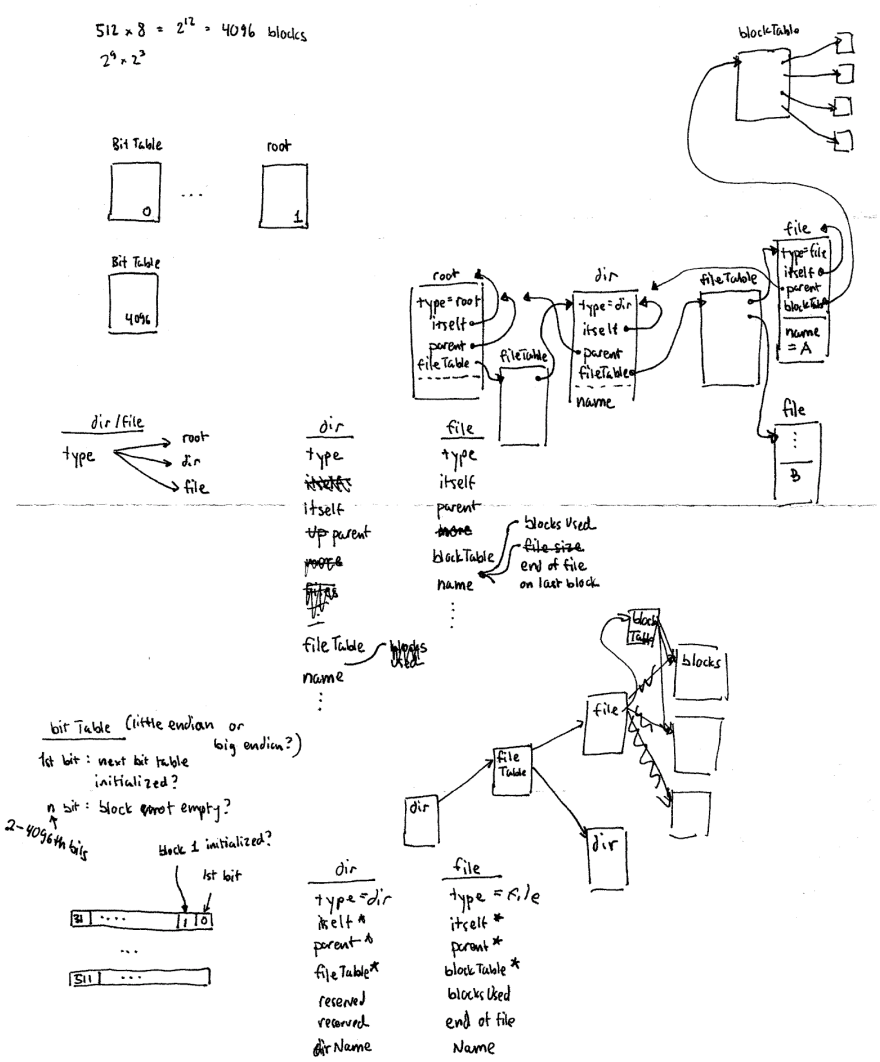
A) Objectives (1/2 page maximum)

In this lab we designed and implemented a persistent filesystem that interfaced the launchpad with an SD card. We also implemented interpreter commands that allow us to send commands directly to the middle-level filesystem interface to make, write to, and read from files.

B) Hardware Design (none)

C) Software Design

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



We have a bit table that indicates which blocks are in use; each bit represents each block so we can indicate up to 512 blocks are free or used. To find the next free block, we simply search for the next '0' in our bit table.

Our directory structure is using pointers to files. Our implementation only uses root, but if we wanted directories, we would have pointers to directories as well.

Our file structure is an inode. The inode contains data such as name, size, and a pointer to a blockTable. The blockTable is a list of pointers to blocks allocate to the file, we can determine which pointers are valid based on value (ie -1 means the pointer is not valid).

2) Middle level file system (eFile.c and eFile.h files)

See our code.

3) High level software system (the new interpreter commands)

Our interpreter commands are 1-to-1 function calls of our eFile functions. For the file display, we simply call eFile_Read() until we get an error.

D) Measurement Data

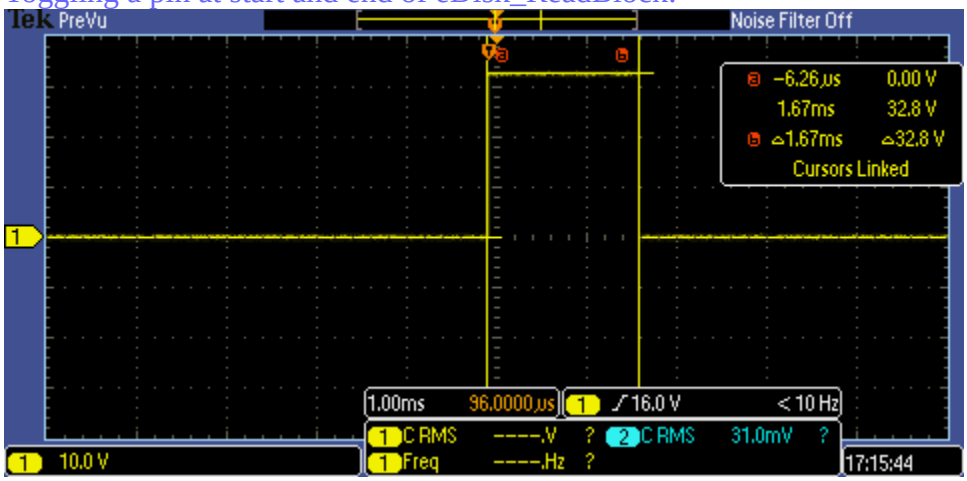
1) SD card read bandwidth and write bandwidth (Procedure 1)

We take 1.67ms to read a block, so our bandwidth is 599 blocks/s = 306587 B/s.

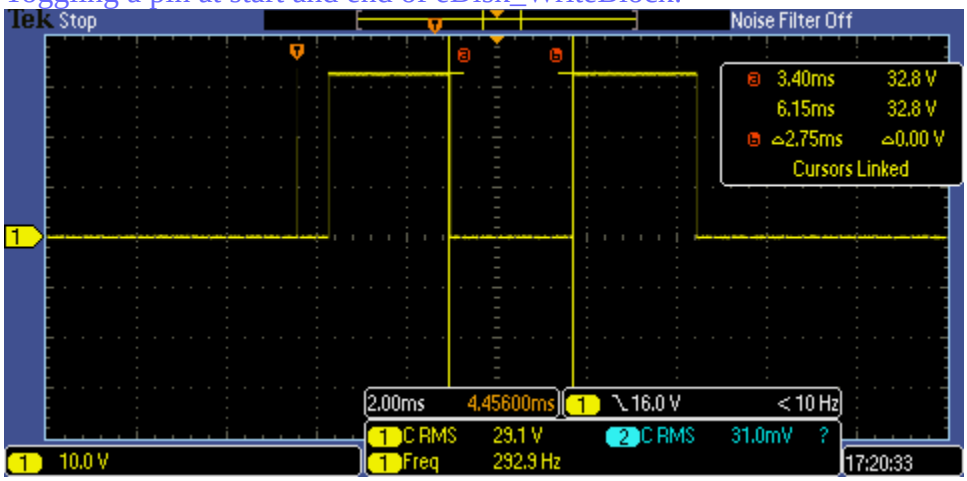
We take 2.75ms to write a block, so our bandwidth is 364 blocks/s = 186182 B/s.

We measured this by toggling a pin at the beginning and end of eDisk_WriteBlock and eDisk_ReadBlock.

Toggling a pin at start and end of eDisk_ReadBlock:

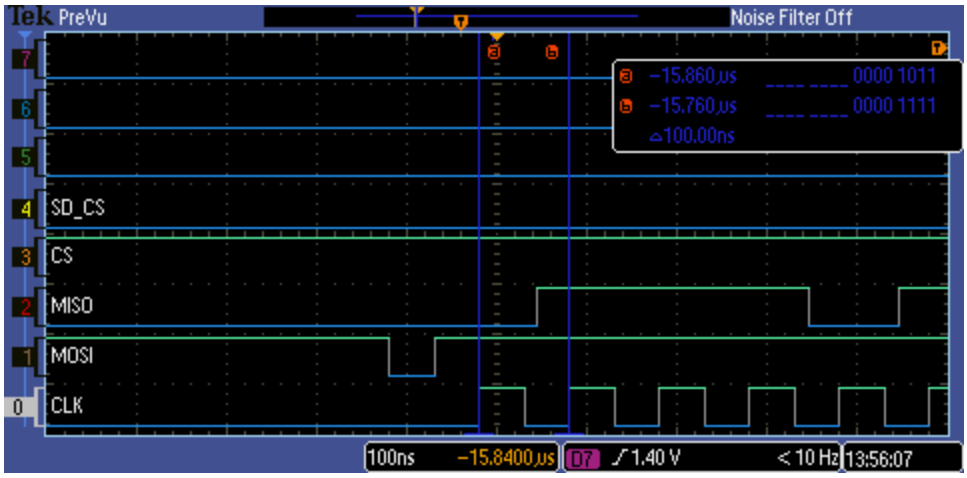


Toggling a pin at start and end of eDisk_WriteBlock:



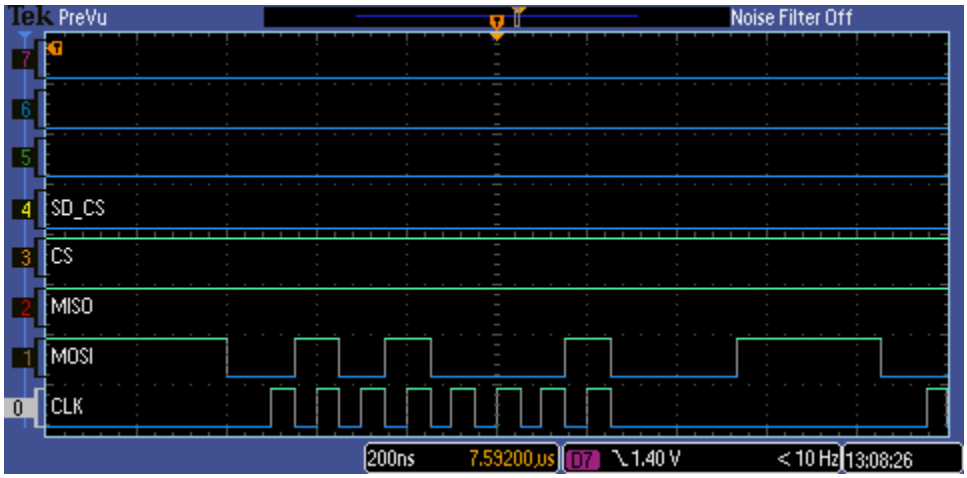
2) SPI clock rate (Procedure 1)

The SPI clock rate is 100ns => 10MHz

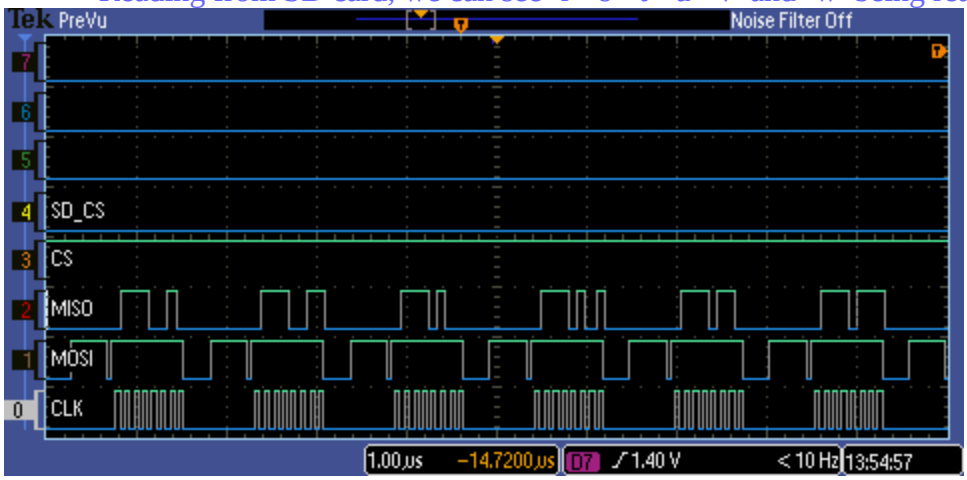


3) Two SPI packets (Procedure 1)

Command 17:



Reading from SD card, we can see ‘r’ ‘s’ ‘t’ ‘u’ ‘v’ and ‘w’ being read in increasing order.



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.

No, we use pointers to blocks so we can always utilize the entire disk.

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.

(number of blocks used by 10 files)*512 – (size of 10 files).

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.

We can expect the bandwidth to be our clock rate or less; in this case 800ns/bit => 1.25MB/s; this is bounded by our SPI clock rate.

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).

We can store 128 files. We can increase this number by implementing directories or by increasing more pointers to files.

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 the synchronization.

No, one possible way to have 2 threads stream to the same file is by using semaphores on every write. Obviously, we will also need synchronize the threads themselves to write in the correct order. To make it work, one thread opens the file for write, then both threads can call eFile_Write(); this implementation does not support 2 or more files.

```
eFile_Write(char data){
OS_bWait(&writeLock)
/* other code */
OS_bSignal(&writeLock)
}
```