Sd Card Interface Using FPGA for Multimedia Applications

Dr. Srigitha S Nath
Department of Electronics and Communication Engineering
Saveetha Engineering College
Chennai, India
hod.ece@saveetha.ac.in

Dr. Navaneethan S
Department of Electronics and Communication Engineering
Saveetha Engineering College
Chennai, India
navaneethans@saveetha.ac.in

Sakthekannan M S
Department of Electronics and Communication Engineering
Saveetha Engineering College
Chennai, India
sakthekanna@gmail.com@gmail.com

Udaya Krishnan M
Department of Electronics and Communication Engineering
Saveetha Engineering College
Chennai, India
udayakrishnan.ece.sec@gmail.com

Yogavignes B M
Department of Electronics and Communication Engineering
Saveetha Engineering College
Chennai, India
yogavignesbm@gmail.com

***Abstract*— Field Programmable Gate Array (FPGA) is an application that is based on chips that can process audio, video, and image. For FPGA systems, onboard memory may be used; however, it cannot be readily expanded by putting in extra cards. Since microcontrollers can read and write Secure Digital (SD) cards, they are more useful. The primary goal is to develop a low-cost, non-volatile, flash-memory, detachable, portable, and user-friendly storage solution for Field Programmable Gate Array (FPGA) that can store vast quantities of data. Verilog enables complete access to the SD card's contents, eliminating the need for a microcontroller or general-purpose CPU on-chip. In this work, Spartan 6 and Artix-7 FPGA are utilized. For this, the proposed study utilizes a Strontium 4GB micro SDHC (class 6) card. The SD card stores files in the FAT32 format. The objective is to extract a BMP image file from an SD card. The proposed design achieved the hardware resources are 916 look-up tables (LUT) and 464 flip-flops.**

Keywords—Field Programmable Gate Array (FPGA), Artix-7, Verilog HDL (Hardware Description Language), SD (Secure Digital) card, 4-bit SD mode, SPI (Serial Peripheral Interface) protocol.

I. INTRODUCTION

SD cards, an abbreviation for "secure digital," are nonvolatile memory cards used solely by multimedia applications. A semiconductor-based memory card of the newest generation is a portable or stationary electronic device designed only for data storage. SD cards have various advantages, including their mobility, high storage capacity, high transfer rates, low cost, low power consumption, high degree of adaptation, and high level of security. SD cards are a typical portable storage option in digital systems such as microprocessors, microcontrollers, Digital Signal Processor (DSP) or FPGA chips, digital cameras, computers, mobile phones, and embedded systems. For connecting with SD cards, the 1-bit SD mode, 4-bit SD mode, and Serial Peripheral Interface (SPI) mode are all supported. SD cards employ flash memory chips from the most recent generation of semiconductor storage technologies. An SD card can store a large amount of data, transport files swiftly, be utilized in a range of mobile devices, and offer superior security [9],[10],[11].

II. LITERATURE REVIEW

According to Dumitrel Catalin Costach [1] an FPGA controller reads and writes to an SD card using the SPI protocol. SPI transfer mode is necessary for the growth of basic storage systems. If we wish to utilize more SD cards, we must ensure that they all use the same clock, data, and chip select signals. This kind of memory access demands the use of several individual cards, yet it enables the storage of vast amounts of data.

According to Gul Munir Ujjan [4] introduced a fundamental hardware architecture based on FPGA and an integrated NIOS-II processor for processing SD cards in 4-bit SD mode. Using the Eclipse platform, software for the NIOS II processor is created. FAT-32 read-and-write instructions for a single block have been written and tested. These instructions can be executed in 1-bit SD mode. The read performance of SD cards with four bits is approximately 67% greater than that of SD cards with one bit. SD 4-bit mode is thus 80% faster than SD 1-bit mode during write operations. Improving overall performance may be a high concern, thus it is essential to provide a mechanism that does multi-block writes and reads and calculates 16-bit CRC write instructions quickly. This inevitably necessitates more expensive hardware resources. The proposed firmware would be beneficial for recording video in real-time, among other applications.

According to Pallavi Polsani [6] external A/D converters, D/A converters, and EEPROMs, serial synchronous communication between master and slave devices are used to connect the microcontroller to the other devices. There are two primary classifications of protocols. Priorities: 2) Inter-I2C Each protocol has been optimized for inter-IC communication on a board. SPI has become the predominant standard for delivering data streams at low to medium rates within and between processors. SPI (Serial Peripheral Interface) is a master-slave system that transmits data in bits and is highly configurable. For the verification and implementation of the SPI design, System Verilog was utilized. The functionality and code coverage of the system is validated. The whole RTL is created in Verilog for synthesis, whilst the System Verilog-designed and Spartan 3E-implemented verification architecture ensures quality.

According to J. Y. Qiang [5] it has been demonstrated that SPI bus is a sort of synchronous, full-duplex serial interface data bus line with a simple protocol and a high data transmission rate. FPGAs facilitate rapid device design and testing, making them a popular choice for parallel processing.

Here, we shall discuss the creation and operation of SPI. The communication bus analyses and applies the operation sequence and four modes. An FPGA interface capability for an SPI bus is supplied by a module circuit. The SPI is designed using the hardware description language Verilog, while its waveforms are modeled using the vivado simulator. Simulation of waveform analysis demonstrates the feasibility of the method.

According to Omar Elkeelany [3] discovered it utilizing a programmable Field-Programmable Gate Array (FPGA) data extractor device that enables bidirectional SD card hardware architecture. SD card can read and write 5000 blocks per second, or 1.051 seconds, in total. In the previous attempt, 60 seconds were spent on the same block size. The variances stem from the researchers' selection for unique software-based techniques and finite-state machine designs. This proves the technological possibility of data transfer rates as high as 25 Mb/s. This letter offers a system for real-time SD card storage of data from SG actions at remote sites. Future work will address scaling issues, such as the impact of bigger SD cards and SD cards from various manufacturers.

According to Zinlin [2], it permits the rapid storing of essential data on an SD card. It then designs and develops an SD card reader system architecture based on FPGA to suit the requirements of SD memory card readers. Before being placed, the gadget underwent testing. The device's ability to read and write data to SD cards demonstrates that it satisfies the FPGA device requirement for outdoor garage devices. This needed the creation of an SD card reader that could be implemented on any number of FPGA-based devices.

III. SD PRINCIPLE

A. Mode of Transmission

SD cards can transmit in one of three ways: the serial peripheral interface (SPI), which uses a different parallel input and output from 1-bit SD cards, as well as 1-bit SD cards' one-time transfer type and different command and data channels, and 4-bit SD modes, which use a wider parallel transmission bus and have additional pins and some reset pins. SD card mode provides faster and more reliable access. In 1-bit SD mode, the maximum transfer rate is 25 Mbps, whereas in 4-bit SD mode, the maximum transfer rate is 100 Mbps. Even if the data transfer rate is higher in 4-bit SD transfer mode, the system's complicated structure and timing make it challenging to build.

B. Bus Protocol of SD

The idea behind SD communication is straightforward; it operates in a master-slave fashion. A master controller and many slave controllers connected through 3-6s lines are usual in such a setup. They are labeled as "CMD," "CLK," and "DAT0-3" (data lines).

Table 1. Pin functions of SD mode.

Name	Functions
CLK	microcontroller uses this pin to send clock signal to SD card.
CMD	bidirectional pin for information and command transmission between microcontroller and SD card.
DAT0-3	Four bidirectional pins are used for bulk data transfer data between microcontroller and SD card.

C. System File of SD

On the SD card, a FAT16 file is divided up into four portions. File Data Table (FDT), File Allocation Table (FAT), and Partition Boot Record (PBR) are the three tables that make up the File System (File Directory Table). The BPB (BIOS Parameter Record Blocks), the hard drive flag record book, the partition boot record code section, and the end Flag 55AA are the standard components of partitioned boot files.

D. Command Format for SD

SD internal memory requires a total of six bytes to execute an instruction. This consists of the one-byte command code, four bytes of command inputs, and the checksum bits.

When transferring SD card instructions, the host system must always give 4 bytes of argument data, regardless of whether any parameters are included in the command. In this case, the SD card will just ignore the supplied value.

Below Table 2. describes the commands of SD card and the functions of each command of the SD card.

Table 2. Commands of SD card for SPI mode

Commands	Functions
CMD 0	Software gets reset.
CMD 1	Initialization starts.
CMD 3	Request card send back the RCA address.
CMD 7	Card entering the state.
CMD 9	Read CSD registers.
CMD 10	Read CID registers.
CMD 12	Stop reading data
CMD 16	Change of size in read/write.
CMD 17	Read command for single command.
CMD 18	Read command for multiple commands.
CMD 23	Amount of block sent.
CMD 24	Write in for single block.
CMD 25	Write in for multiple blocks.
CMD 32	Start erase block.
CMD 38	Erase the command.
CMD 55	ACMD <n> leading command.
CMD 58	Read OCR.

IV. SPI PROTOCOL

The Serial Peripheral Interface (SPI) is a protocol utilized by many electrical devices. A card reader for SD or RFID cards and a wireless 2.4 GHz transceiver are examples of peripherals that utilize SPI to connect with a

microcontroller. Sending and receiving any number of bits is possible so long as the flow is continuous. I2C and UART both transmit data in discrete packets with a defined number of bits. Start and stop criteria, which indicate the beginning and end of each packet, distort data transmission [7].

Master-slave is the relationship between SPI devices. The device in control is known as the master, while its subordinates (often sensors, displays, or RAM) are referred to as slaves. A straightforward SPI network consists of a single master and a single slave however a single master may supervise several slaves. Fig. 1 depicts the connection between the master and the slave.

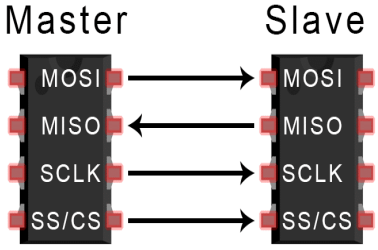

Fig. 1. MOSI

A. Pin Functions of SPI

- Master Output/Slave Input (MOSI) - the master transmits data to the slave.
- Master Input/ Slave Output (MISO) – slave delivers data to master,
- Clock (SCLK) – clock signal line.
- Slave Select/Chip Select (SS/CS) – allows the master to select to which slave to transfer data.

B. MISO and MOSI

The master sends individual data bits to the slave over the MOSI connection. The master transmits data to the slave using the MOSI pin. The norm for data transmission between the master and the slave is most significant bit first. The slave sends the least significant piece of data back to the master.

C. Steps

The master generates the frequency of the clock. When the master pushes the SS/CS pins to a reduced voltage state, the slave is activated. The master transmits information bit by bit to the slave over the MOSI line. The slave inspected each received bit. When a response is required, the slave sends the data back to the master bit-by-bit over her MISO line. The master performs an analysis on the received bits.

D. Advantages

Since there is no beginning or end, information may be sent without interruption. The I2C bus does not require a sophisticated method of addressing that is slave-specific. The rate of data transfer is more rapid than I2C. By separating the MISO and MOSI channels, data transmission and reception may be performed concurrently.

V. PROPOSED WORK

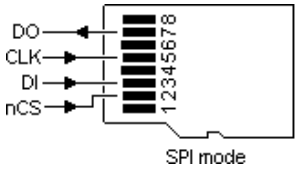

Fig. 2. SPI mode pin diagram

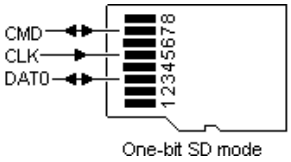

Fig. 3. One-bit SD mode pin diagram

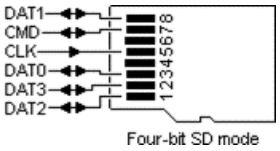

Fig. 4. Four-bit SD mode pin diagram

Among these three modes from Fig 2, Fig 3, Fig 4 (SPI mode, one-bit SD mode and four-bit SD mode), the latter two are significantly slower than the former. The higher bit rate of four-bit SD mode allows for rapid and efficient command execution, including read/write operations.

Table 3. Comparison of one-bit SD mode, four-bit SD mode, SPI mode.

	One - bit SD mode	**Four - bit SD mode**	**SPI mode**
Input Signal	1 CLK PIN, 1 DATA PIN	1 CLK PIN, 1 CMD PIN, 4 DATA PINS	MOSI, MISO, CS, CLK
Minimum Frequency	0 MHz	0 MHz	1 kHz
Maximum Frequency	25 MHz	25 MHz	75 MHz
Bit Rate	25 Mbps	100 Mbps	25 Mbps

A. Control Structure of FPGA

Fig. 5 illustrates the recommended FPGA block architecture for 4-bit SD mode. Regarding the data pins, 4-bit SD mode requires only one clock and four data ones. The Clock Control module's principal purpose is information sharing. All functionality of 4-bit SD mode may be assured with the Command Control package. Due to the SD card's limited 512-byte memory location, two FIFO buffers are designed to temporarily store data during transmission and reception[9][12][13].

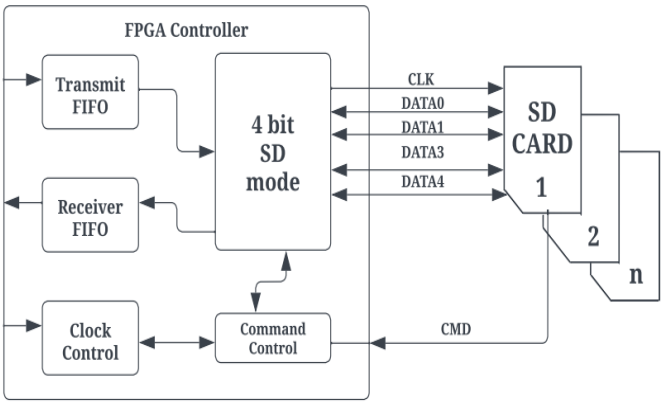

Fig. 5. FPGA block diagram for 4-bit SD mode.

B. SD Card Initialization

To start an SD card increased actions are required. A switch on the rear of every SD card slot indicates whenever a card is inserted. We must enter the card choose pin. It will disable the card. SD must always be initialized with a minimum 76 - 160 pulses transmitted to the clock. SD lacks an internal clock primary source.

Command 0 is just a computer reset which puts the SD card to rest. When it reaches this condition, it may be configured to operate in SPI mode. There is only one NCR needed.

cmd_out<= 56'hFF_40_00_00_00_00_95

Command 8 is to check if we are using the correct card. Otherwise, this particular program will always return to the beginning. This part of the initialization procedure is mandatory.

cmd_out<= 56'hFF_48_00_00_01_AA_87;

Then to get R3 reaction after the R1 reaction. The one and only thing we need to understand is that the final byte we get has to be (hex) AA. This signals the detection of an SD card version 2 (SDHC). There is only one NCR necessary. Furthermore, the ACMD41 instruction configures the SD card to operate in SPI mode. There is only one NCR necessary.

cmd_out<= 56'hFF_69_40_00_00_00_01;

During the initial run, the idle flag remains set. This activation procedure is complete if the flag is cleared. Alternatively, command 55 is delivered. The ACMD command has a feature in that all commands are preceded with command 55.

cmd_out<= 56'hFF_77_00_00_00_00_65;

CRC, NCR, and null arguments are often transmitted as 0xFF. The SD card operates in this manner, which explains why the data remains high. Whenever the SD card gets busy, the output data pin goes to ground, and when it is ready, it goes too high. This is helpful when writing on cards. To read a state register, no instructions are required. Fig. 6 illustrates the initialization of Secure Digital card.

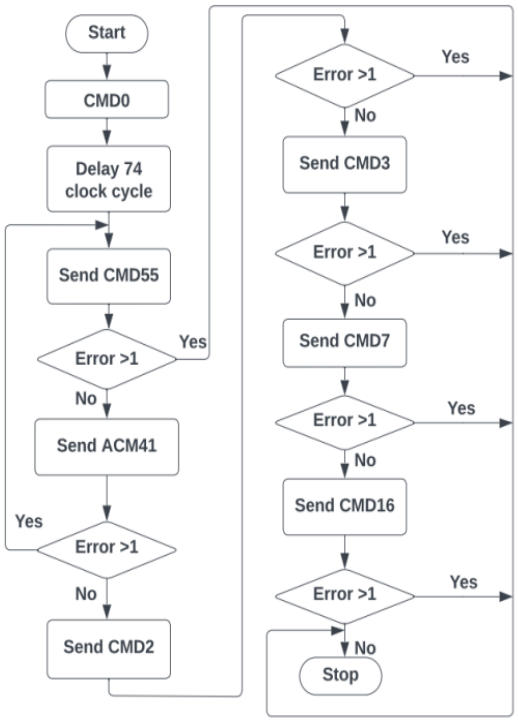

Fig. 6. SD card initialization flow chart.

C. Write Data for SD card

The data could well be transferred to the SD card's 'Memory Core' using the instructions shown below, followed by real data.

WRITE_BLOCK –Single block of data is being written (512 bytes).

WRITE_BLOCK

A block on an SD card has been believed to be a sequence of 512 bytes of memory addresses for as long as anybody can remember. The WRITE BLOCK command is used to permanently store data in a memory block beginning at address 2000. For the command set to operate properly, it must appear as follows:

Everything begins at byte 0x18 (command). Second through fifth bytes (0x000007d0) (argument) (This option is needed to be 0, even when the other commands have no further arguments. (CRC) represents the random values included in the sixth byte. Seven-byte NCR. After receiving the command from the FPGA, the R1 should respond. Every bit must be zero. With the R1 response byte set to zero, the FPGA is able to transmit the data for writing to the SD card. The minimal data length should be 512 bytes. This is true even if the length of the real data is shorter. Immediately following the 512 bytes of data is the Data Token byte.

Next, a 16-bit CRC byte must be utilized for the final verification. The total number of bytes in the data packet will be 515, which is equivalent to 1 plus 512 plus 2. The least significant bit (LSB) is set to 1 in this informative token (0xFE).

The command to be concerned with is 24, and until a clear R1 data is received, dummy bytes are sent; a dummy byte is followed by a token byte (11111110b); the same 512 bytes of data are sent; two CRCs are sent to determine if the data is valid; and finally, a data response is received to determine if the data is viable. This is accomplished in hardware, not via an SD card erase command.

cmd_out<= {16'FF_58, address, 8'FF};

After receiving the data response, the SD card may be written to; however, before the card's status can be verified, it must be de-asserted eight times (1 byte) and then reasserted. While the preceding code snippet may occasionally function and is frequently how datasheets are to be understood, the SD card must be deselected prior to beginning the writing procedure. When the SD card is being used for something else, the output data will be reduced (if this option is set), but when it is no longer in use, the output data will be increased to indicate that it is accessible. Fig. 7 illustrates write data of Secure Digital Card.

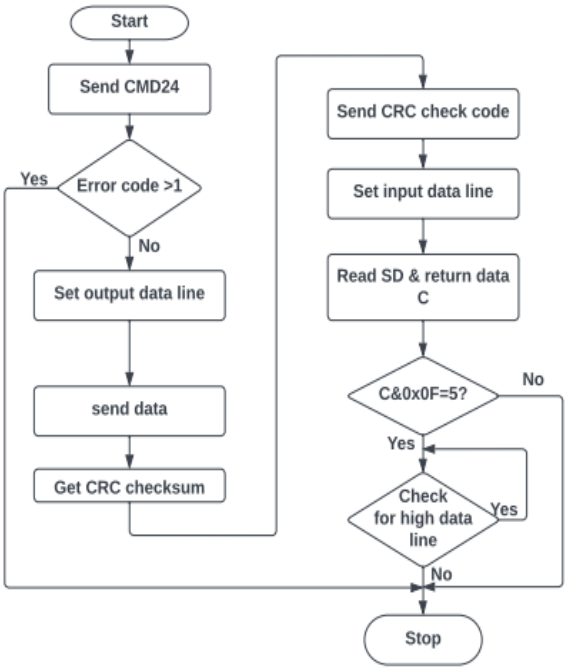

Fig. 7. Write data flow chart for SD card.

D. SD Card Read Data

We can retrieve the information saved in the "Memory Core" of SD card by following the instructions listed below.

A single block of data (512 bytes) is read from the SD card in READ BLOCK mode.

READ_SINGLE_BLOCK

Each SD card block is represented by a 512-byte memory address sequence. The READ SINGLE BLOCK command is used to get a single block of data beginning at the given memory address. In this instance, 2000 is the beginning address. The command packet format must be as follows:
When the time comes to record the succeeding data block.
0x51 is the value of the first byte (command).
(The argument) positions 0x000007d0 through 0x000007ff (This field must be set to zero even when there are no parameters for those other commands.) (CRC) represents the random values included in the sixth byte. Seven-byte NCR. The response from R1 should reach FPGA immediately following the instruction. All bits should be set to zero. The FPGA may get information from the SD card if the R1 response byte is zero. Each READ SINGLE BLOCK request from an SD card returns 512 bytes in response. Fig. 9 illustrates read data of Secure Digital Card.

cmd_out<= {16h'FF_51, address,8'hFF};

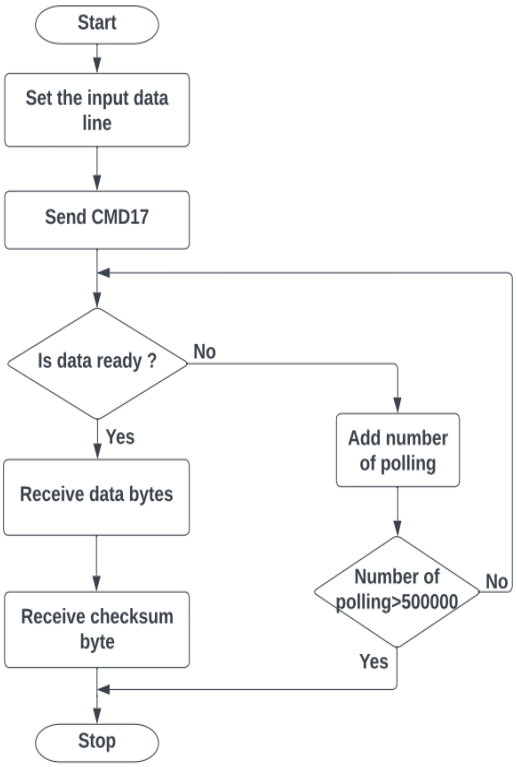

Fig. 8. Read data flow chart for SD card.

E. File Format of FAT32

Data is read from and written to memory modules using FAT32. Initially, clusters concentrate on a small number of sectors. It is displayed in Fig. 9.

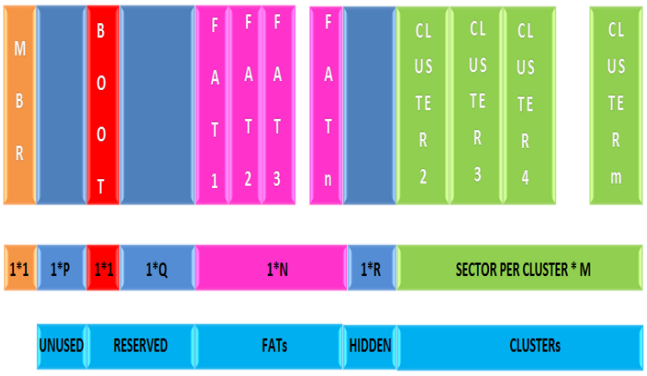

Fig. 9. Format of FAT32.

The first and foremost sector is the MBR which is known as Master Boot Record. This comes after a lengthy time in which many sectors remained inactive. These sectors are designated as reserved ones; the BOOT sector comes first, followed by the FAT sector. The count of FAT sectors is determined by the file system size. Clusters followed FAT sectors and before insecure ones.

F. Reading File from FAT32

Fig. 10 illustrates that it is able to read FAT32-formatted files. This sector read has been discovered to be the final phase of every operation. We can use the READ_SINGLE_BLOCK command supplied by the SD Command Layer to read only this one sector from the memory core of the SD card.

sdcard_fat _32_read. V

From Fig. 10FAT32 sector cluster counts are expressed by 32 bits. Every cluster is represented by a 32-bit set. If each cluster consists of 512 bytes, then the sector will have 128 cluster pointers. This section describes how space is allotted to files in a FAT32 file system.

The following equation may be used to compute the number of the next cluster pointer within the FAT32.

Number of sectors of FAT in the next cluster pointer = number of first sectors in partition + reserved number of sectors + ((clusters of current number * 4)/bytes per sector).

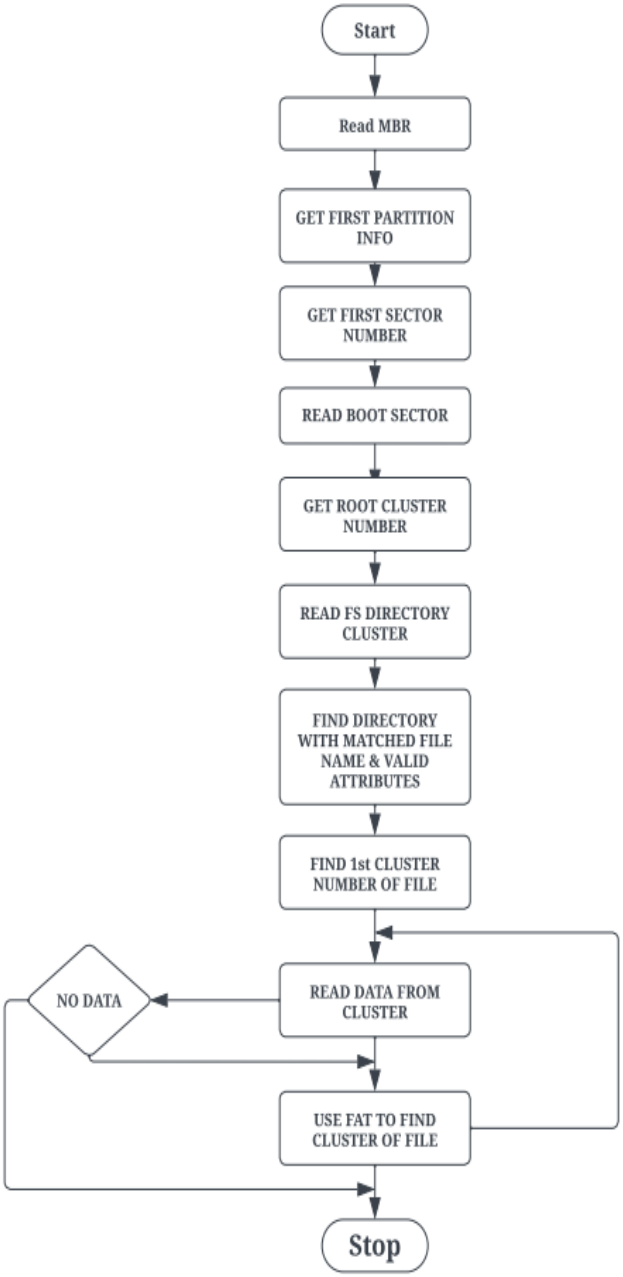

Fig. 10. FAT32 Algorithm for reading.

From Fig. 11 the yellow-colored box indicates the cluster that contain the data for each file in FAT32 and the respective cluster pointers. The red-colored line shows that the next cluster pointer which matches the present cluster is being sought, while the green-colored line shows that the next cluster is being sought using the cluster thing about having in the FAT32 cluster pointer.

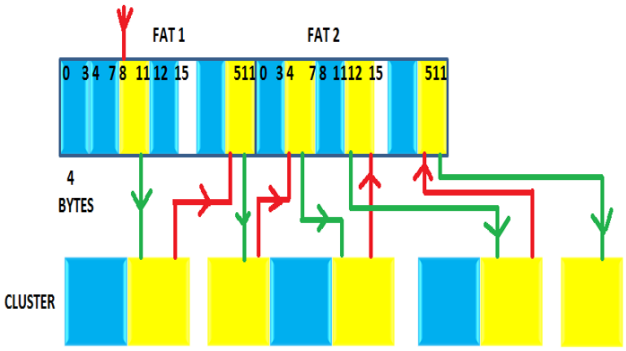

Fig. 11. Read Scrambler file in flash 'Memory Core'.

VI. Results

Table 4. Comparison between existing methods and our method

Proposed Work	**LUT (Look-Up Table)**	**Flip Flop**	**Clock period**
Spartan 6	916	464	5.35 ns
Artix-7	653	413	3.254 ns
Existing Method			
[1] Artix-7	1025	579	10.5ns

Comparing the specifications of the Spartan 6 FPGA (530 Slice Registers, 916 LUT's, 464 Flip Flops, 5.35 ns clock period, 186.925 MHz frequency) and the Artix-7 (512 Slice Registers, 653, LUT's, 413 Flip Flops, 3.254 ns clock period, 307.342 MHz frequency) demonstrates that our proposed results provide better resource utilization than existing methods.

VII. Conclusion

This research proposes an FPGA controller for SD card reading/writing in 4-bit SD mode. It is allowed to address individual cards out of sequence, notwithstanding the architecture's suggestion. There is a large quantity of storage capacity available.

References

[1] D. C. Costache, L. A. Perişoară and A. Florescu, "FPGA Implementation of a SD Card Controller using SPI communication," 2020 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), 2020, pp. 1-4.

[2] Zhenlin LU, Jingjiao LI, Yao Zhang. 2010. The Reading/Writing SD Card System Based on FPGA. 2010 *First International Conference on Pervasive Computing Signal Processing and Applications (PCSPA)*, pp. 419-422.

[3] O. Elkeelany and V. S. Todakar, "Data Archival to SD Card Via Hardware Description Language," in IEEE Embedded Systems Letters, vol. 3, no. 4, pp. 105-108, Dec. 2011.

[4] Gul Munir Ujjan, Abdul Malik, Mohd Zaid Abdullah, Shakil Ahmed, "Implementation of 4-bit data transmission for accessing SD card with FPGA Embedded Soft Processor;" ICIIT '19, February 20–23, 2019, DaNang, Viet Nam.

[5] Jiayi Qiang, Yong Gu and Guochu Chen, "FPGA Implementation of SPI Bus Communication Based on State Machine Method," *Journal of Physics*: *Conference Series 1449 012027, 2020.*

[6] Pallavi Polsani, V. Priyanka.B, Y. Padma Sai, "Design & Verification of Serial Peripheral Interface (SPI) Protocol," *IJRTE*, Volume-8 Issue-6, March 2020.

[7] Fareha Naqvi, "Design and Implementation of Serial Peripheral Interface Protocol Using Verilog HDL," *International Journal of Engineering Development and Research* 2015.

[8] N Nandhagopal, S Navaneethan, V Nivedita, A Parimala, D Valluru "Human Eye Pupil Detection System for Different IRIS Database Images" Journal of Computational and Theoretical Nanoscience volume 18, issue no 4, page no 1239-1242, 2021.

[9] Manish Kumar Birla. 2006. FPGA Based Reconfigurable Platform for Complex Image Processing. *IEEE, 6th International Conference on EIT*, pp.204-209.

[10] V. Asanza, A. Constantine, S. Valarezo and E. Peláez, "Implementation of a Classification System of EEG Signals Based on FPGA," 2020 Seventh International Conference on eDemocracy & eGovernment (ICEDEG), 2020, pp. 87-92.

[11] S Navaneethan, N Nandhagopal, V Nivedita "An FPGA-based real-time human eye pupil detection system using E2V smart camera", Journal of Computational and Theoretical Nanoscience 16 (2), 649-654 2019.

[12] Yansi Yang, Yingyun Yang, Lipi Niu, Huabing Wang and Bo Liu, "Hardware system design of SD card reader and image processor on FPGA," 2011 IEEE International Conference on Information and Automation, 2011, pp. 577-580.

[13] M. K. Birla, "FPGA Based Reconfigurable Platform for Complex Image Processing," 2006 IEEE International Conference on Electro/Information Technology, 2006, pp. 204-209.