDHC card (slave unit) with master unit (e.g., computer, host, or any application-specific device), we need a system which can talk with the on-card controller of the SDHC card for smooth execution of single/multiblock data read/write.

Figure 1 represents a generic model of a data archival system and it shows how the flash-based cards like microSD, SD, or SDHC cards can be used as the plug and play memory module for real-time application. Generally, the signal is received from the external world in a buffer, converted into a serial bit stream and subsequently stored into the memory card. The stored data at later stage may be transmitted for further processing. Also, the flash memory acts as a portable unit and it can be

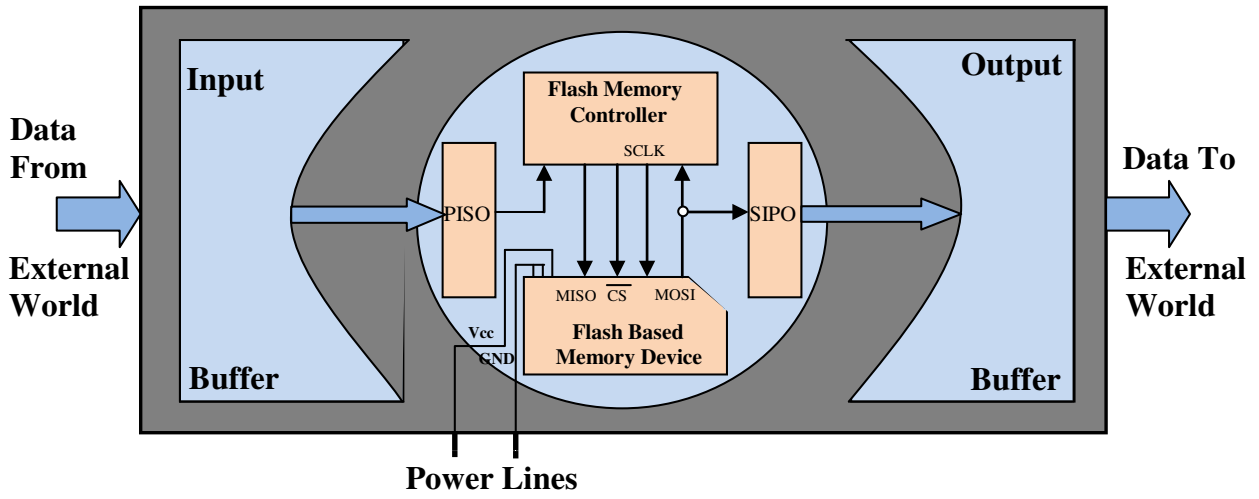


**Fig. 1** Thematic model of a data archival system

removed from the system where it is presently housed in and accessed by some other system to retrieve the desired signal.

The flash memory is very much useful in fields where data transportation and archival is a key requirement. The memory can be used as a data concentrator (Fig. 2) where the proposed controller architecture along with the flash memory can be used as the removable memory of any data concentrator network device. The flash memory has extensive application in database [4], networking [5], biomedical application [6, 7], virtualized storage system [8], cloud computing [9], geographic remote sensing [10], mobile devices [11–13], etc. It can be used in a router to store the routing table for further access. NAND-based flash system is also widely employed as cache in virtualized system [8]. In such application scenarios, the efficiency of single and multiblock data transfer is very important which consequently affects the input-output operations per seconds (IOPS) measure of storage system. Generally, we wish to maximize this metric with respect to different types of flash as this indicates the measure of flash utilization. As stated, flash-based system can be used as a cache or a data concentrator or to cater any such storage requirements. However, here in the paper, we do not analyze the pros and cons of such utilization of flash in detail as the work mainly concentrates on efficient implementation of a controller for single or multiblock data transfer with respect to flash memory. The implementation may be exploited in any kind of flash resource utilization and will ultimately contribute in the calculation of the metric of memory resource utilization. Therefore, it is observed that the implementation of an efficient flash-based data transfer is a fundamental driving factor in the improvement of flash resource utilization and the paper is focusing on that rudimentary aspect.

The flash based cards like microSD, SD or SDHC cards work in two different bus modes. They are the Secure Digital (SD) bus mode and the Serial Peripheral Interface (SPI) bus mode. The SPI mode is a synchronous serial protocol with less complexity. It is extremely popular for interfacing the peripheral devices and no native-host interface is needed for this mode. For its simplicity and usefulness in the low cost embedded system application, we have considered designing an entirely on-board hardware-based controller for smooth realization of SPI bus mode-based data transfer protocol to communicate with the flash-based memory card.

To date, we find that limited research work describes the design of data archival system and subsequent implementation using HDL [1]. In some research work, SPI mode-based data communication system has been implemented. Another work was proposed in the literature where the SDHC card had been used in SD mode (i.e., bulk data transfer mode) for video signal storage and processing [14]. With the newly emerging technologies, flash-based memory devices have been used as the efficient storage unit and till now it accomplishes the need of memory storage even on the modern era of technological advancement [2, 3, 15–23].

In light of the above, this paper proposes a novel, real-time, low-cost, system-on-chip application specific controller for multimedia data storage and retrieval to flash-based memory cards. The architecture of this real-time controller has been designed using FSM-based approach. The HDL used here is very high speed integrated circuit hardware description language (VHDL). Also, the design is such that there is no use of any on-chip general purpose processor (GPP), external controller, hardware resources, or any high-level languages during

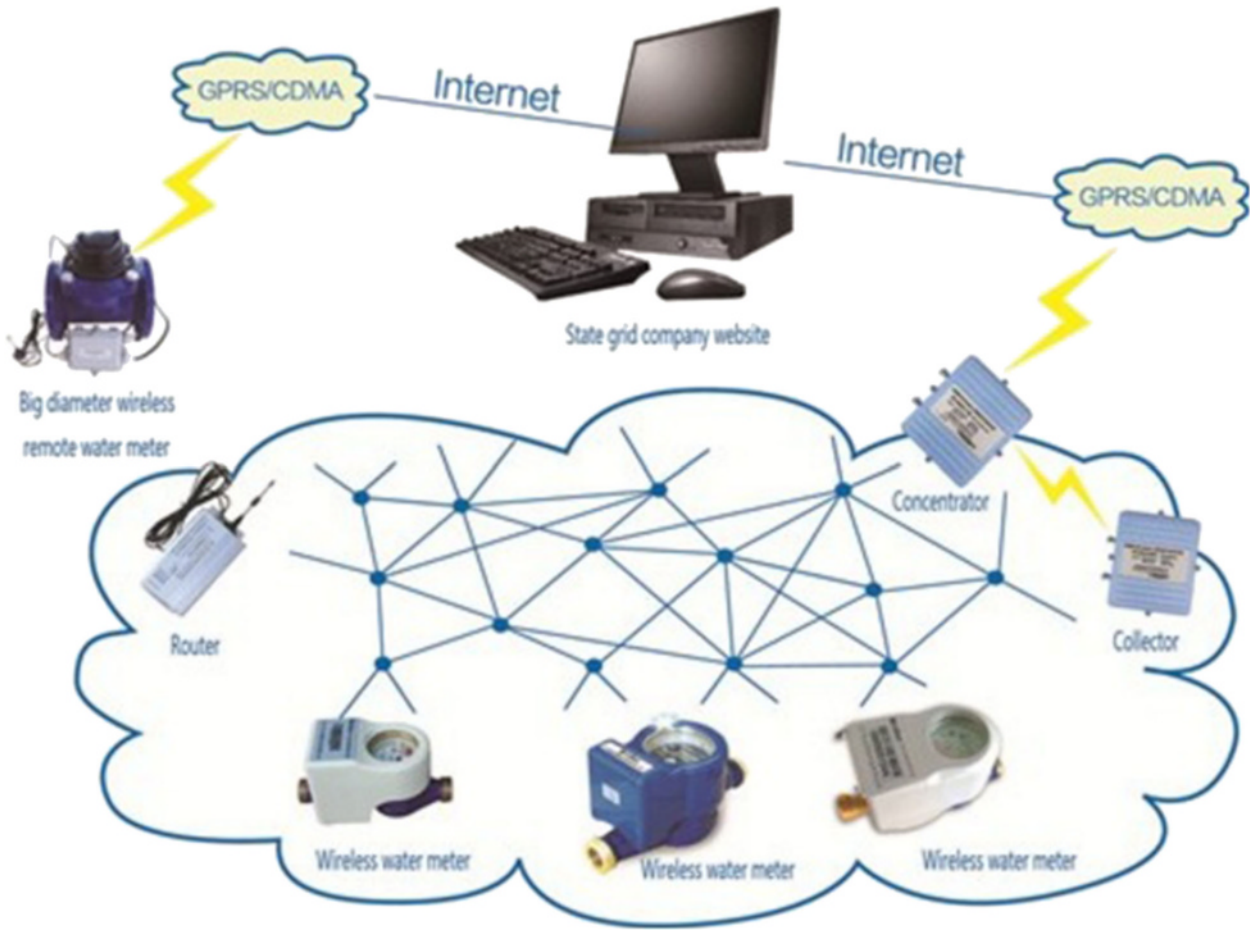


**Fig. 2** Block diagram of a real-time data concentrator system [37]

the operation. The prototype has been entirely implemented in target FPGA board. The physical attribute of an FPGA chip, being compact in size and low in power consumption, makes it an ideal platform for the implementation. Also, we have tried to exploit the parallel processing capability of FPGA during design and implementation of various modules. Till date to the authors' knowledge, optimal-in-hardware implementation of such an application-specific controller and the study of its various modules were not explored in details by earlier research. In this work, the proposed controller has been examined for both the audio and video data storage and retrieval separately. Also to test the importance of the work with respect to practical workloads [24], we have collected the dataset from MSR Cambridge Traces [25], SNIA Iotta Repository [26], and UMass Trace Repository [27] and tried to establish the importance of the controller with respect to flash read-write procedure.

Again in nutshell, the objectives of this paper are as follows:

1. To design a modular low-cost, system-on-chip, application-specific controller for multimedia data storage and retrieval to flash memory cards like SD, SDHC, and microSD card in SPI mode with less overhead.
2. The design is completely FSM based and the controller has been primarily realized using five different modules and a control unit. This five different modules, along with the control unit, are the different functional areas of the proposed system, which is implemented completely in a single chip.
3. When the card is in idle state, the system has an option of working in power saving mode.
4. The controller will work in real time, in modular fashion and the implementation is on a single FPGA. The proposed design tries to utilize the parallel processing capability of FPGA. No other external devices or on-card intelligent controller has been used for this implementation. Here, the prototype has been completely synthesized and tested for Spartan 3E xc3s500e-4-fg320 FPGA.

5. The architecture has been designed using HDL only. Here in this paper, we have considered VHDL for implementing the controller. No high-level languages were used for this design. This FSM-based design using VHDL is one of the basic feature of this proposed controller, which makes it faster than any other controller.
6. It is completely a prototype design of the proposed controller; the design can be implemented in any other platform instead of Spartan 3E target device (even using ASIC also) with a very minor modification in configuration part of the implementation. There will be no change in the design phase.

The rest of the paper is organized as follows. Section 2 introduces the related works and highlights the novelty of the proposed approach. Section 3 describes the proposed FPGA-based controller, its architecture, execution process, and overall operation of the controller. Section 4 presents the hardware-specific implementation and synthesis details for the target Xilinx Spartan-3E (xc3s500e-4-fg320) FPGA development platform. Experimental results are described in Section 5 and Section 6 concludes the paper.

## 2 Review of the related works

FPGAs have been used for prototype design in a range of engineering application [1, 14–19]; however, till date to the authors' knowledge, the design of a complete application-specific controller for different flash memory card access with detailed description of the modules and their operation is limited. Table 1 depicts some of the earlier work in this domain.

Elkeelany et al. [1] proposed an FPGA-based data archival system to SD card, using Verilog HDL and they accessed the card in SPI mode. Scalability issues have not been achieved in this design. They have partially applied FSM based approach and the implementation issues of different SD cards have not been discussed explicitly in this paper.

Yang et al. [14] presented the SDHC card video player based on SoPC technology. The IP core and two display buffer SRAMs were alternately utilized for their proposed design. They have accessed the SDHC card in SD mode for bulk data transfer. The proposed design has been implemented using high-level language.

In another work, Abdallah and Elkeelany [15] proposed a FPGA-based simultaneous multi-channel data acquisition system and they had verified the proposed architecture for analog signals. The design includes analog-to-digital converter to convert the analog signal to digital data. The time-critical tasks were implemented in hardware, while the other tasks were implemented using embedded C. The use of the high-level language in this paper makes the system slower with additional overhead.

**Table 1** Existing papers on SD card controller design

| Work | Controller design | Platform model |
|---|---|---|
| [1] | Data archival to SD card using HDL | Altera Cyclone II |
| [14] | Design of SDHC card video player on SoPC | NIOS II CPU with IDCT hardware acceleration IP core |
| [15] | Simultaneous multi-channel data acquisition system | Altera Cyclone II |
| [16] | NAND flash memory controller for SD/MMC card | Freescale DSP 56858 platform with UMC 0.18 μm CMOS process |
| [17] | Portable analog data capture using custom processing | WOLFSON WM8731 ADC, NIOS-II processor |
| [18] | A high efficient flash storage system for two-way cable modem | TWCNP-OS |
| [Proposed] | FSM-based application-specific controller using HDL | Xilinx Spartan 3E xc3s500e-4-fg320 |

Lin and Dung [16] proposed a novel NAND flash memory controller for SD/Multimedia Card (MMC). They have designed Bose-Chaudhuri-Hocquenghan (BCH) error correction code (ECC) [28] for correcting the random bit errors of the flash memory chip. The UMC 0.18 μm CMOS process was used to implement the proposed memory controller chip. This proposed controller was verified for MMC only.

Elkeelany and Vince [17] proposed a portable analog data capture system using custom processor. The SD card had been used in 1-bit SD mode for their proposed system. The SPI mode or 4-bit SD mode-based communication were not discussed in their design.

The works, summarized in Table 1, have established the concept of FPGA-based implementation for the SD card data archival system either in SD mode or SPI mode. Some researchers [15–18] have taken help of high-level languages or external controllers, on-board processors, and other resources apart from only FPGA logic resources during implementation of data access mechanism. In some paper [14, 15, 23, 29, 30], partially, FSM-based approaches have been used for realization of data transfer mechanism. The single/multiple blocks read/write procedures were designed using FSM, and they have been implemented those procedures using HDL for target device. Also, the BCH code for NAND flash memory has been optimized in previous work [31] and the data-intensive application using FPGA has been performed in earlier researches [32, 33].

Flash-based storage system has other advantages also. Many a time, it has been observed that the efficient data

communication of flash-based system plays a significant role in the improvement of flash resource utilization in many of the systems. Flash resource can be utilized as a cache-based storage system or can be integrated with hard disk drive (HDD) and can act as a hybrid system. Flash resources are also utilized in virtualized storage system [8] where efficient managers are designed to get more high cost effectiveness than normal caching algorithm. Flash-based multi-tiered systems are also studied presently in the literature. Some of them are multi-tier SSD-based solution [34], a hyper-visor-based design [35] etc. Most of the works, however, emphasize on the improvement of caching policies with respect to standard existing caching algorithm like LRU and that analysis is out of scope of this paper. Here we primarily analyzed on multimedia data storage and retrieval to flash-based storage system and this in turn has profound effect on the improvement of flash-based resource implementation.

In our work, we aim to design an application-specific controller for efficient multimedia data communication with flash-based cards in SPI mode and the controller architecture was entirely designed using FSM-based approach. There are mainly five states present in the proposed FSM and the states are named as initialization state, idle state, card-read state, card-write state, and decision-making state. During the realization of the controller architecture, these states are mapped into the modules of the controller. Now some of these modules are used to accomplish card read/write procedures and therefore internal architecture of those modules are again implemented based on FSM format for the realization of above procedures. Note that the proposed architecture and implementation aims to minimize both the clock utilization and on-board resource utilization of the FPGA board. Also in this work, we have considered the required clock cycles, workload, response time, etc. as performance metric to compare the effectiveness of the proposed approach with respect to other existing papers. However, only in the initialization phase, we have represented the performance with respect to "time" metric to compare the achieved results with the reported values in the literature.

## 3 Proposed FPGA-based controller

This section initially describes the basic characteristics of the SD/SDHC card and microSD card and then introduces the proposed controller in rest of the section.

### 3.1 High capacity SD card

The SD/SDHC card communication is based on the advanced nine pin interface, i.e., Clock, Command line/Master Out Slave In (MOSI), 4xData lines/ Master In Slave Out (MISO), and 3xPower lines. The card supports three communication protocols [21]. They are SD 1-bit mode, SD 4-bit mode, and SPI (Serial Peripheral Interface) mode. Table 2 shows the pin configuration of the SD/SDHC card and Fig. 3 shows the thematic representation of electrical interface of the card (slave) in SPI mode with the FPGA board (Master).

The SD/SDHC card communication protocol in SPI mode is entirely a command-dependent protocol and the card responds to every command with a pre-defined response pattern. In the way of initialization, first the card is initiated with CMD0 command. Then, the controller validates the voltage range by generating the CMD8 command. It also identifies the version of the card (version 2 (SDHC) card or some other cards). Subsequently, the controller generates the application-specific commands such as (CMD55 + ACMD41) to complete the initialization process. The controller will continuously generate (CMD55 + ACMD41) command until the card initializes itself by giving a "00000000" response. The SDHC card supports two types of addressing mode. They are block addressing mode and byte addressing mode. The CMD58 command identifies the addressing mode of the version 2 SDHC card. Also, CMD16 command is issued to fix the data block length to 512 bytes. After initialization process, the card goes to the idle state until the next command is being generated for single/multiblock read/write.

The speed class of the card denotes minimum writing performance of the card to record a video normally [36]. Various speed classes defined by SD Association are 2, 4, 6, and 10. Throughout this work, we have used the SDHC and SD card with speed classes 4 and 2, respectively, which means that the SDHC and SD card, used in this purpose, supports minimum 4 and 2 MB/s writing speed, respectively, for video recording.

### 3.2 MicroSD card

The microSD card communication is much similar to the SD card communication. The difference between these two monsters in the present age data storage medium is in their pin configuration. The microSD communication is based on the 8-pin interface where all the pins from the SD card are present except the second ground (Vss2) pin.

**Table 2** SD/SDHC card pin details

| Pin No. | Name | Function in SD mode | Function in SPI mode |
|---|---|---|---|
| 1 | DAT3/($\bar{CS}$) | Data line 3 | Chip select/slave select ($\bar{SS}$) |
| 2 | CMD/DI | Command line | MOSI |
| 3 | Vss1 | Ground | Ground |
| 4 | VDD | Supply voltage | Supply voltage |
| 5 | Clock | Clock | Clock (SCLK) |
| 6 | Vss2 | Ground | Ground |
| 7 | DAT0/DO | Data line 0 | MISO |
| 8 | DAT1/IRQ | Data line 1 | Unused/IRQ |
| 9 | Dat2/NC | Data line 2 | Unused |

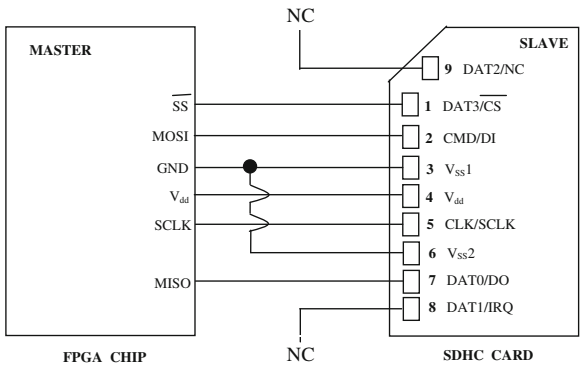


**Fig. 3** SDHC card electrical interface

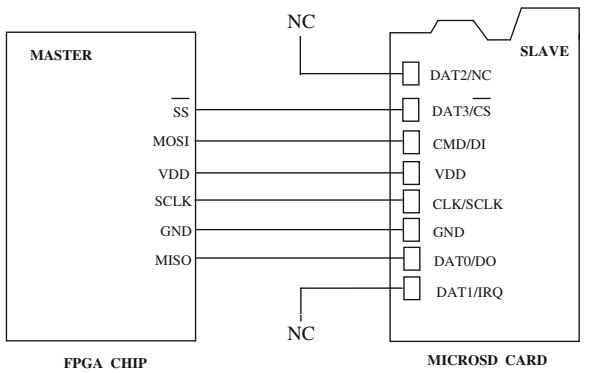


**Fig. 4** MicroSD card electrical interface with FPGA board

Table 3 shows the pin details of the microSD card, and Fig. 4 shows the interfacing of the microSD card with the Spartan3E target FPGA board.

The microSD card communication is also based on command-dependent protocol, and it is almost similar to the SD and SDHC card communication methods. The capacity of the microSD card denotes how it works. If the capacity is less than or equal to 2 GB ($\leq$ 2 GB), then the card works similar to the SD card; otherwise, the principle of operation is the same as the SDHC card.

The definition of the speed class for microSD card is the same as the SD card [36]. We have used the class 4 microSD card throughout the work, which means, the card supports 4 MB/s writing speed for video recording in a normal mode.

### 3.3 Architecture of the proposed controller

The workflow of the proposed host controller is based on the initialization of the card followed by data transfer (read/write) sequences. The overall external view of the controller interfacing the SDHC/SD card is given in Fig. 5. The same process can be used for interfacing the microSD card also. The Vss2 pin remains unconnected when the process is used for interfacing the microSD card as the card contains only 8-pin interface, and the Vss2 pin is not present in the physical architecture of the microSD card.

**Table 3** MicroSD card pin details

| Pin No. | Name | Function in SD mode | Function in SPI mode |
|---|---|---|---|
| 1 | Dat2/NC | Data line 2 | Unused |
| 2 | DAT3/($\bar{CS}$) | Data line 3 | Chip select/slave select ($\bar{SS}$) |
| 3 | CMD/DI | Command line | MOSI |
| 4 | VDD | Supply voltage | Supply voltage |
| 5 | Clock | Clock | Clock (SCLK) |
| 6 | GND | Ground | Ground |
| 7 | DAT0/DO | Data line 0 | MISO |
| 8 | DAT1/IRQ | Data line 1 | Unused/IRQ |

The state diagram of the overall control flow of the controller is shown in Fig. 6, and the internal architecture is shown in Fig. 7. The state diagram of Fig. 6 is working as a backbone for the architecture of the controller. The complete architecture has been implemented using VHDL.

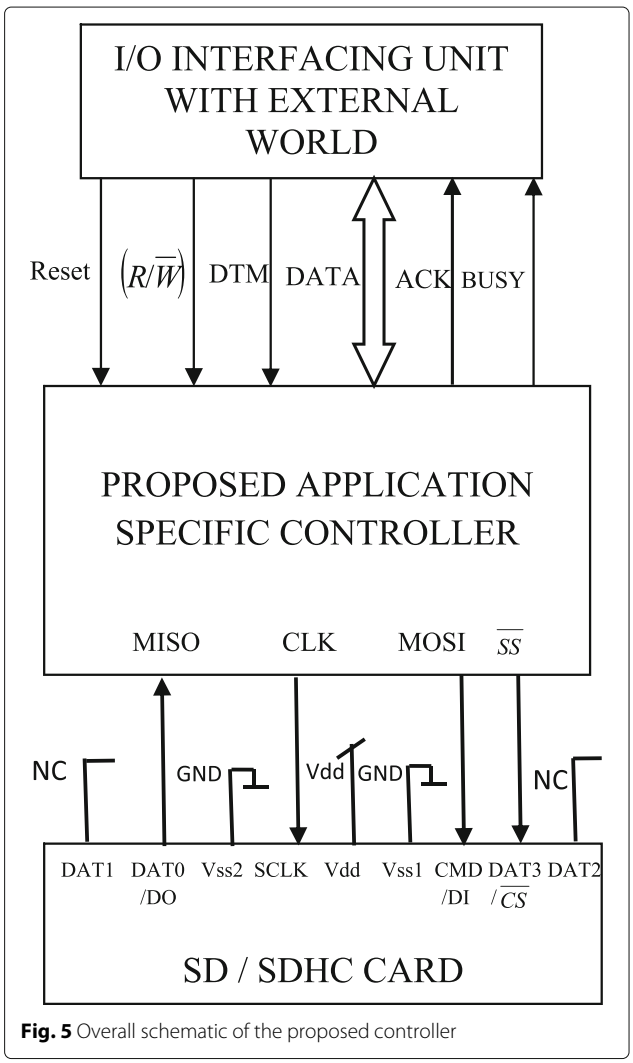


**Fig. 5** Overall schematic of the proposed controller

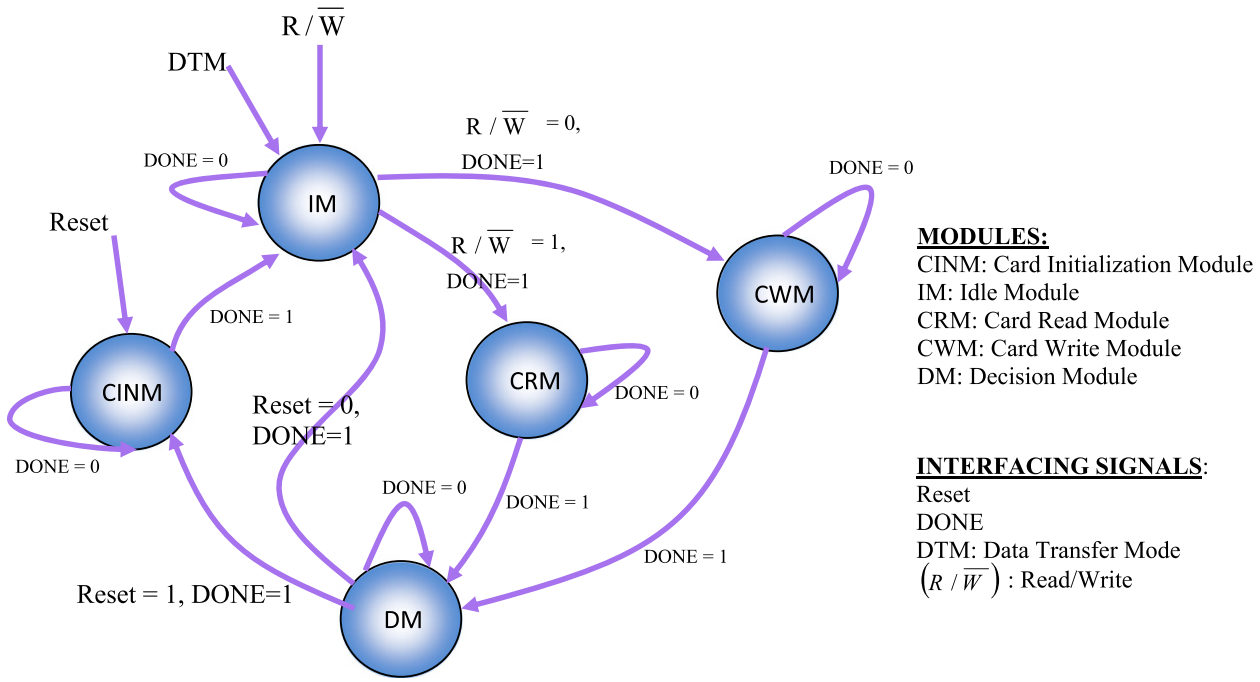


**Fig. 6** State diagram of the controller

As we observe from the above mentioned flow sequence and the schematic of the internal architecture, the proposed controller is divided into five different modules. They are card initialization module (CINM), idle module (IM), card read module (CRM), card write module (CWM), and decision module (DM). Along with the above modules, a control unit (CU) is there to monitor and control the activities of each module and the flow of respective driving signals. The CU operates based on the FSM shown in Fig. 6.

Each of the modules and CU contains several internal and external data and control lines. The communication with the external world is done by the controller either using I/O interfacing units or a customized multiplexer. The Reset, DTM (Data Transfer Mode), R/$\bar{W}$ , DATA, ACK, and BUSY signals are interfaced with the controller via I/O interfacing unit. The Reset signal, connected with CU, initiates the data storage or retrieval operation. DTM signal selects the single/multiblock data transfer mode of the controller, R/$\bar{W}$ is used for read/write operation selection, and a 8-bit bidirectional DATA bus is used for communication with external world. Also, other signals like Clock, Chip Select (CS), MISO, and MOSI signals are connected between SDHC card and the controller through a (4×1) bi-directional customized multiplexer where each input line of the multiplexer is a 4 bit width data bus. Each data bus consists of Clock, Chip Select ($\bar{CS}$), MISO, and MOSI signals. These four input buses of the multiplexer connect the four modules, say, CINM, IM, CRM, and CWM of the controller with the external card based on the SELECT bus. The output bus from the multiplexer communicates with the card. Only in the data bus from the IM, the clock signal remains unconnected to realize the power saving mode of the controller. Therefore as a whole, the designed multiplexer has 2 bit SELECT bus (S1 and S0), 4 input bus lines ($4 \times 4 = 16$ lines) and one output bus line ($1 \times 4 = 4$ lines). SELECT bus connected with the multiplexer in sequence helps to communicate the individual modules with the card, and CU controls the entire selection process. The module selection activities of the SELECT bus is described in Table 4.

BUSY and ACK are the two status signals present in the controller and they are connected with CU. The BUSY signal represents the busy state of the controller and ACK signal acknowledges any assigned work accomplished by the controller. On completion, the module deactivates the BUSY signal and activates the ACK signal to intimate the user that the task has been completed successfully. Failure to complete any assigned task makes both the BUSY and ACK signal de-asserted. The activity of the signals is tabulated in Table 5.

The CU also internally communicates with every module in sequence for efficient data transfer with the card. The common signals for all the modules are Reset, CS, and clock signal. The CS and clock signals are also supplied to the card via multiplexer. Once Reset signal is received by CU, it issues a START signal to the CINM along with the clock signal. CU also issues START and clock signal

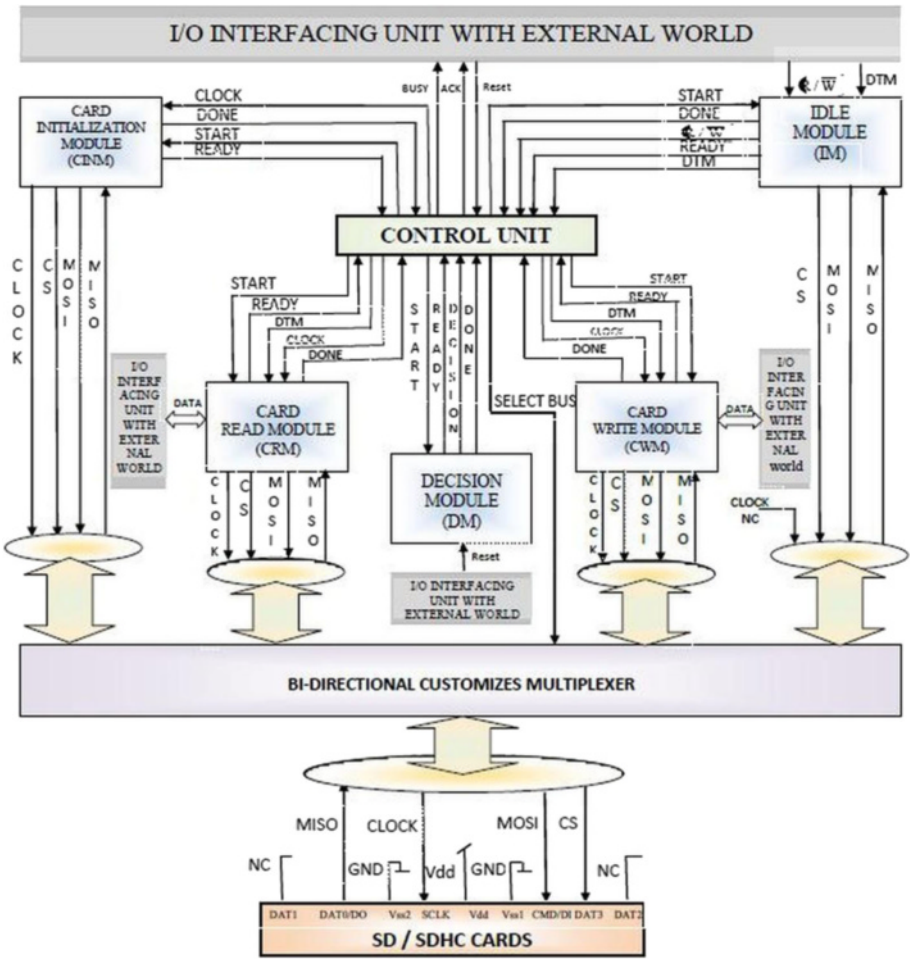


**Fig. 7** Architecture of the proposed controller

for the other modules when they are about to initiate their action. In idle state, no clock is received by the modules and thus they work in power saving mode. Once a module receives a START signal, it acknowledges so by issuing a READY signal to the CU and starts working. After every successful completion of the work, the module intimates CU with DONE signal.

CU starts working with the card initialization module. On receiving the Reset signal, CU activates the CINM and it issues the initialization commands to the card in order to initialize it in SPI mode. The card responds to every command and on completion of initialization procedure, the module receives the final response from the card.

**Table 4** Table for SELECT bus activity

| S1 | S0 | Selected module |
|---|---|---|
| 0 | 0 | CINM |
| 0 | 1 | IM |
| 1 | 0 | CRM |
| 1 | 1 | CWM |

After successful initialization of the card, the control transfers to IM with the START signal. The IM continuously monitors the R/$\bar{W}$ and DTM signal. R/$\bar{W}$ signal specifies whether the next operation will be card read or card write. The DTM signal will intimate the IM regarding the single/multiblock data transfer. Whenever it receives the R/$\bar{W}$ signal, it passes both the R/$\bar{W}$ and DTM value to CU and goes to the idle state by sending a DONE signal. Depending on the R/$\bar{W}$ signal, the CU transfers the control either to CRM or to CWM.

The CU generates different command sequences for either of these modules. In card read sequence, the CRM reads the data block from the card along with the CRC bits and publish it to the I/O interfacing unit. In card write

**Table 5** Table for the status signal activity

| BUSY | ACK | Controller status |
|---|---|---|
| 0 | 0 | Time out, controller failed to read/write |
| 0 | 1 | Task successfully accomplished |
| 1 | X | Controller is busy |

sequence, the CWM receives the data block as an input from the I/O interfacing unit and writes the received block along with CRC to the card and receives the CRC response from the card. After read/write operation, CRM/CWM issues a DONE signal and itself goes to the power saving mode. Finally after completion of entire data transfer, controller generates the ACK signal to the external world through the I/O interfacing unit to intimate that the work is successfully completed.

The DM monitors the Reset signal received from the I/O interfacing unit during operation. Assertion of Reset signal means that CU will again reinitialize the process by activating the CINM. If the Reset signal remains de-asserted, then the CU will activate the IM and again follow the previous sequence of operation for continuous data transfer. The DM gives its decision to the CU by the DECISION signal and goes to power saving mode by issuing a DONE signal.

### 3.4 Overall system operation

The flow of execution and communication between the individual units of the controller and SDHC card is now described. The similar procedure is also applicable for SD and microSD card-based data storage and retrieval. All of them actually works in three different phases namely card initialization phase, card read phase, and card write phase. Previously stated five modules are the astringent of these three phases. The controller communicates with the external world through the I/O interfacing unit and the customized multiplexer. The controller initiates its operation on reception of Reset, DTM, and R/$\bar{W}$ signals and sends the signals to CU. Then it intimates regarding its status using the BUSY and ACK signal. When the controller is busy in processing some task, it makes the BUSY signal high until the task is accomplished. After every successful completion of a process, the controller informs the outer world by asserting the ACK signal and de-asserting the BUSY signal. If the controller fails to complete the task given, then it de-assert both the BUSY and ACK signal. Table 5 describes the operation of the two status signals.

The pseudo-code for the data transmission process to the card is given in Figs. 8, 9, 10, 11, 12 and 13, and the FSM of the controller and the architecture are essentially inspired form the activities described in the codes. The rest of the section describes the operation of different modules.

*Card initialization module (CINM)*: The CINM initiates the SDHC card in SPI mode. Figure 14 contains the FSM of the initialization module. The START signal activates the CINM and it acknowledges the CU with READY signal. The module first elapses 74 or more clock cycles for initiating the card in SPI mode. Then, the commands are generated to complete the initialization process. After completion of the initialization process, the controller validates the addressing type for the SDHC card (either block addressing or byte addressing) by asking to publish its card capacity status (CCS) bit in operational control register (OCR). The high value of the CCS bit in OCR register means that the card is a version 2 SDHC card supporting block addressing mode, and the low value of CCS bit refers the version 2 (SDHC) card supporting byte addressing mode. If the result matches with byte addressing mode, the controller then generates the next command to forcefully make the block length to 512 bytes. On completion of the initialization process CINM will issue a DONE signal.

**Power on initialization of the system**

```
1. Power on
2. If reset = '1' then
3.     Start Initialization (internal signals and states)
4. Else
5.     Start Initialization (Card)
6. End if;
```

**Fig. 8** Psudo-code for power on sequence

*Idle module (IM)*: The idle module works in two modes. They are polling mode and control transfer mode. IM continuously polls the R/$\bar{W}$ signal and DTM signal. Depending upon the status of these two controlling signals, CU transfers the control either to CRM or to CWM. On completion of the operation, the DONE signal is asserted to CU. On polling mode, the controller is in power saving state as the card is in the idle state. The controller sends a constant high value to the MOSI line, and the card also responds with a constant high value via MISO line of the controller.

*Card read module (CRM)*: The SDHC and similar type of cards support two types of data transfer, one is the single block data transfer and another is the multiblock data transfer. The left branch of the flow chart in Fig. 15 describes the read operation from the card. The CRM has been designed to read the data blocks from the SDHC card. The CU issues two different commands for CRM

**Initialization (card) in SPI mode**

```
1. Send initialization commands
2. If received final response = TRUE then
3.     Go to IDLE state
4. Else
5.     Resend Initialization commands
6. End if;
```

**Fig. 9** Psudo-code for initialization (card)

**Idle state**

```
1.  If wr = '1' then
2.      Go to Card Write process
3.  Elsif rd = '1' then
4.      Go to Card Read process
5.  Else
6.      Remain in IDLE state
7.  End if;
```

**Fig. 10** Psudo-code for idle state

depending upon the DTM signal value. One is for single block data transfer and another is for multiblock data transfer. MOSI line of the controller is connected with the CMD/DI line of the card, and similarly MISO line of the controller is connected with DAT0/DO line of the card. After successful transmission of the command via MOSI line of the controller, CRM receives the command response from the card through MISO and then starts reading the data block(s) from the card along with the CRC bits. The block transfer is preceded by a start block token "11111110" along with a block of data, which is followed by the CRC.

The multiple block transfer can be terminated by the command CMD12, generated by the controller.

*Card write module (CWM)*: The CWM has been designed to write the data block to the SDHC card. The right branch of the flow chart in Fig. 15 describes the write

**Card Write Process**

```
1.  If data mode = '1' then
2.      Send single block write command
3.  Else
4.      Send multi block write command
5.  End if;

6.  If received response  = TRUE then
7.      Go to decision process
8.  Else
9.      Restart Card Write process
10. End if;
```

**Fig. 11** Psudo-code for card write process

**Card Read Process**

```
1.  If data mode = '1' then
2.      Send single block read command
3.  Else
4.      Send multi block read command
5.  End if;

6.  If read process completed then
7.      Go to decision process
8.  Else
9.      Restart Card Read process
10. End if;
```

**Fig. 12** Psudo-code for card read process

operation. On the way of execution, CU transfers the control to CWM along with the DTM signal to ensure a single block data write or a multiblock data write. In this implementation, both single and multiblock write operation have been taken into consideration. For single block write operation, the controller generates the command with the starting address and then it starts writing 512 bytes of data. For multiblock write, the controller writes the data block until the CMD12 stop command is being issued. The CRC bits are appended to each data byte for the entire write operation. The card sends back the response pattern in the MISO line of the controller, where "XXX00101" means the data block is accepted, "XXX01011" means the data block is rejected due to the CRC error, and "XXX01111" indicates that the data block is rejected due to the flash program error (in the pattern, "XXX" means do not care bits). The multiple block write improves the

**Decision process**

```
1.  If reset = '1' then
2.      Start Initialization (card)
3.  Else
4.      Go to IDLE state
5.  End if;
```

**Fig. 13** Psudo-code for decision process

throughput as a single command is generated for bulk of data blocks write procedure.

*Decision module(DM)*: The DM has been designed to decide the destination of the controller after completing the data transfer operation viz. whether the control will go to the IM or to the CINM. It is required for successive sequence of data transfer. After performing the data transfer process, the control comes to the DM. The DM constantly monitors the Reset signal and depending on the status of the Reset signal, it decides whether the control will go to the IM for performing the next data transfer operation or it will go to the CINM for initialization of the card.

The ACK signal is finally issued by the controller through the I/O interfacing unit, to intimate the user that the work given to the controller has been completed successfully and it is ready to process next set of operation.

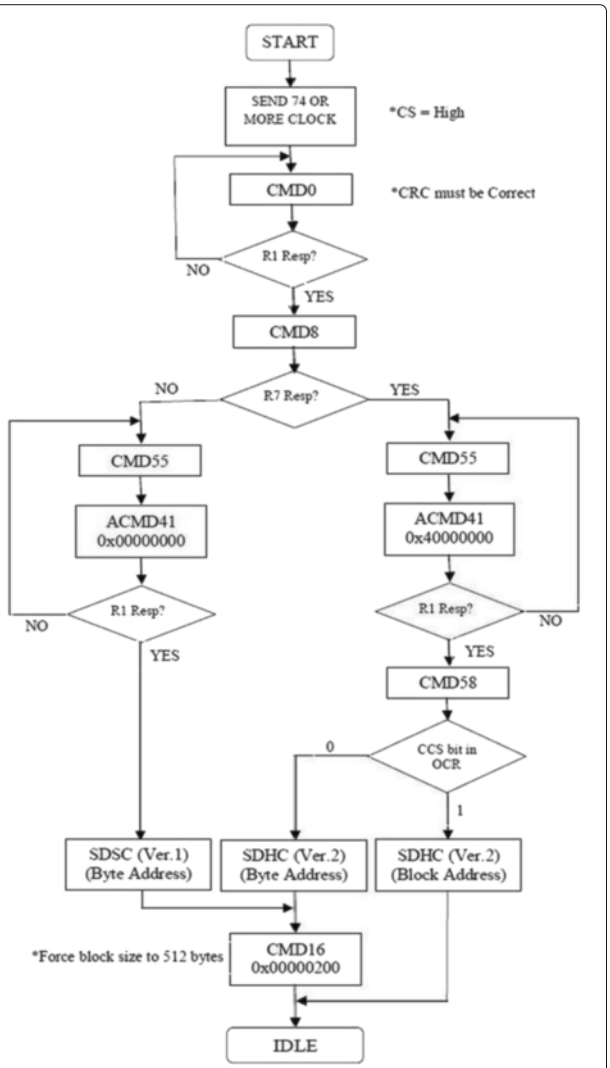


**Fig. 14** Flowchart for the initialization of SD/SDHC card

## 4 Hardware-specific implementation details

To explore the feasibility of the proposed architecture, the FPGA-based controller was implemented with the help of synthesizable VHDL. This actually reduces the processing time of the proposed controller than any other high-level languages. The target development platform is based on Spartan-3E (xc3s500e-4-fg320) FPGA chip. The card has been accessed through a multi-port card reader connected with the target FPGA board using 6-pin cable. A SanDisk 8 GB SDHC card, a Cannon 16 MB SD card, and a 2 GB microSD card, with speed classes 4, 2, and 4, respectively, have been used for testing and verification of the proposed controller.

### 4.1 Application-specific controller initialization challenges

From the hardware point of view, traditionally, the personal computer accesses the flash memory through a permanent device interface, which implies a fixed point access of the memory. In this proposed design, the system comes up with a detachable unit to be connected with the memory card so that the card can be accessed anywhere.

Now from the designers' point of view, synchronization of clock throughout the design is a big challenge to the designer as the flash memory card requires 100- to 400-KHz clock frequency for the application specific command execution during initialization [3] and maximum 25-MHz clock frequency for data transfer [1]; which ultimately points to the decrement of clock frequency and it implies sacrifice in performance in a single clock input system. If the clock frequency does not match in initialization process, then the card uses to poll the command in a 50-ms time gap to complete the initialization process.

### 4.2 Synthesis details

The design was successfully synthesized using Xilinx ISE version 14.1 for the Spartan-3E XC3S500C target device, and then it was compiled and built for implementation. This process consists of translating, mapping, placement, and routing of the signals. For the design implementation process, no partition was specified and the design was translated and mapped successfully. All signals were placed and routed successfully as well; all the timing constraints were met. Five percent of the total logic slices on the device was utilized for this implementation. Brief description of the resource utilization during implementation is given in Table 6.

The number of logic slices utilized for the proposed controller is 226 out of 4656 (5%).

## 5 Experiments and results

In this section, we have tested the working performance and efficiency of the proposed controller extensively for different synthetic dataset and practical work loads. Initially, we have defined the metrics used to evaluate

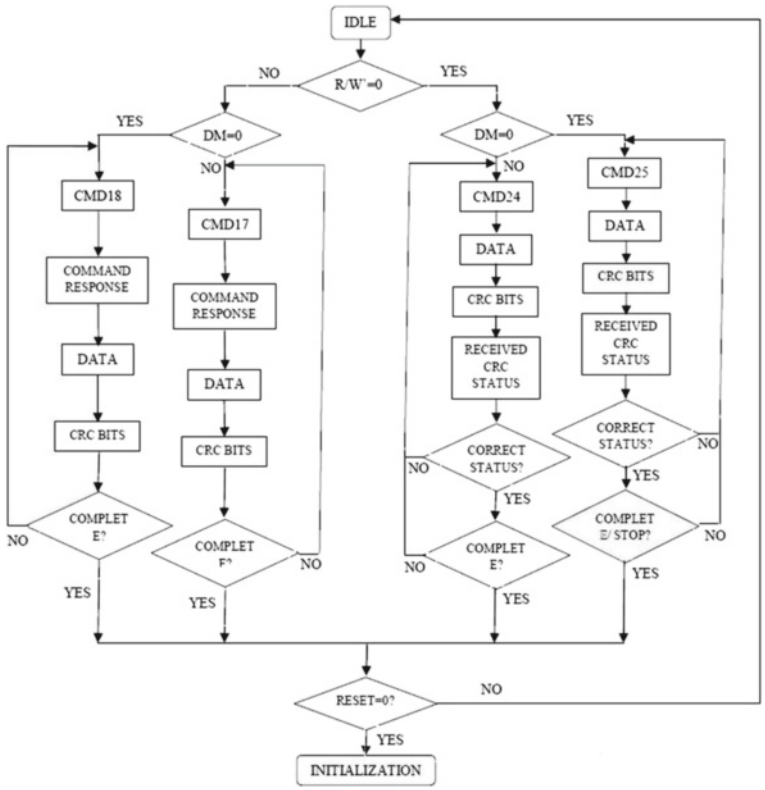


**Fig. 15** Flow chart for read and write operation of SD/SDHC card

the performance of the controller. Subsequently, different synthetic and practical workloads were described as case studies of the paper.

### 5.1 Performance metrics

In this paper, we have used following metrices to evaluate the performance of the controller. They are defined as follows.

*Clock cycle taken per process (CCTPP)*: In order to make the performance evaluation of the proposed controller independent of system clock and other system specific parameters, we have described the performance of the controller in terms of clock cycles. Here, CCTPP describes the clock cycle taken by an individual process to successfully complete a process. If we navigate the proposed design into multiple systems, then CCTPP parameter will be system independent.

**Table 6** Resource utilization during implementation

| Logic blocks | Number of logics used | Number of logics available | Utilization (%) |
|---|---|---|---|
| Logic slices | 226 | 4656 | 5 |
| Slice flip/flops | 167 | 9312 | 2 |
| Slice LUTs | 421 | 9312 | 6 |
| Clock buffer | 1 | 24 | 4 |
| Number of bonded IOBs | 15 | 232 | 6 |

*Time taken per process (TTPP)*: This is the time taken by the individual process to accomplish the task assigned successfully. We have used the conventional units of time to define the TTPP.

*Input/output operations per seconds (IOPS)*: IOPS is a measurement process used to characterize the storage devices like flash storage memory. The IOPS is not defined independently and it is a combination of three metrics. Along with IOPS other two metrics, say response time and workload metrics are also defined to characterize the performance of the memory module.

### 5.2 Simulation details

The proposed architecture was first simulated in the Xilinx ISE 14.1 virtual environment for initial verification. In this phase, Fig. 16 gives the waveform representation of the initialization process completely. Figure 17 represents the timing diagram representation of the single block write, and Fig. 18 gives the timing diagram representation of the multiblock data write operation. The simulation results are only feasible for data write operation and the implementation section represents the entire results for initialization and both the read/write operation. Table 7 elaborates the different abbreviation of the input and output ports used to simulate the design.

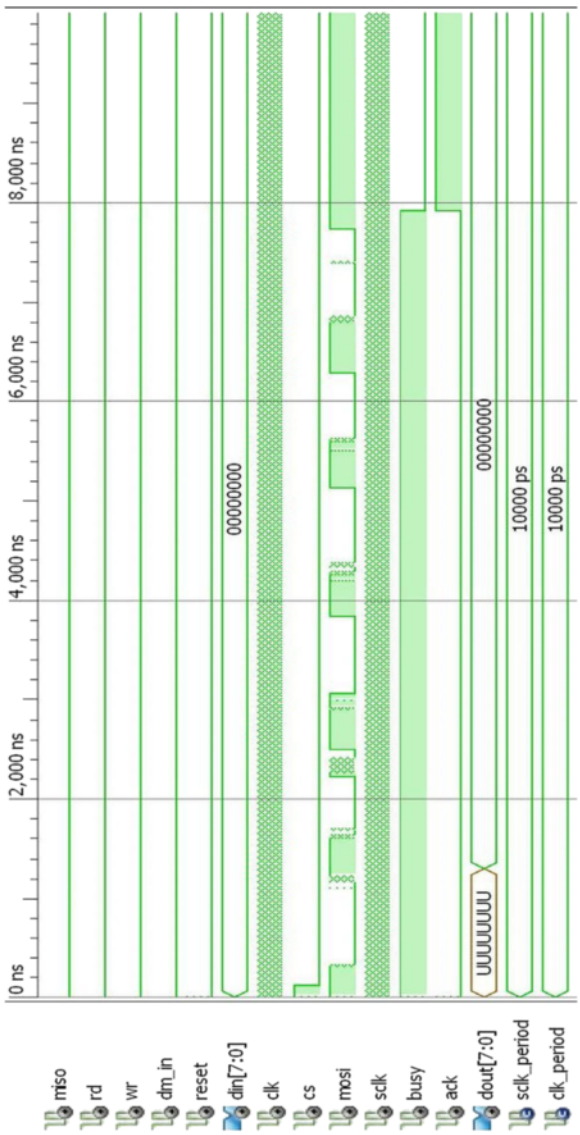


**Fig. 16** Simulation waveform for initialization process

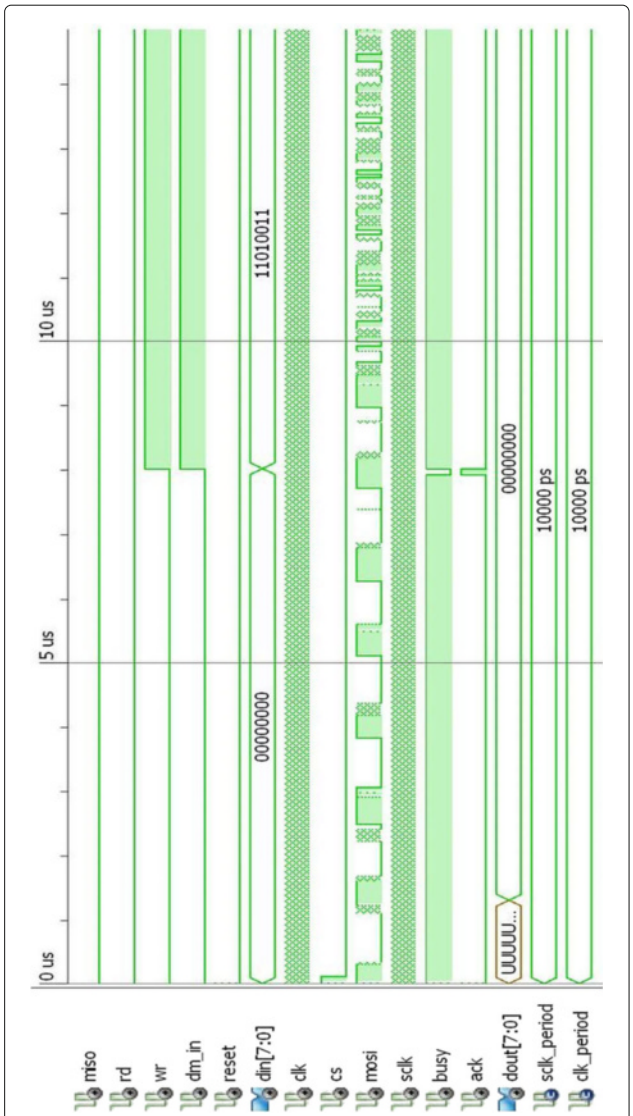


**Fig. 17** Simulation waveform for single block write process

### 5.3 Implementation details

The proposed controller has been tested using various case studies with different type of input data pattern.

#### *5.3.1 Case studies*

Seven different case studies have been performed to verify the effectiveness of the proposed controller for SD, SDHC, and microSD cards with varying volume size of the data. For first four cases, synthetically, text, audio, and video patterns were generated. Other three datasets were taken from practical workloads representing block I/O traces from MSR Cambridge Traces [25], SNIA Iotta Repository [26], and UMass Trace Repository [27].

*Case I: pre-declared embedded data pattern*: The first case study uses a pre-declared embedded data pattern with 8-bit data. The proposed controller tests the basic read/write operation for a Cannon 16 MB SD card using this data pattern. The same 8-bit data pattern has been written repeatedly to the SD card for testing the single and multiblock data write operation. Later on, the same data pattern has been retrieved to validate the single as well as multiblock data read operation from the same SD card.

*Case II: pseudo-random number sequences*: The second case study has been performed for testing the proposed controller for SDHC card. A pseudo-random number (PRN) generator has been designed to generate 8-bit random data pattern continuously to perform the single block as well as the multiblock write operation for a SanDisk 8 GB SDHC card. The previously used Cannon 16 MB SD card and a 2 GB microSD card have also been tested for single and multiblock read/write operation using those PRN sequences.

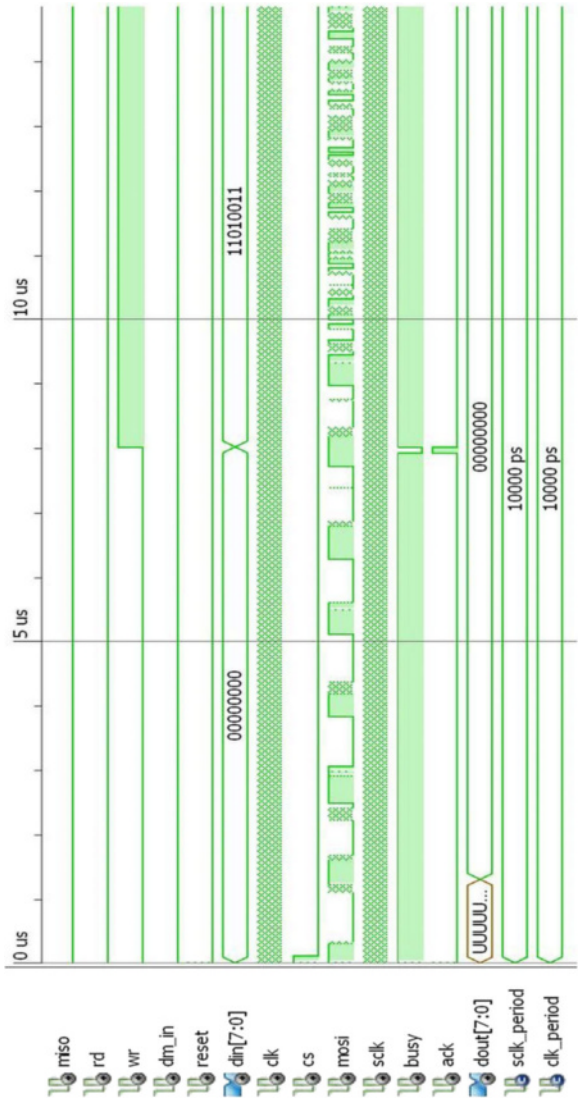


**Fig. 18** Simulation waveform for multiblock write process

*Case III: audio signal transmission*: The third case study has been performed for testing the efficiency of the proposed controller with continuous data transfer. An analog signal in audio range is fed to an on-board LTC 1407A analog-to-digital converter (ADC) [20]. The digital data at the output of the converter is fed to the controller for data storage and retrieval. The LTC6912-1 available in the on-board ADC provides two independent inverting amplifiers with programmable gain to maximize the conversion range of the ADC to 1.65 ± 1.25 V. The gains for both the channels are independently programmable using a 3-wire SPI interface to select voltage gains among "0," "−1," "−2," "−5," "−10," "−20," "−50," and "−100" V/V (LTC6912-1) [20]. The analog-to-digital conversion formula is given below in Eq. 1:

$$D[13:0] = \text{GAIN} \times [(V_{\text{IN}} - 1.65\ \text{V})/1.25\ \text{V}] \times 8192 \quad (1)$$

**Table 7** Abbreviation of the ports in simulation waveform

| Abbreviation | Full name |
|---|---|
| miso | Master in slave out |
| rd | Read |
| wr | Write |
| dm_in | Data mode selection |
| reset | Reset |
| din | Data in |
| clk | Clock |
| cs | Chip select |
| mosi | Master out slave in |
| sclk | Slave clock |
| busy | Busy signal |
| ack | Acknowledgment |
| dout | Data out |

Here, $D[13:0]$ represents the 14-bit 2's complement value of the analog input. The maximum sample rate for the ADC is approximately 1.5 MHz [20]. The 14-bit digital output from the ADC has been compressed to 8-bit digital data pattern, so that the proposed 8-bit controller can easily read the data byte for storing into the SDHC memory card. On later phase, the retrieval of the stored data from the same SDHC card indicates the memory read operation for the analog signal.

*Case IV: video signal transmission*: The fourth case study is performed with video signal storage and retrieval. The on-board DB15 VGA connector port [20] has been used to display the video frames in a CRT monitor. The VGA signal timing is specified, published, copyrighted, and sold by the Video Electronics Standards Association (VESA) [20]. Thus in this paper, the detail specifications of the processing video signals are not mentioned in depth.

The frames of the video signal have been generated and then decoded in the binary-numbered matrix. Then, the data from the generated matrix have been stored in the card for testing the card write operation. Here, the multiblock data write operation has been performed as the volume of data is too high to perform the single block write operation. On later phase, the previously written data have been retrieved from the same card for validating the multiblock card read operation process. The FPGA board was connected with a CRT monitor through a VGA cable for displaying the retrieved result (i.e., the video signal) from the card. The CRT monitor, used in this case study, has the horizontal frequency of 90 kHz and the screen resolution of 1024 × 768 pixels with color quality of 16 bits. So, the refresh rate of the monitor is 72.

*Case V: dataset from MSR Cambridge Traces*: The Microsoft Research (MSR) Cambridge traces [25] are a commonly used data repository, built by Microsoft research group. This repository was formed to advance

the state of the art computing and also to solve the practical world problems through technological innovation. This is a collaboration of the academics with the government and industry researchers. There are 35 different traces available for this MSR Cambridge Traces and they represent 1-week block I/O traces of enterprise servers at Microsoft Research Cambridge. The characteristics of the traces are given along with file names, their attributes, file size, etc. in Table 8. We have used those datasets to check the performance of the proposed controller in multiblock data transfer procedure. The SanDisk 8 GB SDHC card has been used as the flash storage in this case study.

**Table 8** Characteristics table for MSR Cambridge Repository [25]

| Repository name | Abbre--viation | File name | File size in KB | Attributes |
|---|---|---|---|---|
| MSR | M11 | CAM-02-SRV-lvm0 | 227,577 | Timestamps, |
| Cambridge | M12 | CAM-02-SRV-lvm1 | 1,576,883 | Hostname, |
| 1 | M13 | CAM-02-SRV-lvm2 | 117,104 | Disk |
| | M14 | CAM-02-SRV-lvm3 | 1,274,935 | Number, |
| | M15 | CAM-02-SRV-lvm4 | 1,274,935 | Type, |
| | M16 | CAMRESHMSA01-lvm0 | 197,530 | Offset, |
| | M17 | CAMRESHMSA01-lvm1 | 29,069 | Size, |
| | M18 | CAMRESISAA02-lvm0 | 643,399 | Response |
| | M19 | CAMRESISAA02-lvm1 | 8,832,021 | Time |
| | M110 | CAMRESWMSA03-lvm0 | 62,457 | |
| | M111 | CAMRESWMSA03-lvm1 | 86,267 | |
| | M112 | CAM-USP-01-lvm0 | 284,762 | |
| MSR | M21 | CAM-01-SRV-lvm0 | 115,703 | Timestamp, |
| Cambridge | M22 | CAM-01-SRV-lvm1 | 2,393,907 | Hostname, |
| 2 | M23 | CAM-01-SRV-lvm2 | 560,593 | Disk |
| | M24 | CAMRESIRA01-lvm0 | 75,897 | Number, |
| | M25 | CAMRESIRA01-lvm1 | 739 | Type, |
| | M26 | CAMRESIRA01-lvm2 | 10,894 | Offset, |
| | M27 | CAMRESSDPA01-lvm0 | 2,028,692 | Size, |
| | M28 | CAMRESSDPA01-lvm1 | 2,476,675 | Response |
| | M29 | CAMRESSDPA01-lvm2 | 98,864 | Time |
| | M210 | CAMRESSDPA03-lvm0 | 80,578 | |
| | M211 | CAMRESSDPA03-lvm1 | 34,773 | |
| | M212 | CAMRESSDPA03-lvm2 | 61,550 | |
| | M213 | CAMRESSTGA01-lvm0 | 103,206 | |
| | M214 | CAMRESSTGA01-lvm1 | 113,632 | |
| | M215 | CAMRESTSA01-lvm0 | 91,123 | |
| | M216 | CAMRESWEBA03-lvm0 | 105,051 | |
| | M217 | CAMRESWEBA03-lvm1 | 8,308 | |
| | M218 | CAMRESWEBA03-lvm2 | 299,650 | |
| | M219 | CAMRESWEBA03-lvm3 | 1611 | |
| | M220 | CAMWEBDEV-lvm0 | 58,850 | |
| | M221 | CAMWEBDEV-lvm1 | 55 | |
| | M222 | CAMWEBDEV-lvm2 | 9369 | |
| | M223 | CAMWEBDEV-lvm3 | 35 | |

*KB* kilobytes

*Case VI: dataset from UMass Trace Repository*: The UMass Trace Repository [27] is a commonly used data repository. It provides storage, network, and other traces for analysis to the research community. This work is supported by the National Science Foundation. This repository contains different traces namely CPU and memory, network, Storage, weather, power, smart, and multimedia traces under two different categories, namely Financial and WebSearch. The characteristics of the traces are given in Table 9. We have used those datasets to verify the multiblock data transfer process of the proposed controller and the SanDisk 8 GB SDHC card has been used for this purpose.

*Case VII: dataset from SNIA Iotta Repository Historical Section*: Storage Networking Industry Association (SNIA) Iotta Repository [26] is a commonly used repository, used to store, manage, and distribute different traces or datasets for storage. The historical section of this repository includes all the traces which are older than 10 years. The historical section contains five different traces namely Block I/O Traces, Network File System Traces, Parallel Traces, Static Snapshots, and System Call Traces. Each of these traces are further divided into multiple sub-traces. We have used some sub-traces from those available traces to verify the multiblock data transfer process of the proposed controller. The characteristics of the traces are given in Table 10. The SanDisk 8 GB SDHC card has been used as the storage device in this case study.

*Different flash transition layer settings*: The functionality and efficiency of the proposed controller have also been tested using different volume of data. We have tested the read-write operations of the controller for different volume likely 4, 8, 10, and 512 MB and 1 GB.

#### *5.3.2 Results*

The implemented architecture of the proposed controller works in three different modes; namely card initialization mode, card read mode, and card write mode. After card initialization, based on the external commands, controller communicates with the card either for read or for write operation.

**Table 9** Characteristics table for UMass Repository [27]

| Repository name | File name | File size in KB | Attributes |
|---|---|---|---|
| UMass | Financial1 | 151165 | ASU, |
| | Financial2 | 102873 | LBA, |
| | WebSearch1 | 30744 | Size, |
| | WebSearch2 | 135948 | Opcode, |
| | WebSearch3 | 127520 | Timestamps, |
| | | | Optical fields |

*ASU* application-specific unit, *LBA* logical block address

**Table 10** Characteristics table for SNIA Historical Section Repository [26]

| Trace name | Sub-traces | Number of files |
|---|---|---|
| Block I/O Traces | Cello 1999 | 12 |
| | Cello 1996 | 12 |
| | Cello 1992 | 12 |
| | HP LAJW | 12 |
| | Cello 1991 | 12 |
| NFS Traces | Animation dataset | 140 |
| | Harvard SOS Traces | 7 |
| Parallel Traces | Sprite Traces | 1 |
| Static Snapshots | Multimedia file sizes | 1 |
| | Microsoft Longitudinal Study | 1 |
| | Microsoft 1998 Static Study | 1 |
| System Call Traces | LASR Traces | 13 |
| | Seer Traces (ASCII) | 1 |
| | Seer Traces | 1 |
| | CMU DFS Traces | 14 |

To validate the read/write operation two types of test unit has been considered. They are described as follows:

*Test unit: on-board output unit (LED)*: The Spartan 3E target FPGA board contains 8-bit output unit with 8 on-board light emitting diodes (LEDs) [20]. In the testing setup of the experiment, the MISO signal of the controller has been mapped with a single LED of the on-board 8-bit LED unit and the resultant command responses and read/write data patterns were observed. Since the SPI mode is a serial bus interface mode, the SD card gives response serially through its output port to the master. The MISO line signal is left shifted in every clock cycle to observe a series of patterns in 8-bit LED unit available in the FPGA board. Additionally, in the output section, a clock down converter unit has been designed and integrated to slow down the speed of response of the controller for better perception. This is a little compromise with the speed of operation in testing and verification section. However, introduction of this section is purely optional and while integration of the controller with real-time high speed system this unit is to be omitted.

*Test unit: DSO*: A digital storage oscilloscope (DSO) has been connected with the input (MOSI) and the output (MISO) port of the controller. Both the output and input data patterns have been observed in DSO for further verification. The clock down converter unit has also been integrated here to slow down the response speed of the card. But, it is introduced only for the card read section for all the cards and initialization section for the SDHC card only.

The results of the proposed controller have been observed and verified in chronological order as per problem description stated above.

*Case I: result: pre-declared embedded data pattern*: The first case study represents the complete three-state access of a cannon 16 MB SD card. The steps in different phases of data transfer are described below:

*Initialization* - Figures 19 and 20 show the initialization state output mapped in the on-board LEDs. The driving signal for the initialization module is the Reset signal. Figure 19 shows the output response pattern of the SD card after execution of the first command (CMD0). Figure 20 shows the output pattern after completion of the entire initialization step, when the card is in idle mode.

*Card write operation* - The pre-defined 8-bit embedded data pattern has been written to the SD card. Figure 21 shows the CRC status response (viz. "00000101") coming from SD card to the output port. The card responds to every successful write of an entire block. If the operation performed is a single block write, then the card gives the response once after the completion of the write operation of the entire block. But if it is a multiblock write operation, then the card responds after every block write completion, until the end block command (CMD12) has been issued.

*Card read operation* - Here, we consider an arbitrary data pattern, say e.g., "11010011" which has been read from the SD card. Figure 22 shows that data pattern on the LEDs. Both the single block and the multiblock write and read operations have been verified with the data pattern. This 8-bit data pattern has been written repeatedly to form the entire block (of 512 bytes) to perform the block write operation.

*Case II: result: pseudo-random number sequences*: The second case study represents the complete 3-state access for a SanDisk 8 GB class 4 SDHC card. Both the input and output ports are connected with the DSO. The study shows the efficiency of the proposed controller for high capacity SD card. Later on, the 16 MB Cannon SD card and the 2 GB microSD card have also been verified for the read/write process.


**Fig. 19** CMD0 response pattern on the on-board LEDs for SD card


**Fig. 20** Initialization process completion response on the on-board LEDs for SD card

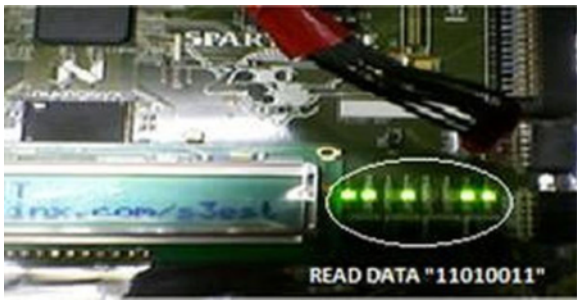


**Fig. 22** Data pattern retrieved in read operation

SDHC card - The results obtained for the 4 GB SanDisk SDHC card are described in this section.

*Initialization* - Figures 23, 24, 25, 26 and 27 show the complete initialization process of the SDHC card. The initialization command and corresponding response pattern of the card have been recorded using DSO. In idle state, the proposed controller gives a logic high signal through the MOSI line and it also receives a logic high signal from the SDHC card.

*Card write operation* - Figures 28 and 29 show the input and output pattern for write operation. The input data have been generated by a PRN generator. Both the single block and the multiblock data transfer operation have been performed. Figure 28 shows the input and output sequence of a single block write operation whereas Fig. 29 shows the multiblock write operation.

*Card read operation* - Figures 30 and 31 show the read operation of the card. The previously written data pattern, generated by PRN generator, has been read to verify the complete operational performance. Figure 30 shows the single block read operation and Fig. 31 shows the multiblock read operation.

SD card - The results obtained for the 16 MB Cannon SD card are described in this section. Figure 32 describes the complete initialization process. The clock divider section has not been introduced in this section. So the process operates in full clock as provided for the initialization of the SD card. To validate the data transfer procedure, the multiblock write operation, shown in Fig. 33, has been performed for the PRN sequences. Later on, the data pattern has been retrieved from the SD card by performing both the single block and multiblock read operation. Figure 34 shows the single block read operation and Fig. 35 describes the multiblock read operation.

MicroSD card - The results obtained for the 2 GB microSD card are described in this section. The operation method and response patterns are nearly the same to the SD card. The microSD card initialization requires 100- to 400-KHz clock frequency. The clock divider module has not been integrated during initialization and block write phases. It has been used only for the read operation to make the validation process realizable. The multiblock read/write operation has been performed only to validate the data transfer operation. Figure 36 describes the complete initialization process. Figure 37 describes the multiblock write process and Fig. 38 indicates the multiblock read operation for the 2 GB microSD card.


**Fig. 21** CRC response pattern for write operation

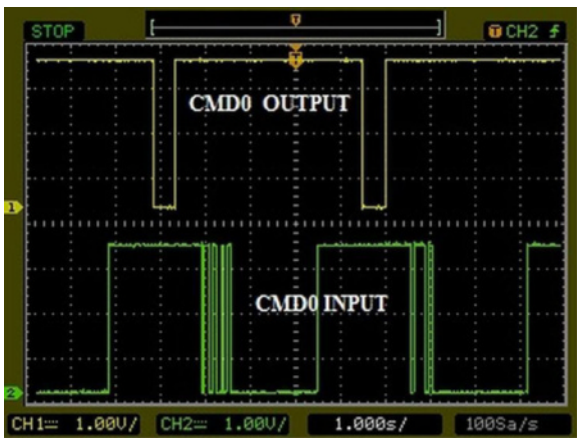


**Fig. 23** CMD0: input sequence and response pattern

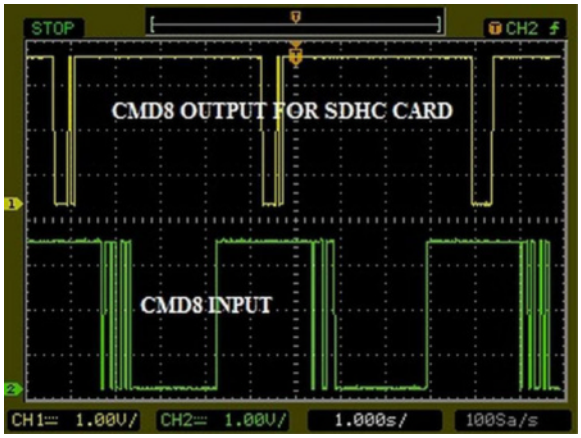


**Fig. 24** CMD8: input sequence and response pattern

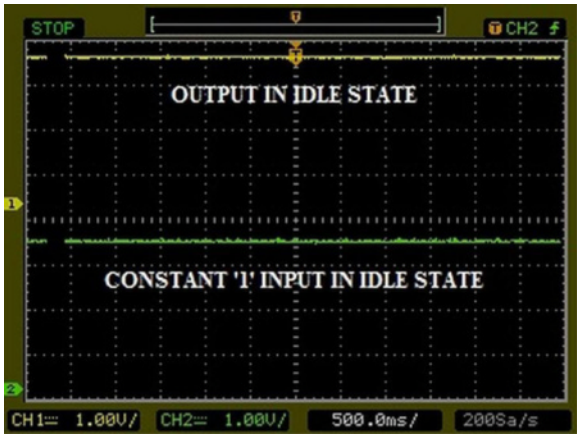


**Fig. 27** Idle mode input and response sequence

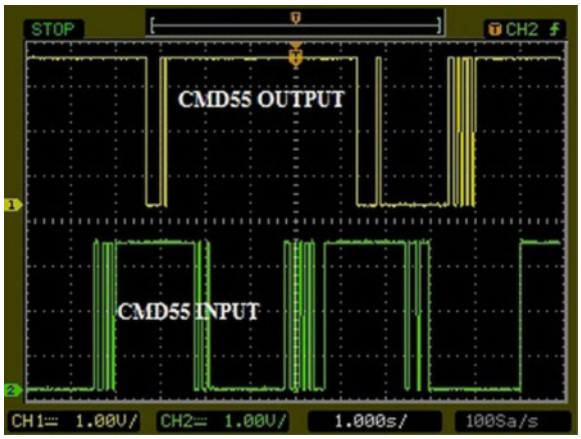


**Fig. 25** CMD55: input sequence and response pattern

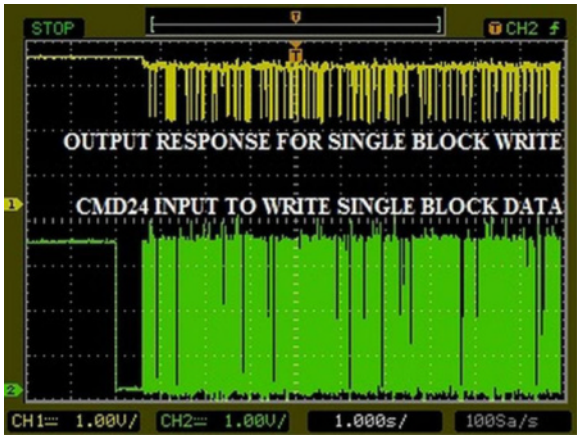


**Fig. 28** Input data with response pattern for single block write operation on DSO

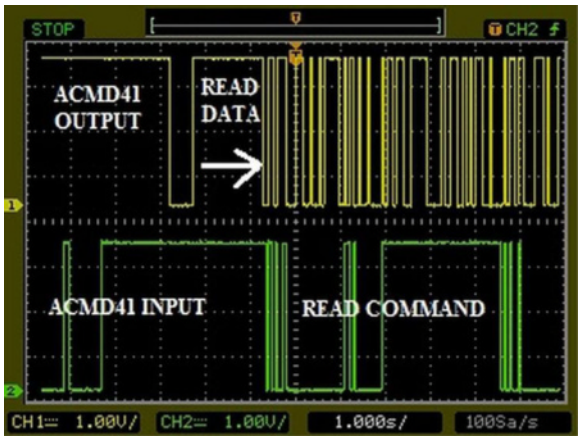


**Fig. 26** ACMD41: input sequence and response pattern with single block read sequence

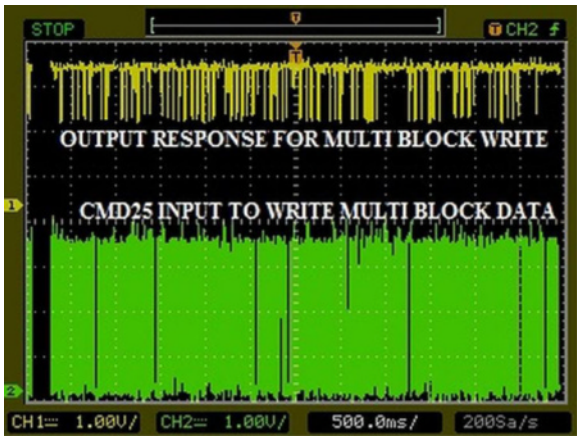


**Fig. 29** Input data with response pattern for multiblock write operation on DSO

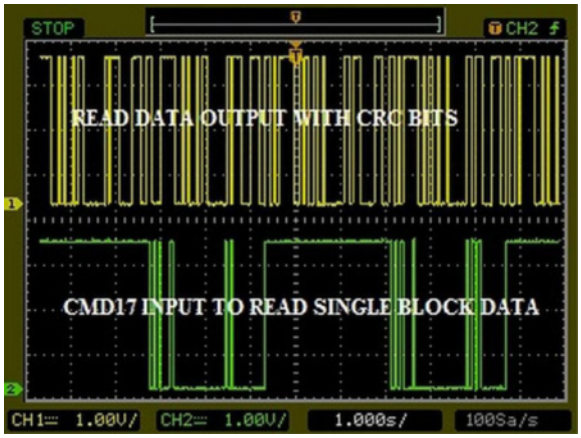


**Fig. 30** Single block read mode input command and data response from SDHC

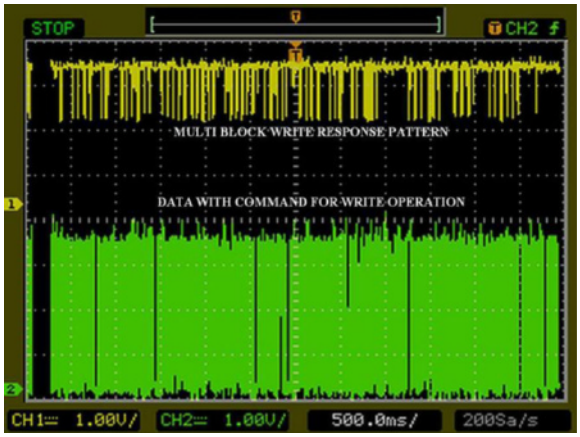


**Fig. 33** Input data with response pattern for multiblock write operation to SD card

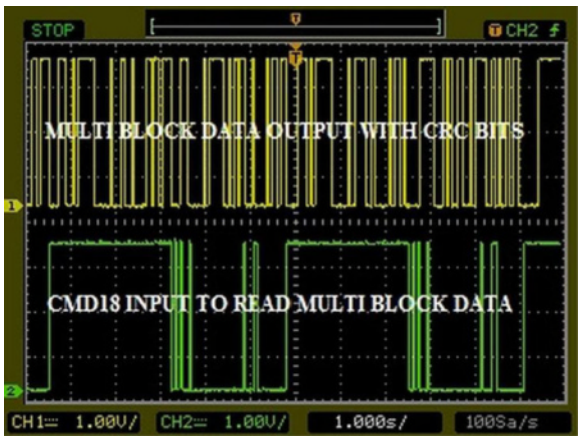


**Fig. 31** Multiblock read mode input command and data response from SDHC

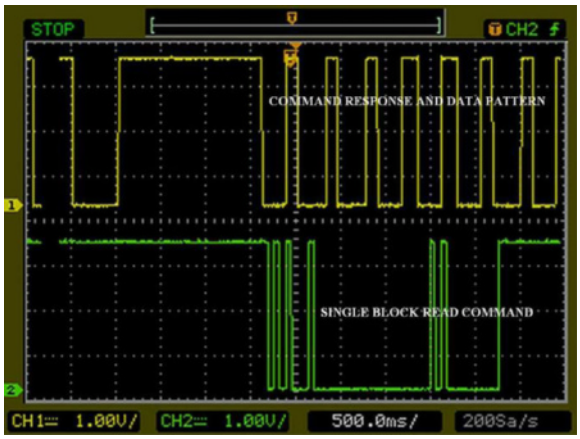


**Fig. 34** Single block read mode input command and data response from SD

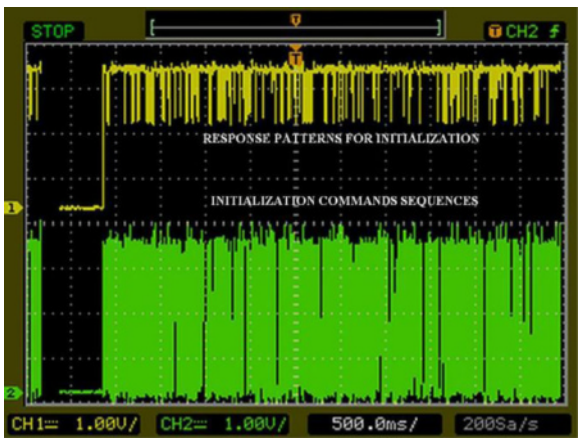


**Fig. 32** Initialization: input sequences and response patterns

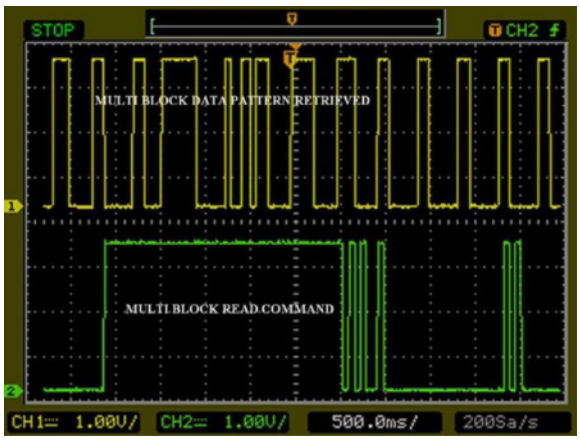


**Fig. 35** Multiblock read mode input command and data response from SD

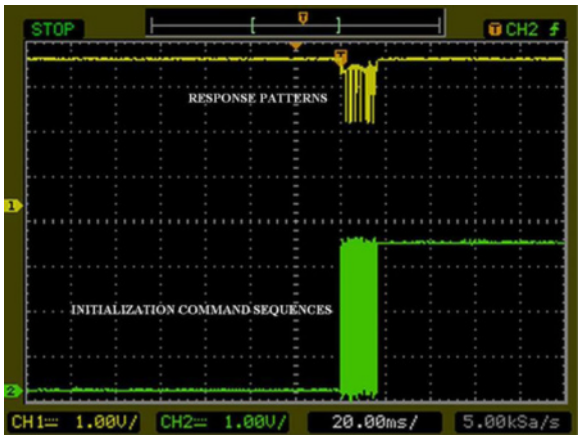


**Fig. 36** Initialization: input sequences and response patterns for microSD card

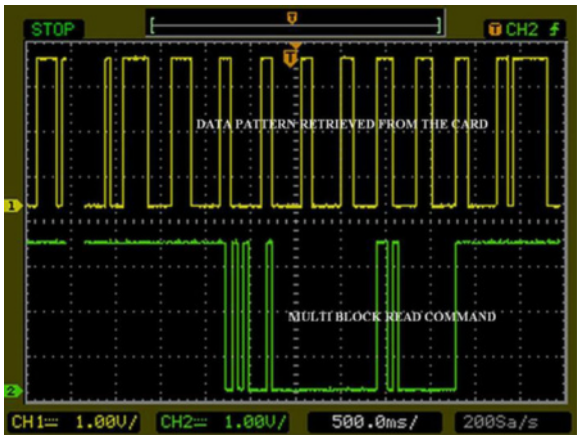


**Fig. 38** Multiblock read mode input command and data response from microSD

The third and the forth case study were to test and verify the efficiency of the proposed controller for real-time audio and video signal storage and retrieval.

*Case III: result - audio signal transmission*: In this study an analog signal is fed to the system in audio range. The received signal has been converted and compressed into 8-bit digital data pattern and fed it to the input data bus of the controller for the verification of multiblock data write procedure in the SDHC card. The data stored in this process is again accessed via multiblock card read procedure of the controller. Both the input audio data and the data accessed from the SDHC card were fed to the DSO. Figure 39 shows the input data pattern and the read operation from the SDHC card. The read and write operations are two different operations which cannot be performed at same time instance. Therefore, initially the analog signal was converted and stored into the SDHC card and during that phase different commands, responses and data patterns were recorded using DSO. Later on, the same data pattern has been retrieved from the SDHC card in read mode and they were also recorded. We got the stored data back along with the CRC bits and on DSO those CRC patterns were present with the original data. For simplicity, the result has been given in the form of bit pattern and the parts of the stored data have been encircled to show the same portion of the retrieved data.

*Case IV: result - video signal transmission*: The video signal has been stored and retrieved in this fourth case study. A video signal of 20 frames/second was fed to the system and subsequently stored into the card. The SanDisk 8 GB class 4 SDHC card has been used to verify the video signal transmission process. The card performs in 4 MB/s writing speed to store and retrieve the video signal. The entire communication was governed under the control of the proposed controller. To ensure the successful data storage and retrieval, later those frames were retrieved from

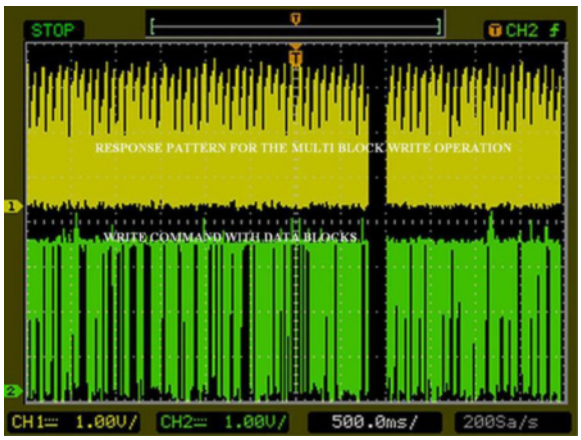


**Fig. 37** Input data with response pattern for multiblock write operation to microSD card

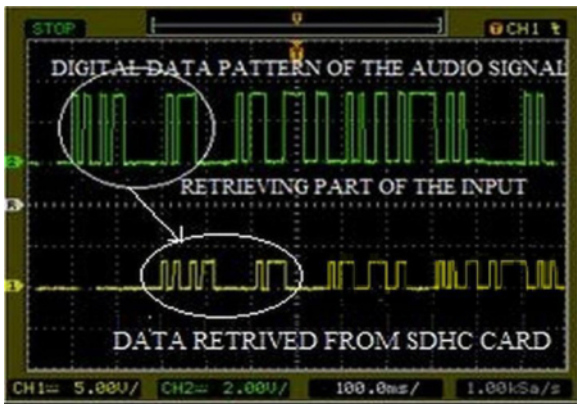


**Fig. 39** Audio signal processing: data stored and retrieval pattern

the card and they were displayed in CRT monitor. The retrieved frames were compared with the original frames which were stored in the card. Figure 40 shows a snapshot of such video data storage and retrieval. The right hand screen in Fig. 40 shows the actual frame that was written in the card and the left hand side monitor shows the frame retrieved from the card. Here the left hand side monitor, the implemented controller along with the card work as a standalone unit.

*Case V: result - dataset from MSR Cambridge Traces*: The datasets were collected from MSR Cambridge traces [24, 25], as explained in "Case studies" section where the description of the datasets is illustrated. In our experimental part, each file of three traces is considered as separate dataset. The datasets were divided into multiple blocks so that we can accomplish multiblock read-write with respect to flash memory. In the performance comparison section, the metrics were computed for multiblock data read write with respect to different I/O traces. Results were tabulated for the datasets collected from the repositories.

*Case VI: result - dataset from UMass Trace Repository*: The datasets were collected from UMass Trace Repository [24, 27], as explained in "Case studies" section. In our experimental part, those datasets have been divided into multiple blocks and the data has been stored in SanDisk 8 GB class 4 SDHC card to accomplish multiblock read-write. In the performance comparison section, the metrics were computed for multiblock data read write with respect to different I/O traces. Results were tabulated for the dataset collected from the repositories.

*Case VII: result - dataset from SNIA Iotta Repository Historical Section*: The datasets were collected from the historical section SNIA Iotta Repository [24, 26], as explained in "Case studies" section. In our experimental part, those datasets have been divided into multiple blocks to accomplish the read write with respect to flash memory. Results were tabulated for the dataset collected from the repositories.

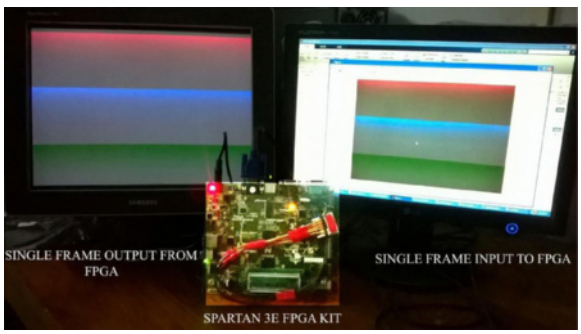


**Fig. 40** Snapshot of video processing

## 6 Performance comparison

This section describes the performance comparison of the controller with reported results. The logic utilization for the proposed controller is only 5% of the total logic present in Spartan 3E FPGA board. This is evident from Table 6.

The proposed controller has been tested for a single block as well as for multiblock (5000 block) data read and write operation for the SD, SDHC, and microSD cards. Table 11 shows a summary of clock cycle elapsed by SDHC card, and Table 12 shows a summary of clock cycle elapsed by the microSD card during initialization, single and multiblock reads, and single and multiblock write operation. In the initialization phase, the proposed system utilizes full bandwidth supported by SD card technology. Initially the SDHC card receives CMD0, CMD8, CMD55, ACMD41, CMD58, and CMD16 commands and the controller receives corresponding responses for each of the commands. (CMD55 and ACMD41) pair of commands may be required to send "$n$" times until the card generates the response pattern x"00". Therefore the total number of clock cycles elapsed in initialization phase by SDHC card is $112(2 + n)$. However in Table 11 we have considered the ideal case of $n = 1$, and the initialization process takes the minimum of 336 clock cycles for SanDisk 8 GB class 4 SDHC card in SPI bus mode. Similarly, SD card initialization process requires $56(3 + 2n)$ clock cycles and microSD card requires $56(1 + 2n)$ clock cycles for initialization. The difference is due to the fact that CMD58 command is not necessary for the initialization phase of SD card and CMD58, CMD16 commands are not required for the microSD card initialization. Only CMD0 and (CMD55+ACMD41) in place of CMD1 initiates the microSD card in SPI mode. In the ideal case with $n = 1$, the initialization process takes the minimum of 280 clock cycles for cannon 16 MB class 2 SD card and 168 clock cycles for microSD card in SPI bus mode. The HCS bit should be high in ACMD41 command for initialization of the SDHC card, whereas for SD card and the microSD card with capacity $\leq$ 2 GB requires HCS bit to be de-asserted at the time of initialization.

Tables 13 and 14 show a comparative study of the speed of response for SD card. Also in Table 13, the initialization time for SDHC card and microSD card has been

**Table 11** Clock cycles achieved for SDHC

| Process | CCTPP |
|---|---|
| Initialization | 336 |
| Single block read | 4176 |
| Single block write | 4184 |
| Multiblock (5000) read | 20600056 |
| Multiblock (5000) write | 20640056 |

**Table 12** Clock cycles achieved for microSD card

| Process | CCTPP |
|---|---|
| Initialization | 168 |
| Single block read | 4176 |
| Single block write | 4184 |
| Multiblock (5000) read | 20600056 |
| Multiblock (5000) write | 20640056 |

reported. Here the comparison is only done with respect to SD card initialization time available in the literature. Also from Table 14, we find the percentage increase in speed of response in read-write phase for SD card with the proposed controller.

Table 15 shows the software speed-up for the proposed controller. The SanDisk 8 GB class 4 SDHC card, Cannon 16 MB class 2 SD card, and the class 4 microSD card have been used for multiblock data transfer using the proposed controller, designed in VHDL. The similar data transfer setup has been implemented based on two modern operating systems (Windows 7 and Windows XP (Service Pack 3)) to validate the speed up of the proposed controller with respect to existing software based approaches. The modern operating systems are mostly written in JAVA or a high-level languages which ultimately makes the system's in-build controller slower than the proposed controller, optimized using VHDL. The computer used for this purpose contains Intel second-generation dual core processor, 2 GB DDR3 RAM as the basic specifications.

Figure 41 shows the comparison chart of the initialization process performed in the SD card. The work has been compared with the closest work reported in Elkeelany et al.'s paper [1]. The chart in Fig. 41 clearly shows the reduction in the initialization time. The proposed process introduces 65.56% improvement in the initialization time. Figure 42 describes the comparison in clock cycle required for single block data transfer (single block read/write) and Fig. 43 shows the corresponding improvements in percentage. The proposed architecture introduces 65% improvement of clock cycle for single block read and 44% improvement of clock cycle for single block write operation. On the other hand, Fig. 44 explains the comparison in multiblock data transfer and Fig. 45 shows corresponding improvement in percentage. Here, the proposed system introduces 21.59% improvement in required clock cycle for multiblock read and 2.87% improvement

**Table 13** Speed comparison in initialization phase

| Process | TTPP | | | |
|---|---|---|---|---|
| | SD card achieved | O. Elkeelany et al. [1] | SDHC achieved | MicroSD achieved |
| Initialization | 22 ms | 63.88 ms | 27 ms | 20 ms |

**Table 14** Speed comparison in read/write phase

| Process | CCTPP | | |
|---|---|---|---|
| | SD card achieved | Elkeelany et al. [1] | % Reduction |
| Single Block Read | 4176 | 12025 | 65 |
| Multiblock Read (5000 Block) | 20600056 | 26275000 | 21.59 |
| Single Block Write | 4184 | 7472 | 44 |
| Multiblock Write (5000 Block) | 20640056 | 21250000 | 2.87 |

in required clock cycle for multiblock write. The speed of SD card data access in terms of clock cycles for single block read/write is increasing to 65 and 44%, respectively, and for multiblock read write it is increasing to 21.59 and 2.87%, respectively. The comparison chart of the software speed-up of VHDL with respect to the high-level language is shown in Figs. 46, 47 and 48 for SDHC, SD and microSD card, respectively.

The performance of the proposed controller has also been tested for three different commonly used repositoties. These three repositories are MSR Cambridge Traces [25], UMass Trace Repository [27], and SNIA Iotta Repository [26]. Tables 8, 9 and 10 represent the characteristics of MSR Cambridge Traces [25], UMass Trace Repository [27], and SNIA Iotta Repository [26], respectively. The files in each repositories, the size of the file, and the attributes of the files are given in the tables. Tables 16, 17 and 18 show the performance of the controller for both read and write process with respect to repository datasets and workload, average CCTPP(read), average CCTPP (write), and IOPS metrics. All the datasets of MSR cambridge and UMass repositories have been used and only Harvard SOS subtraces of NFS trace of SNIA Iotta repository has been used for experimental section.

**Table 15** Software speedup in multiblock data transfer phase

| Card used | Process | TTPP | | |
|---|---|---|---|---|
| | | VHDL achieved | Windows 7 | Windows XP |
| SDHC | Multiblock read (5000 blocks) | 60 μs | 2.01 s | 2.70 s |
| | Multiblock write (5000 blocks) | 100 μs | 10.26 s | 11.10 s |
| SD | Multiblock read (5000 blocks) | 57 μs | 1.98 s | 2.46 s |
| | Multiblock write (5000 blocks) | 93 μs | 9.89 s | 10.71 s |
| MicroSD | Multiblock read (5000 blocks) | 48 μs | 1.53 s | 1.99 s |
| | Multiblock write (5000 blocks) | 75 μs | 9.12 s | 10.02 s |

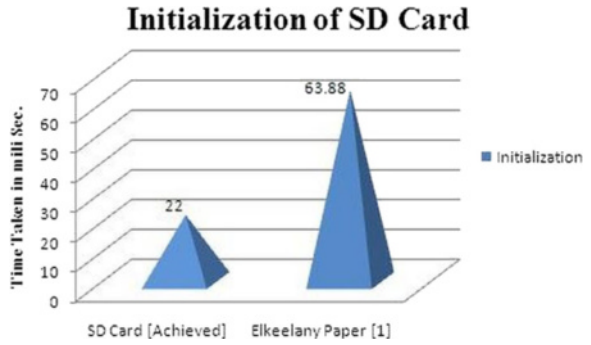


**Fig. 41** Comparison chart for the initialization process of the SD card

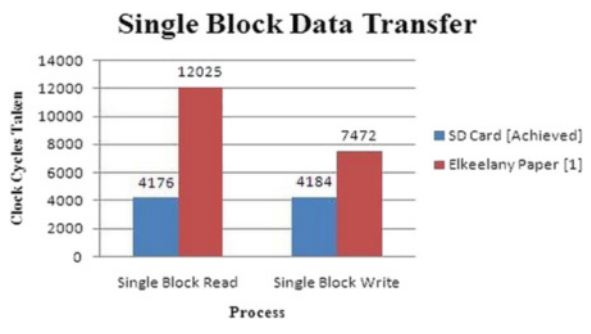


**Fig. 42** Comparison chart for the single block data transfer of SD card

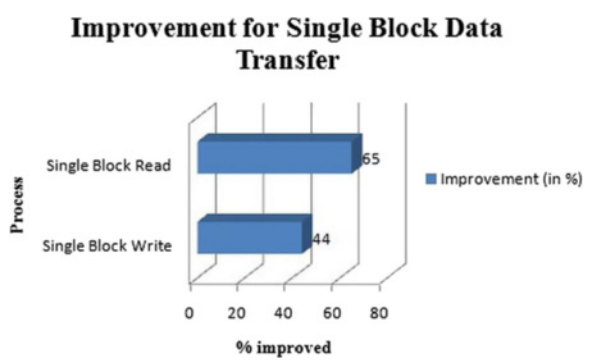


**Fig. 43** Improvement chart for single block data transfer of SD card

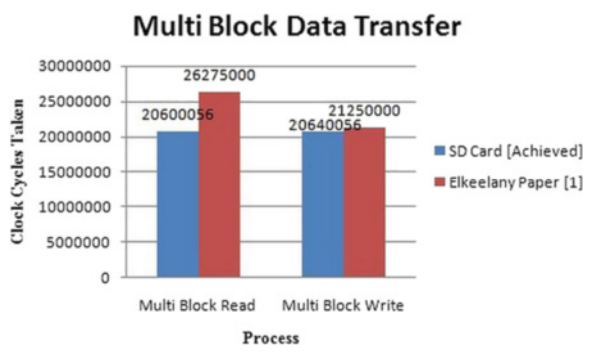


**Fig. 44** Comparison chart for multiblock data transfer of SD card

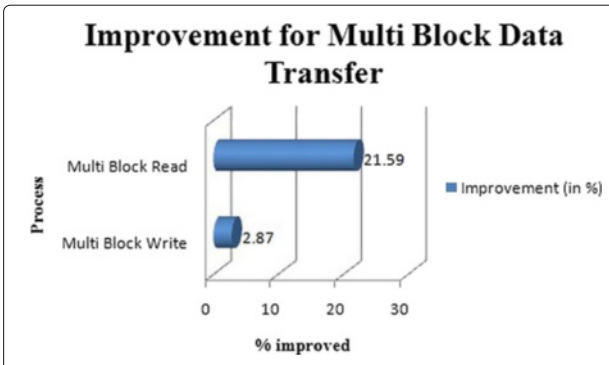


**Fig. 45** Improvement chart for multiblock data transfer of SD card

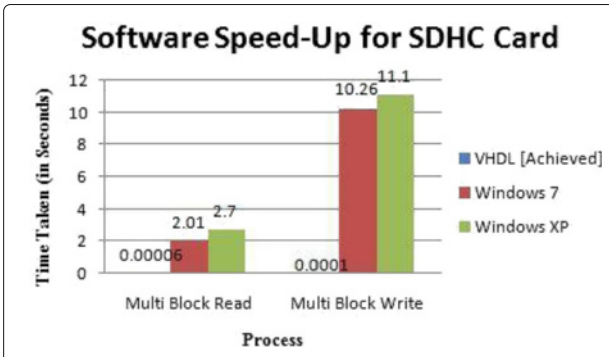


**Fig. 46** Speed up comparison chart of SDHC card

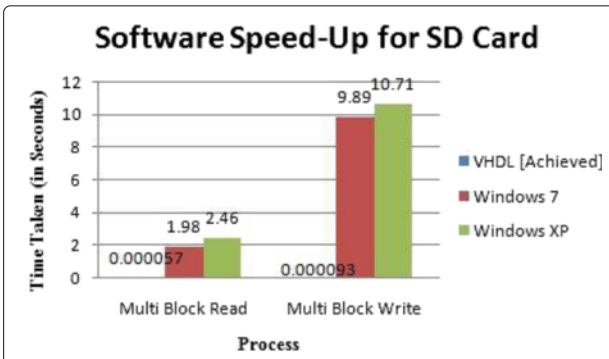


**Fig. 47** Speed up comparison chart of SD card

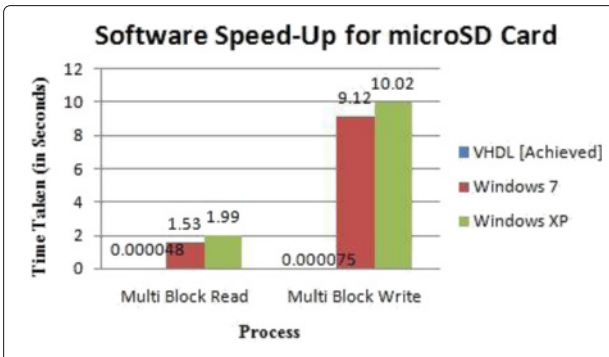


**Fig. 48** Speed up comparison chart of microSD card

**Table 16** Performance table for MSR Cambridge Repository [25]

| Repository name | File name abbreviation | Workload in blocks of data | Average response CCTPP (Read) | Average response CCTPP (Write) | IOPS (operations/second) |
|---|---|---|---|---|---|
| MSR | M11 | 144,741 | 87,980 | 59,568 | 957 |
| Cambridge | M12 | 453,996 | 43,481 | 228,544 | 957 |
| 1 | M13 | 566,045 | 33,408 | 218,313 | 958 |
| | M14 | 929,401 | 79,007 | 118,519 | 958 |
| | M15 | 103,129 | 33,438 | 117,673 | 957 |
| | M16 | 233,319 | 108,512 | 51,613 | 958 |
| | M17 | 386,921 | 153,010 | 199,474 | 956 |
| | M18 | 59,242 | 149,141 | 20,467 | 957 |
| | M19 | 246,939 | 101,917 | 105,340 | 957 |
| | M110 | 123,427 | 36,811 | 52,927 | 957 |
| | M111 | 85,412 | 33,663 | 48,690 | 957 |
| | M112 | 127,229 | 165,980 | 50,863 | 957 |
| MSR | M21 | 189,660 | 154,168 | 72,762 | 957 |
| Cambridge | M22 | 229,310 | 162,438 | 60,576 | 957 |
| 2 | M23 | 283,342 | 42,193 | 133,960 | 958 |
| | M24 | 196,382 | 172,309 | 81,244 | 957 |
| | M25 | 277,067 | 106,205 | 115,738 | 960 |
| | M26 | 83,360 | 33,408 | 35,466 | 958 |
| | M27 | 136,364 | 33,419 | 122,567 | 958 |
| | M28 | 570,197 | 317,922 | 107,067 | 956 |
| | M29 | 1,157,651 | 44,544 | 53,952 | 956 |
| | M210 | 117,600 | 34,965 | 51,319 | 958 |
| | M211 | 164,072 | 33,709 | 107,546 | 957 |
| | M212 | 148,760 | 33,408 | 95,238 | 958 |
| | M213 | 192,269 | 88,610 | 72,740 | 958 |
| | M214 | 154,498 | 48,024 | 64,521 | 957 |
| | M215 | 272,685 | 125,598 | 66,648 | 957 |
| | M216 | 145,708 | 119,592 | 59,161 | 954 |
| | M217 | 190,880 | 81,501 | 78,696 | 957 |
| | M218 | 1,227,759 | 84,665 | 36,812 | 957 |
| | M219 | 811,852 | 889,535 | 166,042 | 957 |
| | M220 | 215,659 | 112,999 | 78,539 | 957 |
| | M221 | 10,816 | 35,696 | 42,812 | 957 |
| | M222 | 162,960 | 32,609 | 68,052 | 957 |
| | M223 | 10,596 | 33,527 | 44,698 | 957 |

**Table 17** Performance table for UMass Repository [27]

| Repository name | File name | Workload in bytes of data | Average response CCTPP (Read) | Average response CCTPP (Write) | IOPS (operations/second) |
|---|---|---|---|---|---|
| UMass | Financial1 | 107,967 | 50,889 | 41,093 | 957 |
| | Financial2 | 57,254 | 20,371 | 41,542 | 957 |
| | WebSearch1 | 306,500 | 128,006 | 66,816 | 957 |
| | WebSearch2 | 303,744 | 126,855 | 66,816 | 956 |
| | WebSearch3 | 289,776 | 121,032 | 66,816 | 957 |

**Table 18** Performance table for SNIA Historical Section [26]

| Trace name | Sub-trace | Workload in number | Response CCTPP (Read) | Response CCTPP (Write) | IOPS (operations/second) |
|---|---|---|---|---|---|
| NFS | Harvard | 12357 | 38093 | 16343 | 956 |
| Trace | SOS Traces | | | | |
| | (Deasna Week1) | | | | |

We have also tested the efficiency, performance and handling capacity of this proposed controller in different FTL settings where volume of data is varied from 4 MB, 8 MB, 10 MB, 512 MB to 1 GB. Table 19 shows the performance in terms of CCTPP for the above mentioned volume of datasets.

## 7 Conclusions

An on-chip design and implementation of a controller has been proposed for SDHC and similar family of cards. The design has also been implemented for the microSD card. In addition to that, the same controller can be used for data communication with MMC also. The FSM-based architecture design, its operation, FPGA-based implementation, control flow and execution, synthesis results, and all other implementation related issues were discussed in details. Results were tabulated for different problems specified above, and it is seen that the efficiency is increasing in the proposed design. The speed of SD card data access in terms of clock cycles for single block read/write is increasing to 65 and 44% respectively, and for multiblock read write it is increasing to 21.59 and 2.87% respectively compared to the closest reported work [1].

Future work will involve the extension of the proposed controller in more wide sense. This paper presently focuses on SPI mode-based data communication for SD, SDHC, and microSD cards. However, this approach can be extended to other mode of data transfer supported by similar family of cards. Regarding increase of speed in data transfer between external system and the storage device, further modification can be incorporated by changing the proposed architecture as well as the design can be implemented in other high-end target platform with a very minor modification in configuration procedure.

**Table 19** Performance in different FTL settings

| Data volume | Process | CCTPP |
|---|---|---|
| 4 MB | Write block | 33,472 |
| | Read block | 33,408 |
| 8 MB | Write block | 66,944 |
| | Read block | 66,816 |
| 10 MB | Write block | 83,680 |
| | Read block | 83,520 |
| 512 MB | Write block | 4,284,416 |
| | Read block | 4,276,224 |
| 1 GB | Write block | 8,368,000 |
| | Read block | 8,352,000 |

**Authors' contributions**
The work has been carried out as the M.Tech. final year project of the first author, SB, under the supervision of the second author, SM. The selection and setup of the project had been carried out by both the authors together. The structuration and coding part was carried out by SB and the testing and debugging part was done by both the authors. This manuscript had been prepared and checked by both of the authors together. All authors read and approved the final manuscript.

**Competing interests**
The authors declare that they have no competing interests.

Received: 28 May 2016 Accepted: 31 October 2016
Published online: 28 November 2016

**References**

1. O Elkeelany, VS Todakar, Data archival to SD card via hardware description language. IEEE Embed. Syst. Lett. **3**(4), 105–108 (2011)
2. SD Card Association and Technical Committee and Specifications, SD, et al., *Part 1, physical layer, simplified specification, Version 2.00*. (SD Card Association, San Ramon, 2006). https://www.sdcard.org/downloads/pls/simplified_specs/archive/part1_200.pdf
3. Part, SD Specifications, Physical Layer Simplified Specification Version 2.00. **31**, 1 (2010). http://www.sdcard.org/developers/tech/sdcard/pls/
4. A Lakshman, P Malik, Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Systems Review. **44**(2), 35–40 (2010)
5. C Border, in *ACM SIGCSE Bulletin*. The development and deployment of a multi-user, remote access virtualization system for networking, security, and system administration classes, vol. 39 (ACM, 2007), pp. 576–580
6. H Müller, N Michoux, D Bandon, A Geissbuhler, A review of content-based image retrieval systems in medical applications–clinical benefits and future directions. Int. J. Med. Inform. **73**(1), 1–23 (2004)
7. S-H Liao, Expert system methodologies and applications—a decade review from 1995 to 2004. Expert Syst. Appl. **28**(1), 93–103 (2005)
8. J Tai, D Liu, Z Yang, X Zhu, J Lo, N Mi, Improving flash resource utilization at minimal management cost in virtualized flash based storage systems. IEEE Trans. Cloud Comput. **99**, 1–14 (2015)
9. C Wang, Q Wang, K Ren, N Cao, W Lou, Toward secure and dependable storage services in cloud computing. IEEE Trans. Serv. Comput. **5**(2), 220–232 (2012)
10. H Nie, Q Xie, Y Zhang, M Li, Q Liu, H Liu, J Zhang, B Li, in *Intelligent System Design and Engineering Application (ISDEA), 2012 Second International Conference On*. Research on solid state storage based remote sensing data storage (IEEE, 2012), pp. 1294–1297
11. H Chen, TN Cong, W Yang, C Tan, Y Li, Y Ding, Progress in electrical energy storage system: a critical review. Prog. Nat. Sci. **19**(3), 291–312 (2009)
12. S Cohn, M Ross, *Methods, systems, and devices for wireless delivery, storage, and playback of multimedia content on mobile devices*. (Google Patents, 2001). US Patent App. 10/040,617
13. H Qi, A Gani, in *Digital Information and Communication Technology and It's Applications (DICTAP), 2012 Second International Conference On*. Research on mobile cloud computing: Review, trend and perspectives (IEEE, 2012), pp. 195–202
14. Y Yang, Y Yang, L Niu, in *2012 Second International Conference on Instrumentation, Measurement, Computer, Communication and Control*.

Design of sdhc card video player based on sopc (IEEE Computer Society, 2012), pp. 900–904. doi:10.1109/IMCCC.2012.216
15. M Abdallah, O Elkeelany, in *Computing, Engineering and Information, 2009. ICC'09. International Conference On*. Simultaneous multi-channel data acquisition and storing system (IEEE, 2009), pp. 233–236. doi:10.1109/ICC.2009.17
16. C-S Lin, K-Y Chen, Y-H Wang, L-R Dung, in *2006 13th IEEE International Conference on Electronics, Circuits and Systems*. A nand flash memory controller for sd/mmc flash memory card (IEEE, 2006), pp. 1284–1287. doi:10.1109/TMAG.2006.888520
17. O Elkeelany, G Vince, in *2007 Thirty-Ninth Southeastern Symposium on System Theory*. Portable analog data capture using custom processing (IEEE, 2007), pp. 120–123
18. C Li, Q Wang, L Wang, in *Computer and Information Technology Workshops, 2008. CIT Workshops 2008. IEEE 8th International Conference On*. A high efficient flash storage system for two-way cable modem (IEEE, 2008), pp. 551–556. doi:10.1109/CIT.2008.Workshops.30
19. C-Y Lu, H Kuan, Nonvolatile semiconductor memory revolutionizing information storage. IEEE Nanotechnol. Mag. **3**(4), 4–9 (2009). doi:10.1109/MNANO.2009.934861
20. S Xilinx, 3E starter kit board user guide. UG230 (v1. 0) March. **9** (2006). http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf
21. SanDisk, Secure Digital Card Product Manual. **1.9**(80-13-00169) (2003)
22. Instruments, Texas, *Msp430x1xx family user's guide*. (SLAU049B, 2006)
23. Y Deng, J Zhou, Architectures and optimization methods of flash memory based storage systems. J. Syst. Archit. **57**(2), 214–227 (2011)
24. D Narayanan, A Donnelly, A Rowstron, Write off- loading: practical power management for enterprise storage. ACM Trans. Storage. **4**(3), 10–11023 (2008)
25. Storage Networking Industry Association and others, MSR Cambridge Traces (2010). http://iotta.snia.org/traces/388
26. Storage Networking Industry Association, *et al*, SNIA Iotta Repository. Microsoft Enterprise Traces, Colorado Springs, Colorado (iotta. snia. org/traces/130) (2011). http://iotta.snia.org/historical_section
27. Application, OLTP, I/O and search engine I/O. umass trace repository (2007). http://traces.cs.umass.edu/index.php/Storage/Storage
28. S Chen, What types of ECC should be used on flash memory. Application Note for SPANSION (2007). http://www.spansion.com/support/application%20notes/types_of_ecc_used_on_flash_an.pdf
29. J No, Nand flash memory-based hybrid file system for high I/O performance. J. Parallel Distrib. Comput. **72**(12), 1680–1695 (2012)
30. R Wang, Z Mi, H Yu, W Yuan, The design of image processing system based on SOPC and ov7670. Procedia Eng. **24**, 237–241 (2011)
31. M Fabiano, M Indaco, S Di Carlo, P Prinetto, Design and optimization of adaptable BCH codecs for nand flash memories. Microprocess. Microsyst. **37**(4), 407–419 (2013)
32. M Baklouti, P Marquet, J Dekeyser, M Abid, FPGA-based many-core system-on-chip design. Microprocessors and Microsystems. **39**(4), 302–312 (2015)
33. F Thomas, M Nayak, S Udupa, J Kishore, V Agrawal, A hardware/software codesign for improved data acquisition in a processor based embedded system. Microprocess. Microsyst. **24**(3), 129–134 (2000)
34. F Chen, DA Koufaty, X Zhang, in *Proceedings of the International Conference on Supercomputing*. Hystor: Making the best use of solid state drives in high performance storage systems (ACM, Tucson, Arizona, 2011), pp. 22–32
35. J Guerra, H Pucha, JS Glider, W Belluomini, R Rangaswami, in *FAST*. Cost effective storage using extent based dynamic tiering, vol. 11, (2011), pp. 20–20
36. Technical Committee SD Card Association, et al., Speed Class Greater Performance Choices, Online available and accessed. Onlineathttps://www.sdcard.org/developers/overview/speed_class/
37. Editor - Metering, Minsen - your ideal supplier of wireless water/gas/electricity meters (2013). https://www.metering.com/minsen-your-ideal-supplier-of-wireless-water-gaselectricity-meters/