Lab 4 Solid-State Disk and File System

Michael Coulter and Jake Klovenski

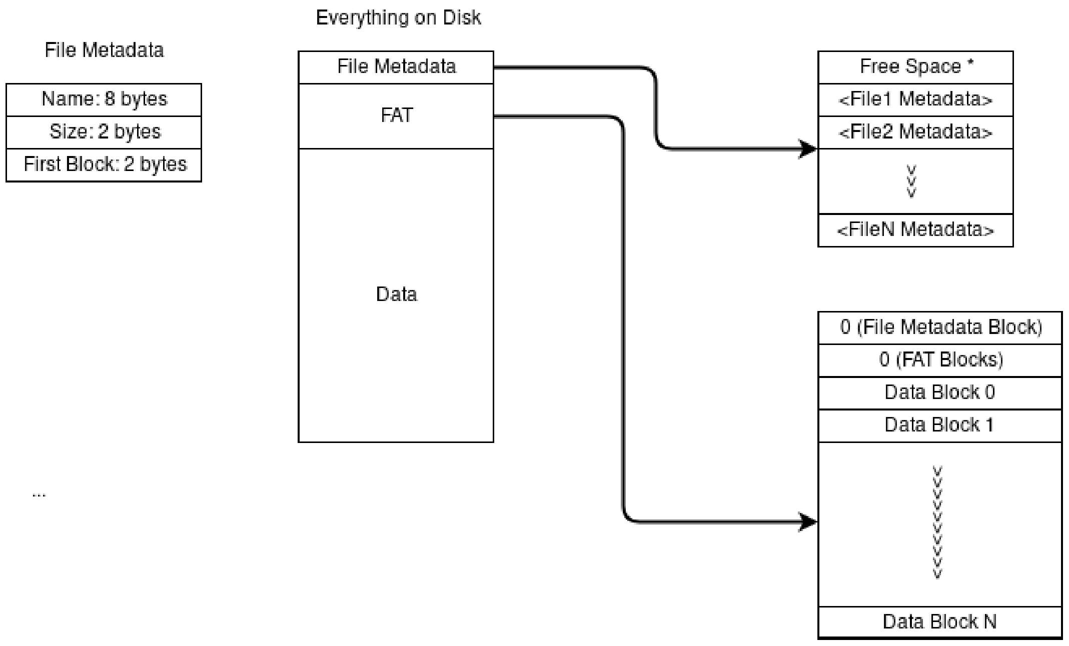
A) Objectives (1/2 page maximum)

Implement a simple FAT file system on the TM4C using the SD Card reader on the ST7735. The board will communicate with the SD Card interface using the same serial connection. The file system will allow the OS to both read from and write to the SD card and allow data to persist through power cycles. The file system is not responsible for data lost due to power being lost before the file is closed. Only one file may be open at a time, and that file may only be open for reading or writing, not both.

B) Hardware Design (none)

C) Software Design (documentation and code)

1) Pictures illustrating the file system protocol, showing:free space management;the directory file allocation schema

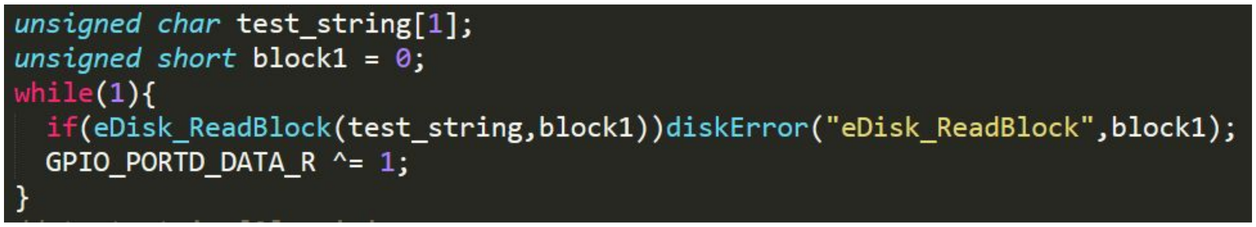


2) Middle level file system (eFile.c and eFile.h files)
    a) Included in Git repository.
3) High level software system (the new interpreter commands)
    a) Included in Git repository.

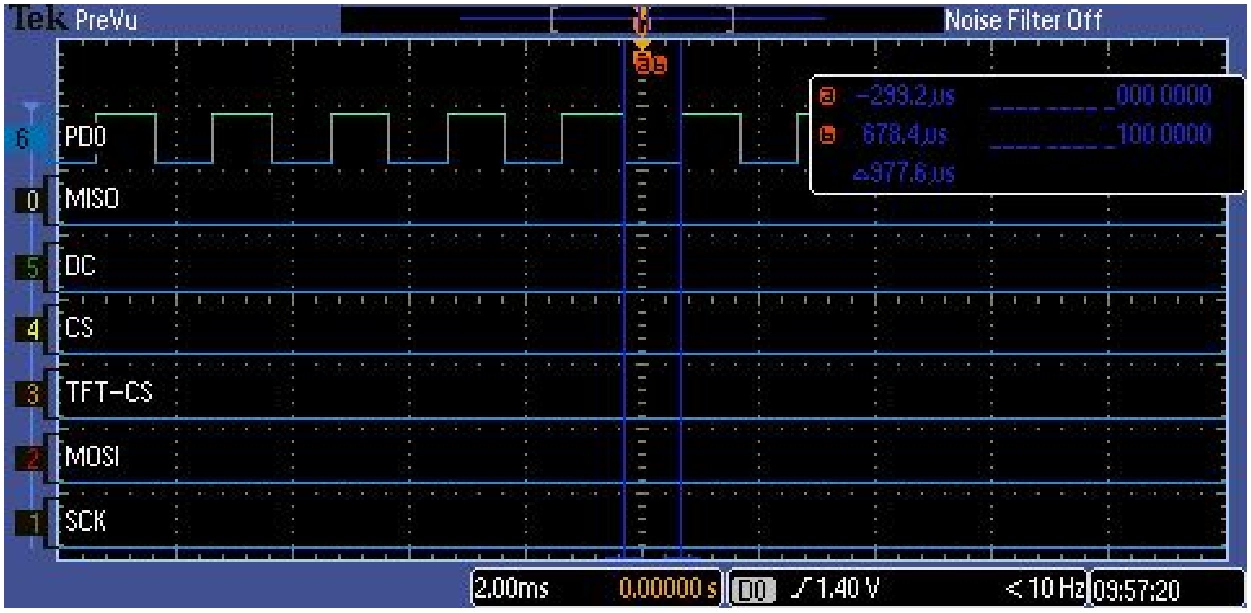
D) Measurement Data

1) SD card read bandwidth and write bandwidth (Procedure 1)

Read Bandwidth

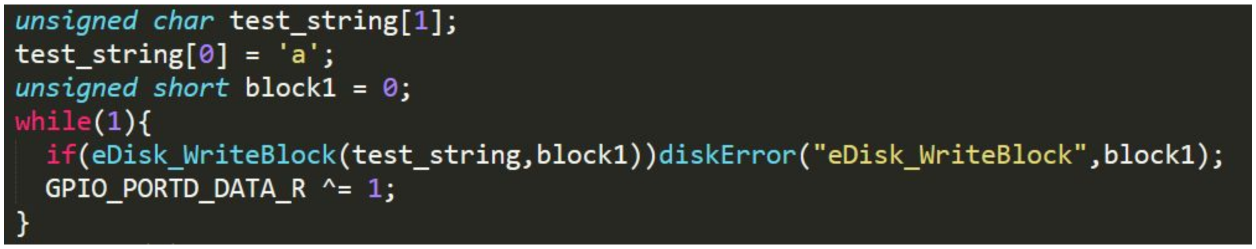


```
unsigned char test_string[1];
unsigned short block1 = 0;
while(1){
  if(eDisk_ReadBlock(test_string,block1))diskError("eDisk_ReadBlock",block1);
  GPIO_PORTD_DATA_R ^= 1;
}
```

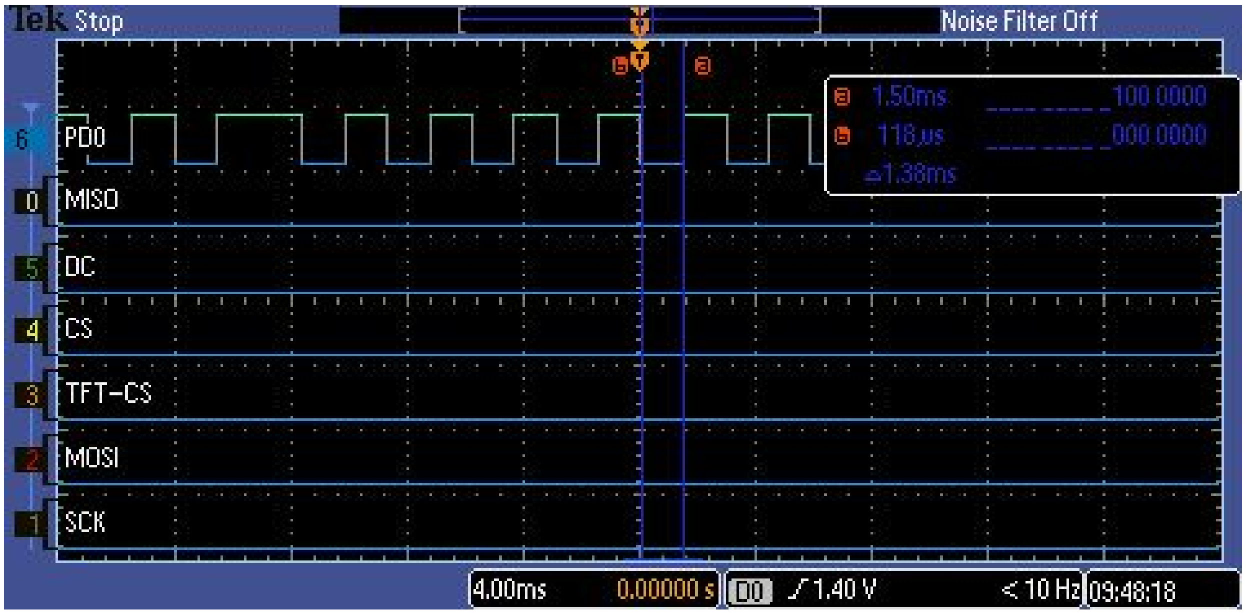


It takes about 977.6us to read one byte from the SD card.

Write Bandwidth

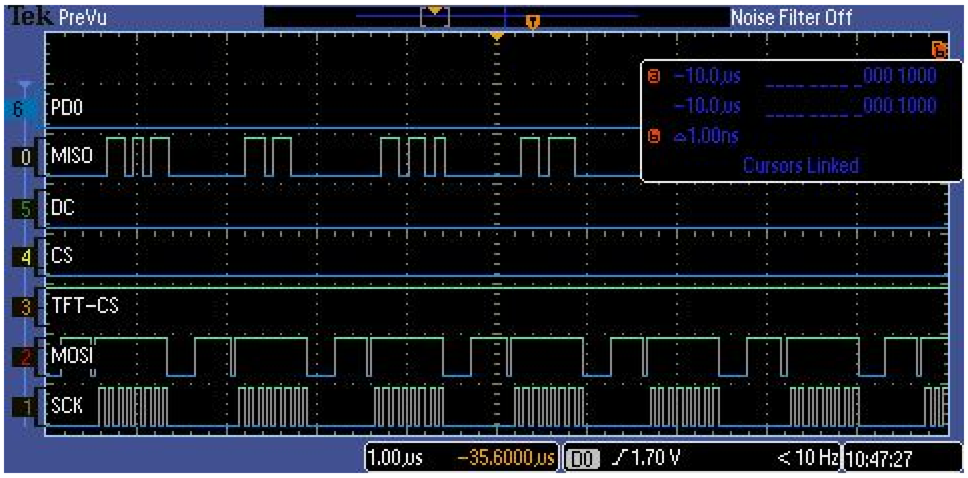


```
unsigned char test_string[1];
test_string[0] = 'a';
unsigned short block1 = 0;
while(1){
  if(eDisk_WriteBlock(test_string,block1))diskError("eDisk_WriteBlock",block1);
  GPIO_PORTD_DATA_R ^= 1;
}
```

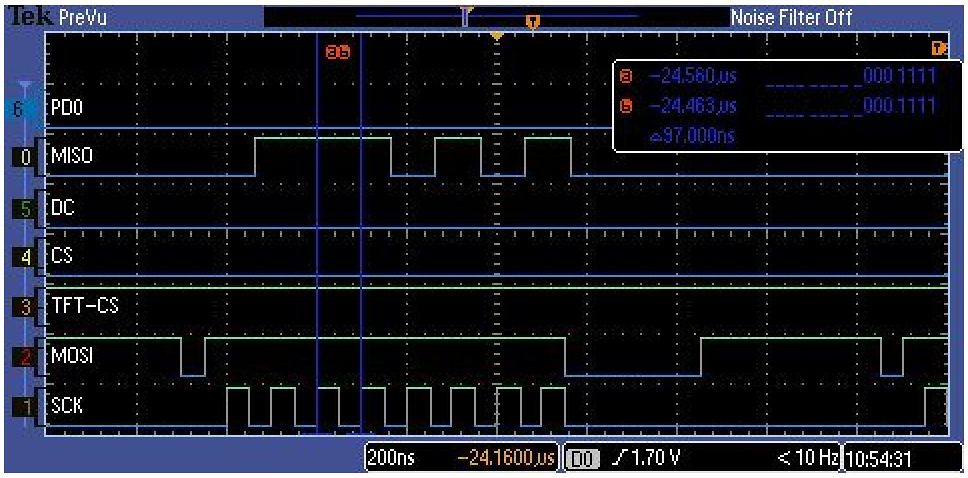


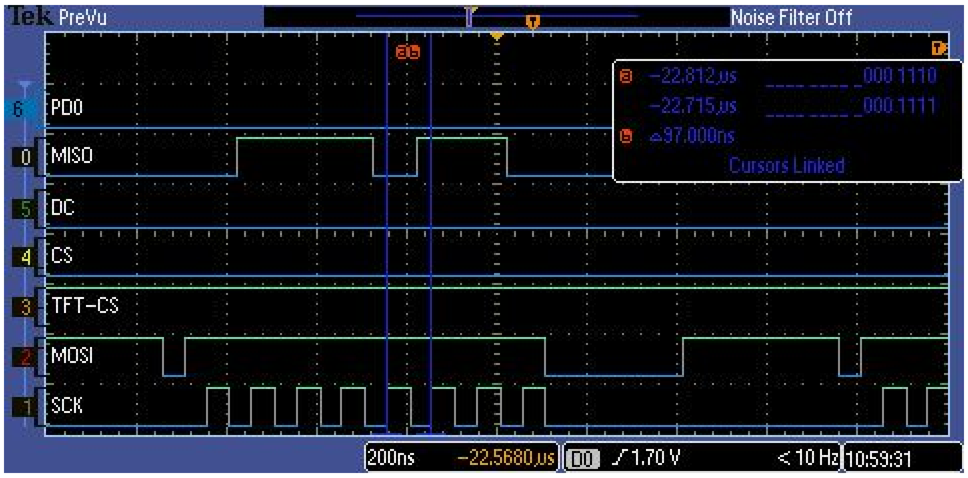
It takes about 1.38ms to write one byte to the SD card.

2) SPI clock rate (Procedure 1)



3) Two SPI packets (Procedure 1)


Tek PreVu
Noise Filter Off
-24.560μs ________000 1111
-24.463μs ________000 1111
Δ97.000ns
PD0
MISO
DC
CS
TFT-CS
MOSI
SCK
200ns -24.1600μs D0 1.70 V < 10 Hz 10:54:31


Tek PreVu
Noise Filter Off
-22.812μs ________000 1110
-22.715μs ________000 1111
Δ97.000ns
Cursors Linked
PD0
MISO
DC
CS
TFT-CS
MOSI
SCK
200ns -22.5680μs D0 1.70 V < 10 Hz 10:59:31

E) Analysis and Discussion (2 page maximum). In particular, answer these questions

1) Does your implementation have external fragmentation? Explain with a one sentence answer.
    a) No, we do not have any external fragmentation because all blocks are the same size and if a file needs a larger file size than the block, a new block will be allocated that may or may not be the next block in memory.
2) If your disk has ten files, and the number of bytes in each file is a random number, what is the expected amount of wasted storage due to internal fragmentation? Explain with a one sentence answer.
    a) At most, we will have 5120 lost to internal fragmentation because a whole block is allocated when the file is created and before the file is written to. This could potentially result in 10 empty allocated blocks.

3) Assume you replaced the flash memory in the SD card with a high speed battery-backed RAM and kept all other hardware/software the same. What read/write bandwidth could you expect to achieve? Explain with a one sentence answer.
    a) RAM is faster than flash memory, so we would expect faster read/write bandwidths.
4) How many files can you store on your disk? Briefly explain how you could increase this number (do not do it, just explain how it could have been done).
    a) Our current file system implementation can currently hold a maximum of 42 files. This is because we only have one block allocated for the file metadata. This can be increased by allocating an additional block of storage for files metadata which would double the amount of files we can store. We will gain an additional 42 files for every block allocated.
5) Does your system allow for two threads to simultaneously stream debugging data onto one file? If yes, briefly explain how you handled the thread synchronization. If not, explain in detail how it could have been done. Do not do it, just give 4 or 5 sentences and some C code explaining how to handle this.
    a) We do not support multiple threads streaming debugging data onto a file. This is because we only have one buffer for storing a block before writing it back to memory. We can improve this by making each thread have a file buffer and making writes to disk atomic.