

PulseRain
TECHNOLOGY

Doc# TRM-0922-01006, Rev 1.0.0

Copyright © 2017

PulseRain Technology, LLC.

10555 Scripps Trl, San Diego, CA 92131



858-877-3485



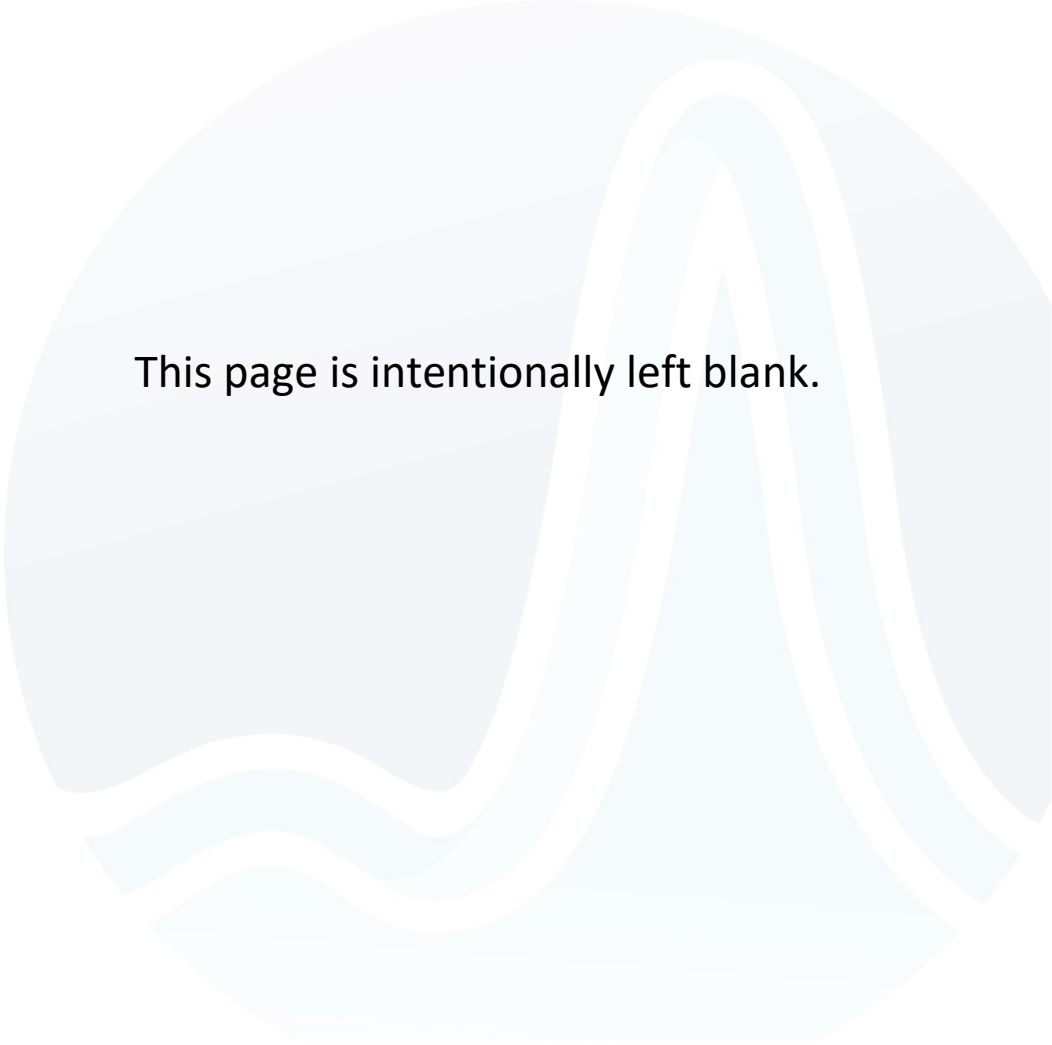
858-408-9550

<http://www.pulserain.com>

PulseRain M10 – microSD

Technical Reference Manual

Sep, 2017



This page is intentionally left blank.

Table of Contents

REFERENCES.....	1
1 INTRODUCTION	2
1.1 THE WHOLE PICTURE.....	2
1.2 THE PHYSICAL INTERFACE	3
1.3 THE FILE SYSTEMS.....	3
2 HARDWARE	4
2.1 ARCHITECTURE	4
2.2 PORT LIST.....	5
2.3 PIN ASSIGNMENT	5
2.4 PING / PONG BUFFER	6
2.5 CRC	6
2.6 REPOSITORY	6
3 SOFTWARE	7
3.1 REGISTER DEFINITION	7
3.2 ADDRESS MAP	8
3.3 WORK FLOW.....	9
3.3.1 <i>Send Command</i>	9
3.3.2 <i>Initialization</i>	9
3.3.3 <i>Read from microSD</i>	9
3.3.4 <i>Write to the microSD</i>	10
3.4 FILE SYSTEMS	10
3.5 ARDUINO LIBRARY.....	10
3.5.1 <i>APIs</i>	10
3.5.2 <i>Examples</i>	12
3.5.3 <i>Scripts</i>	12

References

1. SD Specification Part 1, Physical Layer Simplified Specification, Version 6.00, Apr 10, 2017
2. How to Use MMC/SDC, http://elm-chan.org/docs/mmc/mmc_e.html
3. The schematic of PulseRain M10 board, Doc# SH-0922-0039, Rev 1.0, 02/2017
4. Petit FAT File System Module, http://elm-chan.org/fsw/ff/00index_p.html

1 Introduction

1.1 The Whole Picture

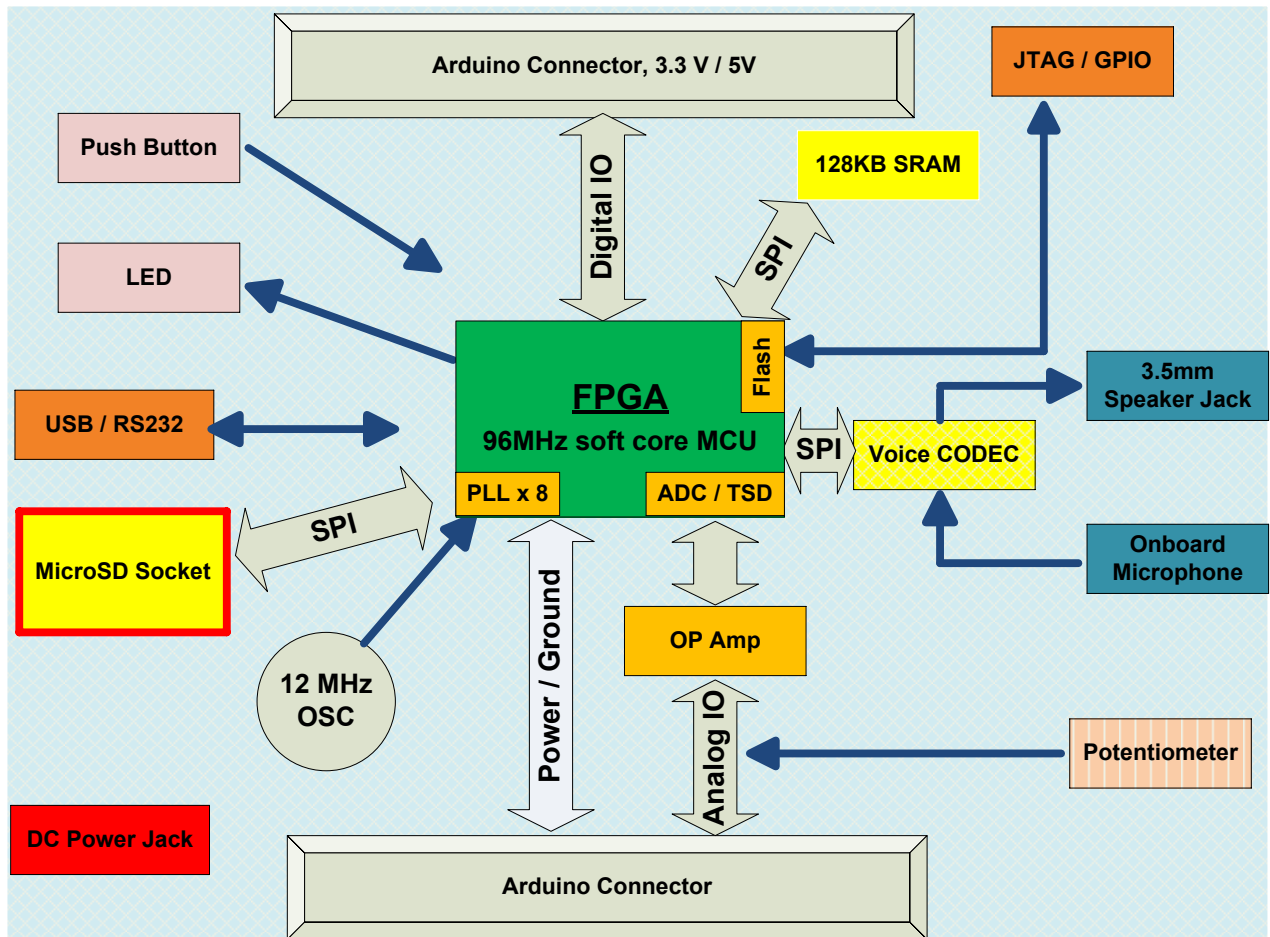


Figure 1-1 The Close View of M10

As shown Figure 1-1, the M10 board takes a distinctive technical approach by embedding an open source soft MCU core (96MHz) into an Intel MAX10 FPGA, while offering an Arduino compatible software interface and form factors. Among all the onboard peripherals, there is a microSD socket. And PulseRain Technology has designed an open source controller inside FPGA that can access the microSD card (provided by the user) through this socket.

1.2 The Physical Interface

The interface of microSD card is defined by SD association, which has the following signal pins:

- DAT 0
- DATA 1
- DATA 2
- CD / DAT 3
- CMD
- CLK

However, the full specification of SD card is only accessible to the member companies of SD association. What's available to the general public is just a simplified version, in which a SPI interface is disclosed (Ref [1]). Thanks to such constraint, most open source embedded systems choose to access the microSD card through SPI bus to avoid any ramification of patent or royalty.

1.3 The File Systems

microSD card is not just a physical interface for raw data. Most of the time it is used with file systems, for which the possible choices are:

- FAT32
- exFAT
- NTFS

However, these files systems are more or less covered by standing patents. Thus practically, most open source embedded systems only choose to support FAT32 with 8.3 short file names. (8 characters at most for the name, followed optionally by 3 characters of extension, with a period standing in between.). And the file names are also case insensitive.

2 Hardware

2.1 Architecture

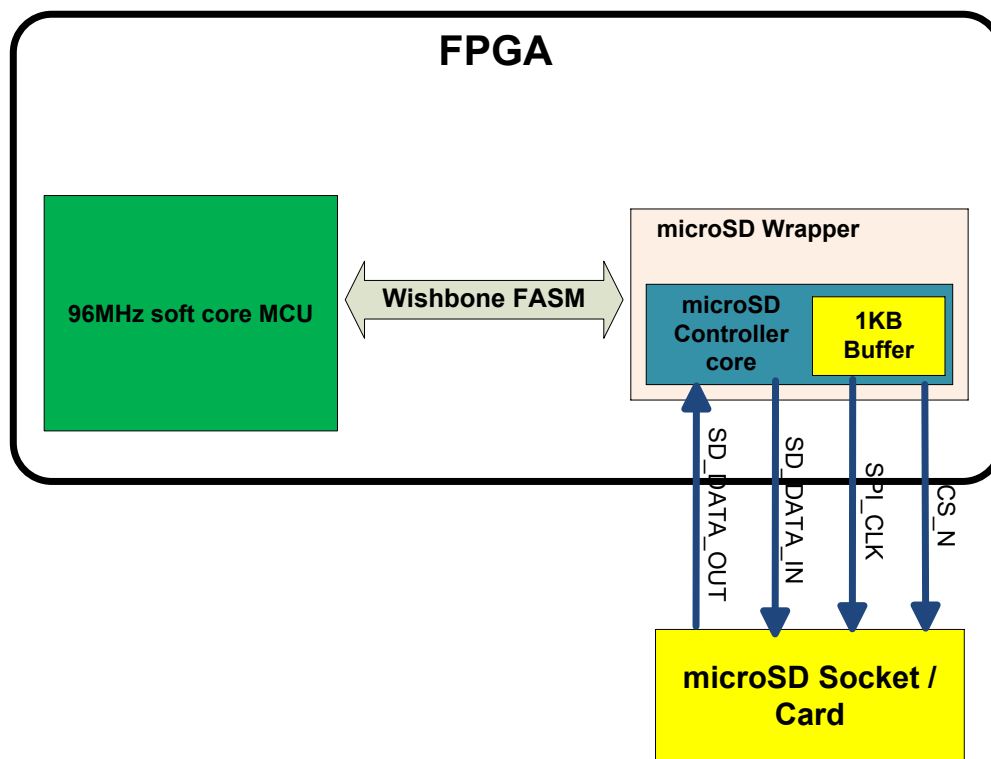


Figure 2-1 The Controller for microSD

As illustrated in Figure 2-1, the microSD controller is mainly composed of two parts: the microSD controller core and the Wishbone wrapper. The Wishbone wrapper is used to communicate with the MCU core for register read/write. And the controller core communicates with the microSD card through 4 pins:

- CS_N (chip select, active low)
- SPl_CLK (clock, default to 8MHz for normal operation)
- SD_DATA_IN (Data from FPGA to microSD)
- SD_DATA_OUT (Data from microSD to FPGA)

PulseRain M10 microSD – Technical Reference Manual

2.2 Port List

The port list of the microSD controller core is defined in Table 2-1.

Group Name	Signal Name	In/Out	Bit Width	Description
Clock / Reset	clk	Input	1	Clock input, 96MHz
	reset	Input	1	Asynchronous reset, active low
	sync_reset	Input	1	synchronous reset, active high
command/ response	response_type	Input	2	The format of the response. The valid values are: 2'b00: R1 or R1b 2'b01: R2 2'b10: R3 or R7 Please see Ref [1][2] for the meaning of those values
	sclk_slow0_fast1	Input	1	0 for slow SPI clock (200kHz), and 1 for fast SPI clock (8MHz) The slow clock is used during initialization stage, while the fast clock is used for normal operation.
	start	Input	1	start pulse to send out the command, response and argument.
	cmd	Input	6	command to be sent out. See Ref [1][2] for more detail.
	arg	Input	32	argument to be sent out. See Ref [1][2] for more detail.
ping / pong buffer	addr_base	Input	10	base address for sequential access from the microSD side
	mem_wr	Input	1	1 for memory write, 0 for read. (random access from host side)
	mem_addr	Input	10	r/w address for random access from host side
	data_in	Input	8	data from the host side
	data_en_out	Output	1	data enable out to the host side
SPI Interface	data_out	Output	8	data to the host side
	cs_n	Output	1	chip select, active low
	spi_clk	Output	1	200kHz (slow clock) or 8MHz (normal clock)
	sd_data_out	Input	1	data from microSD card to FPGA
return / done	sd_data_in	Output	1	data from FPGA to microSD
	done	Output	1	done signal (pulse) when a transaction is finished.
return / done	ret_status	Output	2	Return Status: (will be cleared to zero at start) 2'b01: time out 2'b10: CRC fail 2'b11: OK

Table 2-1 Port List of microSD Controller Core

2.3 Pin Assignment

The pin assignment for the microSD Controller is as following for the onboard FPGA (10M08SAE144C8G):

Signal Name	FPGA Pin Assignment (10M08SAE144C8G)	Description
SD_SPI_CS	69	SPI chip select, DAT3 on the microSD socket
SD_SPI_DO	64	SPI data out, DAT0 on the microSD socket
SD_SPI_DI	66	SPI data in, CMD on the microSD socket
SD_SPI_CLK	65	SPI clock

Table 2-2 FPGA Pin Assignment

2.4 Ping / Pong Buffer

As illustrated in Figure 2-1, the microSD controller core has a buffer of 1024 bytes, which is exactly the size of two sectors for microSD. And this buffer can be used as a ping / pong buffer for both the MCU and the microSD to access.

Interrupt is not supported for the microSD controller, as the ping /pong buffer suffices to keep the throughput high.

2.5 CRC

CRC will be automatically generated by hardware. For SPI mode, CRC is only used for CMD0 and CMD8. And the hardware will fill the CRC according to the following:

Command	CRC7
CMD0	0x4A
CMD8	0X43
default	0x7F

Table 2-3 CRC7 for SPI Mode

2.6 Repository

The RTL code for microSD controller and its Wishbone wrapper is part of PulseRain Technology's RTL library, which can be found on GitHub:

https://github.com/PulseRain/PulseRain_rtl_lib

And look in the "SD" folder for more detail.

3 Software

3.1 Register Definition

The Wishbone wrapper shown in Figure 2-1 contains all the registers to control the microSD controller. In a nutshell, the registers are defined as following:

- CMD (command)

Bits	R/W	Default	Name	Description
5:0	RW	0	command	The 6-bit command to be sent to microSD card. The definition of the valid commands can be found in Ref [1][2]
7:6	RO	0	ret_status	The return status of the command. The possible values are: 0x1: time out 0x2: CRC fail 0x3: OK

Table 3-1 Bit Map for CMD (Command Register)

- ARG0 – ARG3 (8 bit each)
32-bit argument to accompany the command. ARG3 holds the MSB while ARG0 holds the LSB. See Ref [1][2] for more information on each command's argument.
- DATA_OUT (8 bit)
Write only. Data being written to this register will be saved in the ping / pong buffer, and be sent to microSD card afterwards.
- DATA_IN (8 bit)
Read Only. Use this register, along with the buffer address, to read the content of ping / pong buffer.
- BUFFER_ADDR (8 bit)
As shown in Figure 2-1, the microSD controller has a buffer of 1024 bytes, which needs 10-bit address to access. The lower 8 bits are stored in this register. while the higher 2 bits are in CSR.

PulseRain M10 microSD – Technical Reference Manual

- CSR (Control and Status Register)

The bits for CSR are defined in Table 3-2.

Bits	R/W	Default	Name	Description
0	WO	0	sync_reset	Write 1 to this bit to synchronously reset the microSD controller
1	WO	0	sd_start	Write 1 to this bit to start a transaction for microSD
2	WO	0	inc_addr	Write 1 to this bit to increase the current buffer address by 1
4:3	WO	0	response_type	The format of the response. The valid values are: 2'b00: R1 or R1b 2'b01: R2 2'b10: R3 or R7 Please see Ref [1][2] for the meaning of those values
5	WO	0	sclk_slow0_fast1	SPI clock selection. 0 for slow SPI clock (200kHz), and 1 for fast SPI clock (8MHz).
7:6	WO	0	buffer_addr [9 : 8]	The higher 2 bits of the 10-bit ping /pong buffer address

Table 3-2 Bit Map for CSR (Control and Status Register)

3.2 Address Map

The registers defined in Section 3.1 are mapped into MCU's address space, as shown in Table 3-3.

Address	Register Name
0xD7	SD_CSR
0xD8	SD_CMD
0xD9	SD_ARG0
0xDA	SD_ARG1
0xDB	SD_ARG2
0xDC	SD_ARG3
0xDD	SD_BUF_ADDR
0xDE	SD_DATA_IN
0xDF	SD_DATA_OUT

Table 3-3 Address Definition for microSD Controller

3.3 Work Flow

3.3.1 Send Command

To send a command to the microSD card and get response back, do the following:

1. Write the 6-bit command into CMD register.
2. Write the 32-bit argument into ARG0-ARG3, with ARG3 holding the MSB and ARG0 holding the LSB.
3. Set up the buffer address (8 LSBs in BUFFER_ADDR register, 2 MSBs in CSR). If there is any data expected with the associated command, this buffer address will be the base address in ping / pong buffer. And microSD will incrementally write data to (or read data from) the ping / pong buffer starting with this address.
4. Setup the correspondent response_type in CSR (Table 3-2), and set the sd_start in CSR to high to start the command.
5. Keep reading the ret_status in CMD register (Table 3-1) until it becomes non-zero. Check the ret_status against those valid values in Table 3-1 to see if it is successful.

3.3.2 Initialization

The microSD card will NOT enter SPI mode automatically after power on. To make it work under SPI mode, the following needs to be done:

1. Set the SPI clock to be a slow clock (set the bit 5 of CSR to be 0, which will generate a SPI clock of 200kHz by default.).
2. Send CMD0 to microSD. The mode change will be successful if the microSD returns a value of 1 as idle state.
3. Set the SPI clock to be a high clock for normal operation (set the bit 5 in CSR to be 1, which will generate a SPI clock of 8MHz.)
4. The type of the microSD card in the socket needs to be validated. Send CMD8 with argument 0x000001AA, and get the return value for SD card version. It should be version 2 for most microSD card used today.
5. Send CMD55 followed by ACMD41 to start the initialization inside microSD (Ref [2]).
6. After the initialization is completed, read microSD's OCR register with CMD58, and check CCS flag (bit 30). If it is set, the card is a high-capacity card known as SDHC/SDXC. (Ref [2]). The card type is now fully determined.

3.3.3 Read from microSD

The microSD card works on the granularity of sectors. At this point, only sector size of 512 byte is supported by the M10SD library and the microSD controller. To read one sector from the microSD card, do the following:

1. Send CMD17 with sector index as the argument to the microSD card
2. Read the sector data from the ping / pong buffer.

PulseRain M10 microSD – Technical Reference Manual

3.3.4 Write to the microSD

Similarly, to write one sector of data to the microSD card, do the following:

1. Write one sector of data to the ping / pong buffer.
2. Send CMD24 to the microSD card.

3.4 File Systems

As mentioned early, the microSD usually does NOT work on raw data directly. Instead, it uses a file system to manage files. And for that matter, a small module called Petit FAT File System Module (Ref [4]) has been integrated into M10SD library, which supports the following:

- FAT32 File System, with single volume
- Maximum file size is 2GB
- File names are case insensitive, and have to be in 8.3 short name format (8 characters at most for the name, followed optionally by 3 characters of extension, with a period standing in between.)
- File Creation and File size expansion are not supported.
This is because for file creation and file size expansion, new free clusters have to be found, which involves a long search and cluster reorganization. And it is decided to forgo such features in favor of simplicity and smaller code size.

Thus, if the user wants to write to a file, the file should be pre-existing on the microSD card. For that matter, the M10SD library has come with a python script called "bin_file_fill.py" to help the user create a file with specified size, and fill it with dummy bytes.

3.5 Arduino Library

3.5.1 APIs

Although the microSD is a complicated matter by itself, the introduction of file system has made the APIs quite straightforward. Only 4 APIs are exposed by the M10SD library

- `uint8_t begin ()`

Call this function to initialize the microSD card and mount the file system. The possible return values are:

- RES_OK (0): The initialization is successful and the FAT32 file system has been mounted
- RES_ERROR (1): Disk Error, the file system is not FAT32
- RES_NOTRDY (2): Mount failed. The disk is not ready

- *uint8_t fopen (const uint8_t *path)*

Call this function to open a file on the microSD card.

Parameters:

path: the name of the file (including the path) to be opened

Return Value:

non-zero: file open failed

0: file open succeeded

- *uint8_t fread (uint8_t *buff, uint16_t btr, uint16_t *br)*

Call this function to read data from the opened file into a buffer

Parameters:

buff: pointer to the data buffer to be filled

btr: the number of bytes to be read from the microSD

br: pointer to a uint16_t variable, which will store the number of bytes that are actually received from the microSD after this function is executed.

Return Value:

non-zero: file read failed

0: file read OK

- *uint8_t fwrite (uint8_t *buff, uint16_t btw, uint16_t *bw)*

Call this function to write data to the opened file

Parameters:

buff: pointer to the data to be sent to microSD

btw: the number of bytes to be written

bw: pointer to a uint16_t variable, which will store the number of bytes that are actually written to the microSD after this function is executed.

Return Value:

non-zero: file read failed

0: file read OK

3.5.2 Examples

To further facilitate the software development, the following examples can be referenced:

- *wav_play*

This example will read a wav file named "SPIDER.WAV" from the microSD card, and play it through the onboard CODEC (Si3000). This wav file is supposed to be mono channel, sampled at 8kHz.

To run this example, find a microSD card and format it with FAT32. The M10SD library has the "SPIDER.WAV" file in the "extras" folder. Copy this file to the root of the microSD card. And then run the example of *wav_play.ino* in Arduino IDE to hear the sound playing. (A speaker or headset has to be connected to the M10 board through 3.5mm audio jack).

3.5.3 Scripts

To assist the development, the following scripts are also provided in the "extras" folder of M10SD library:

- *bin_file_fill.py*

See Section 3.4 for more detail.