A Hardware Filesystem Implementation for High-Speed Secondary Storage

Dr.Ashwin A. Mendon , Dr.Ron Sass
Electrical & Computer Engineering Department
University of North Carolina at Charlotte

Presented by: Manoja P Rao
ECGR 6185 Adv. Embedded Systems
April 17th 2013

Introduction

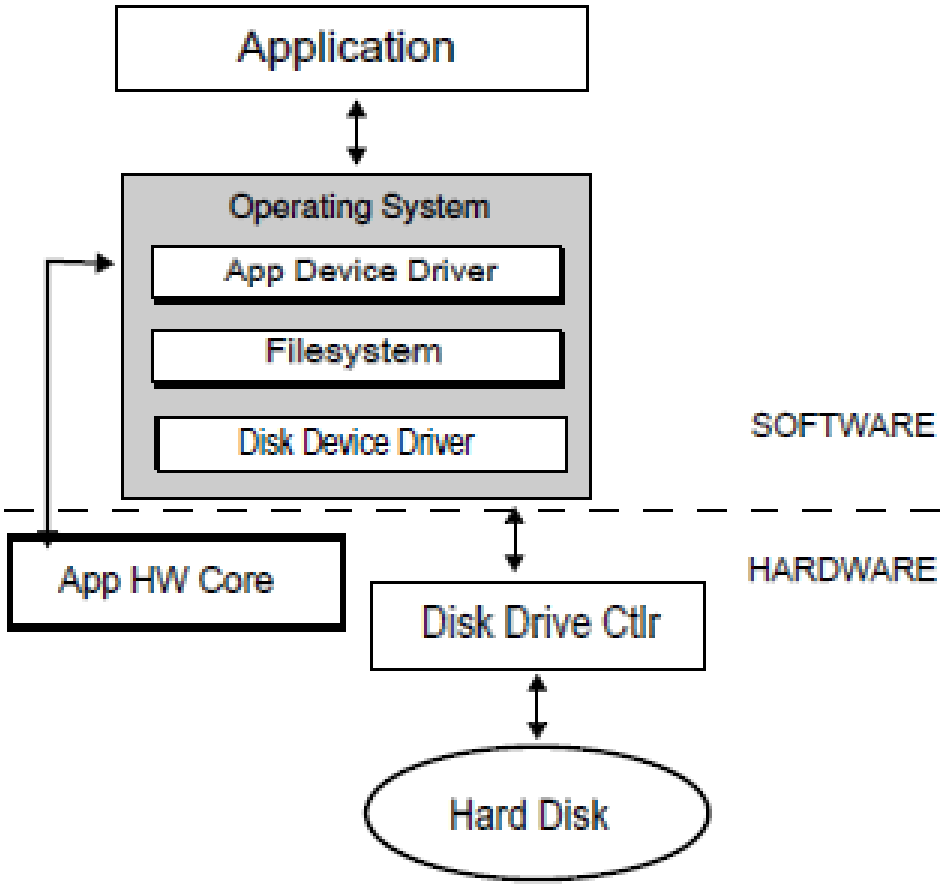
- Platform FPGAs are capable of hosting entire Linux based systems including standard peripherals, integrated network interface cards and even disk controllers on a single chip.
- Filesystems, however, are typically implemented in software as part of the operating system.
- Some applications are very sensitive to file I/O latency and Platform FPGA processor cores are clocked at relatively slow frequencies.
- This paper describes a design and implementation of a filesystem in hardware.

Introduction

- In many simulation experiments like weather forecasting, computational scientists are forced to code their algorithms so that data is explicitly moved between secondary storage and main memory
- If part of the computation is performed by accelerators implemented in the programmable logic of an FPGA, then the data does not necessarily have to go through all of the traditional layers of an OS to reach the core.
- By moving this functionality into hardware, the proposed system gives computational accelerator cores direct access to the files on a disk.

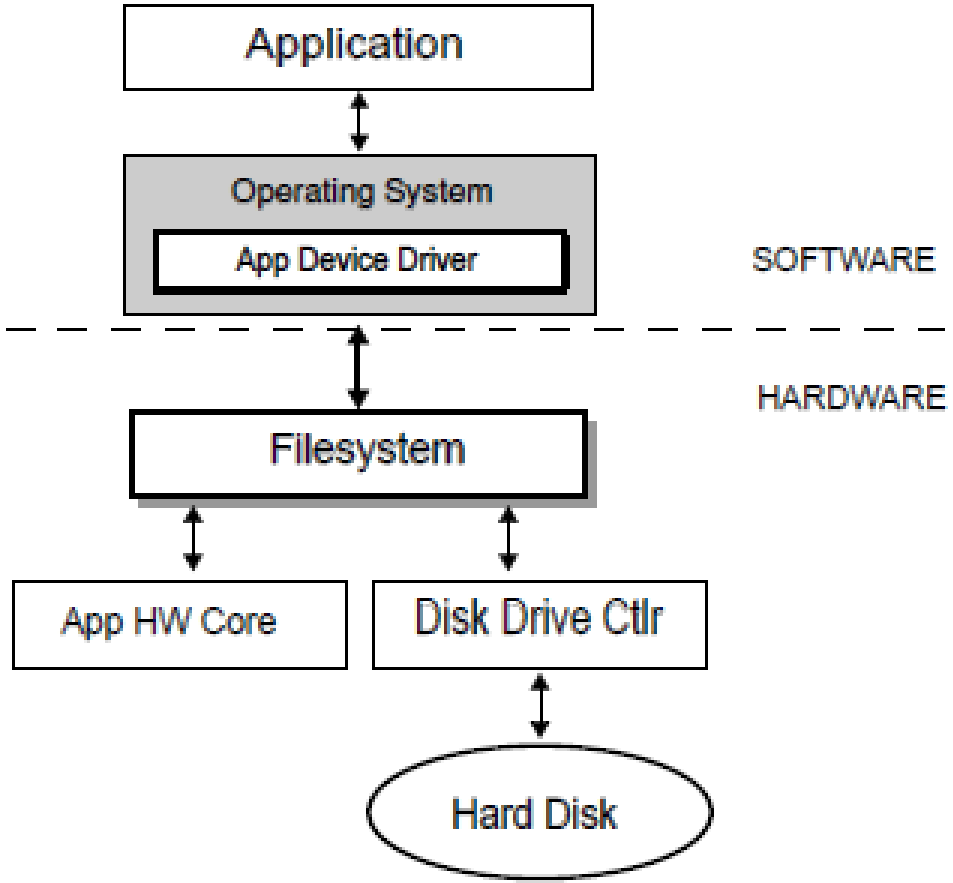
Introduction

Traditional Organization



Introduction

Filesystem-in-Hardware



Background

super block: describes state of the filesystem such as blocksize, filesystem size, number of files stored and free space information

inode list: list of pointers to data blocks

data blocks: contain actual file data

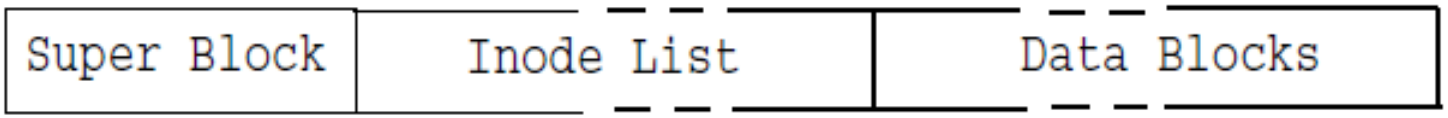
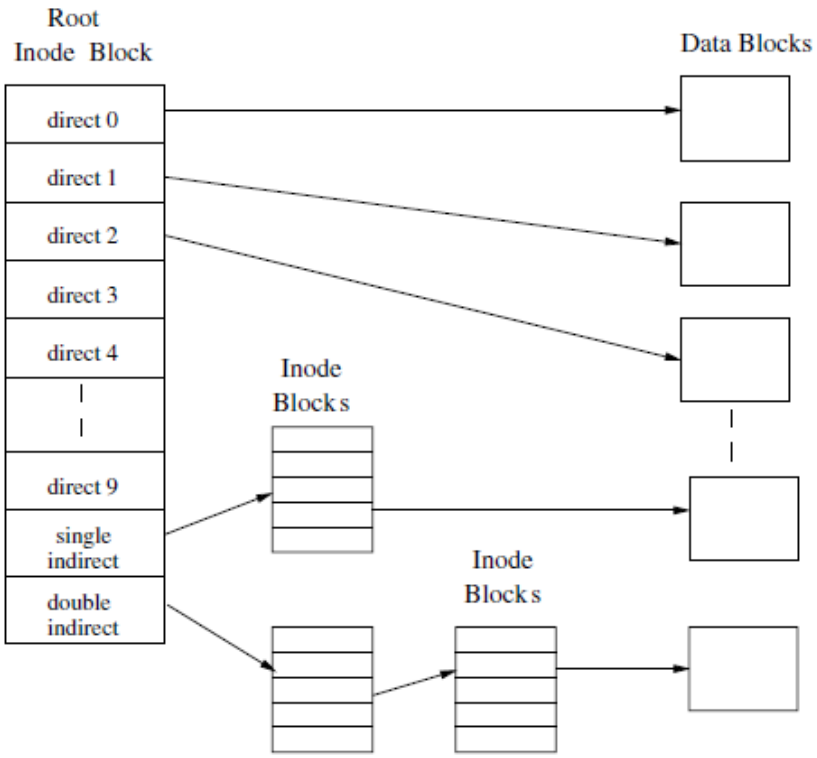


Figure: Filesystem layout

Background

UNIX Inode structure



Direct and Indirect Inode Blocks

Design and Implementation

Experimental Apparatus to evaluate the feasibility and performance of a hardware filesystem core .

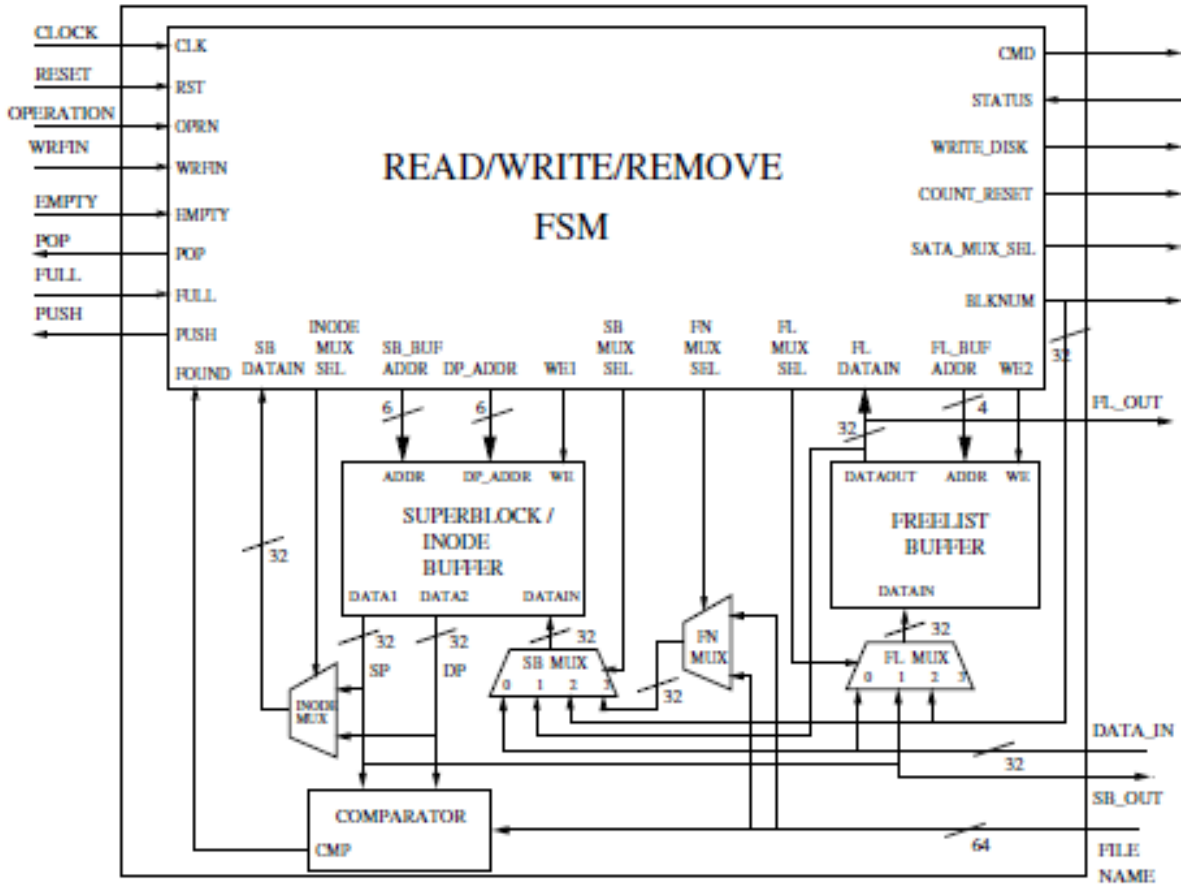
- **Behavioral model of a SATA host controller**
- **Hardware filesystem (HWFS Core)**
- **Testbench to exercise the hardware filesystem**

SATA Host Controller

- SATA host controllers core is expensive.Hence create a behavioral simulation of the SATA host controller.
- **mkafs.vhd** is run in a simulator to create an empty filesystem which is written to a local disk file on the workstation(**virtdisk.bin**).This includes the *SuperBlock* metadata and an interleaved linked freelist of all the data blocks.
- *Superblock is* 4 blocks wide and can store up to 15 files.
- During simulation, the SATA stub opens the empty disk file virtdisk.bin and loads it into a 2D array in memory.
- Depending on the command and block number received from the controller state machine, disk blocks are copied to and from read and write buffers

HWFS Core

Hardware File System: State Machine and Components



HWFS Core

The Hardware FileSystem (HWFS) core consists of

- Controlling State Machine
- Datapath
    i. Two Block RAMs
    ii. A comparator
    iii. Three 32-bit 4:1 multiplexers,
    iv. Two 32-bit 2:1 multiplexers,
    v. FIFOs

HWFS Core: State Machine

- The state machine implements the Open, Read, Write
- and Remove file operations.
- The 2-bit operation port is driven by a testbench to select from the four operations.
- The FSM issues an internal command signal and a 4-byte blockid to read from or write to the appropriate location in the satastub array.
- The satastub sends out the block 4-bytes at a time through portout.
- After completing a block transaction, it asserts the status signal. WriteDisk signals satastub to write the entire 2D array to disk file.

Testbench to exercise the hardware filesystem

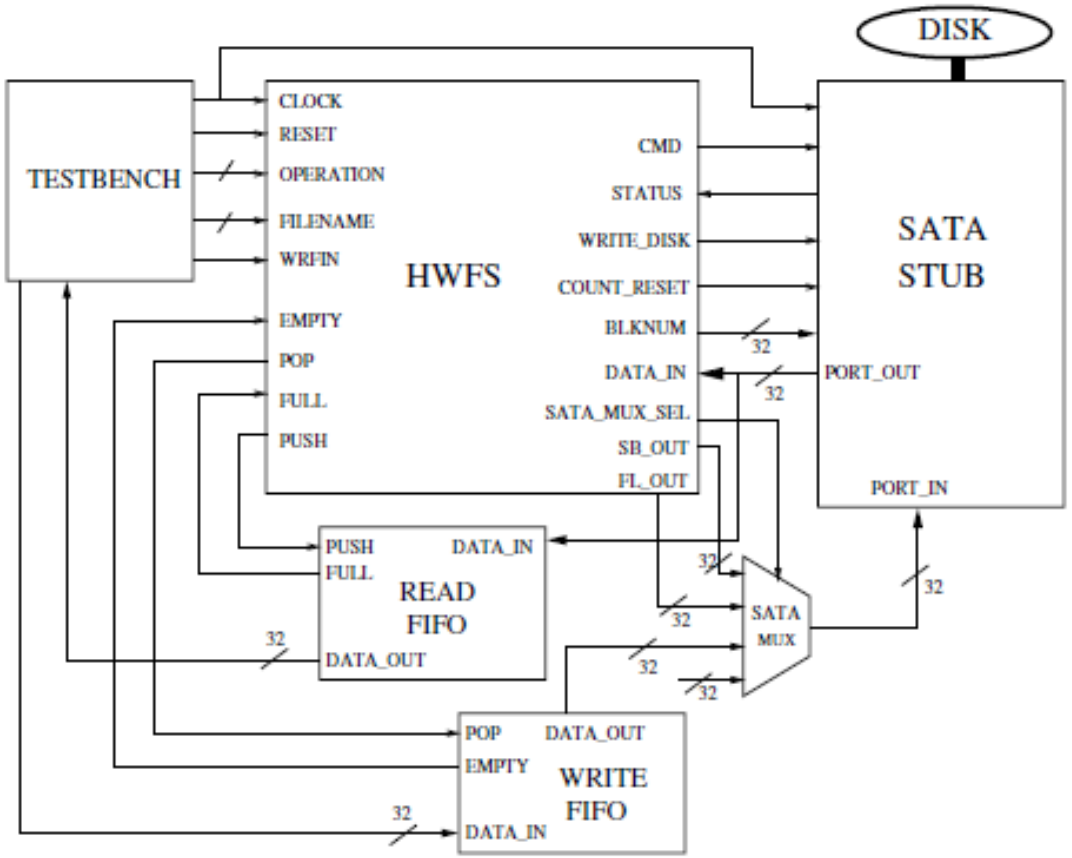
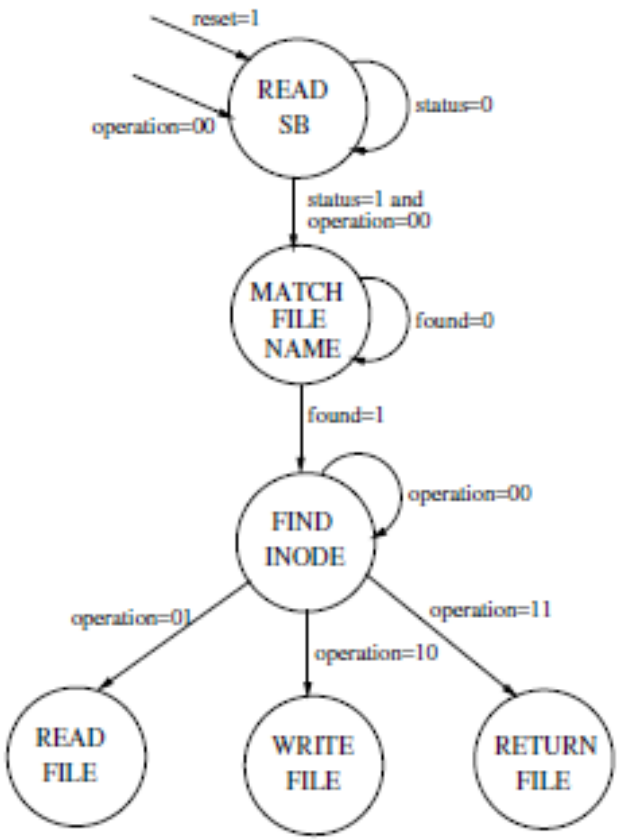


Figure: Experimental Setup

Example Operations : Open File



- State machine starts from the Read Super Block state ,reads blocks 0-3 from satastub into the SuperBlock buffer.

- Match Filename State performs linear search for the 8 byte filename

- If filename found, FSM which transitions to the Find Inode state.

- If the search is unsuccessful, filenotfound signal is asserted.

Figure: Open File State Machine

Testbench to exercise the hardware filesystem

- ModelSim and Xilinx ISE for simulation and synthesis purposes. Block RAMs were chosen in the design for the superblock/inode buffer and freelist buffer. VHDL testbench file instantiates the top level structural VHDL module of the design.
- Input test sequence includes a 64-bit filename for opening the required file from the disk and a 2-bit operation signal to select either: open, read, write and remove.
- On completing the read/write/remove file operations the state machine transitions to the idle state and asserts the stop-simulation signal.
- The testbench checks for this signal and reports a ”Testbench Successful” message alongwith the iteration time.
- ModelSim verification environment, version 6.3b, running on a Linux Workstation was used for simulation and debugging

Results and Analysis

41

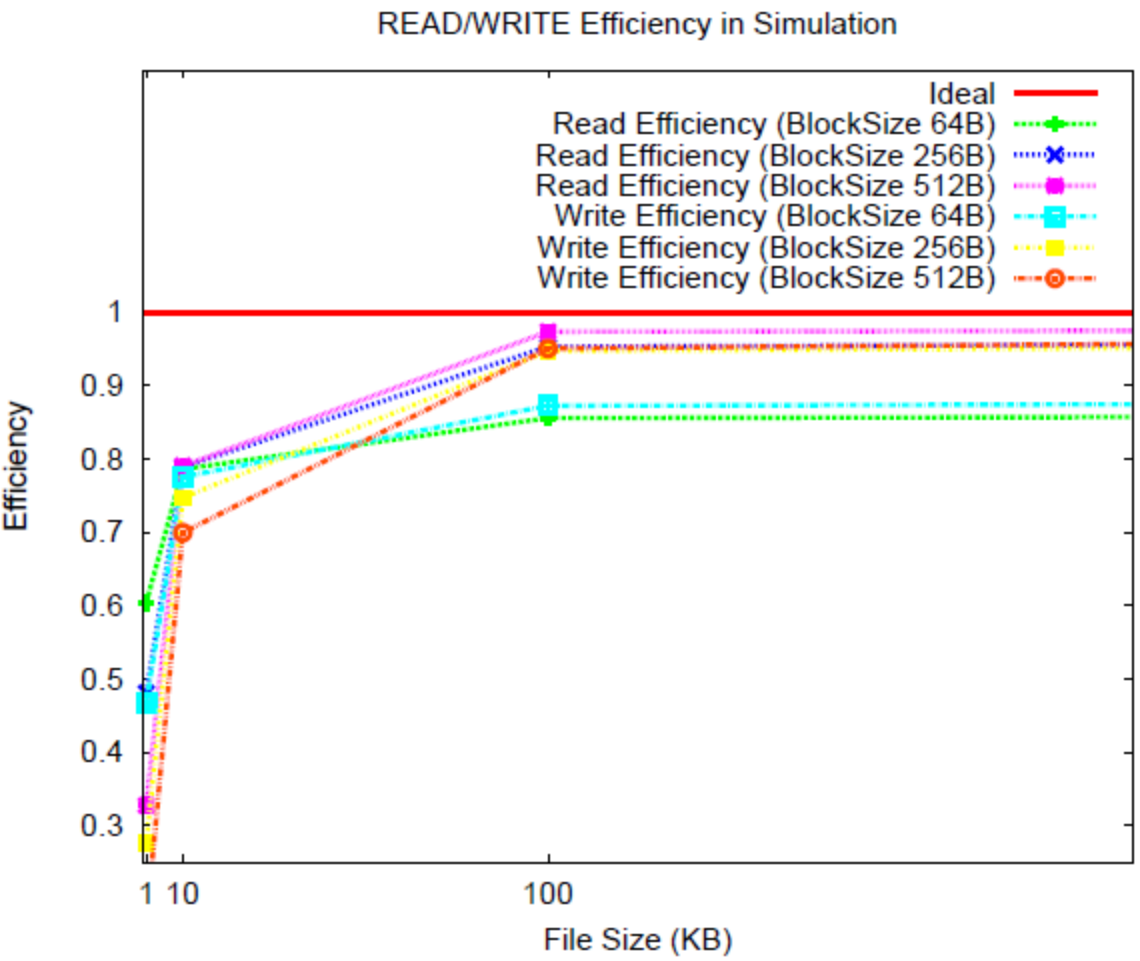
Table 4.1: HWFS Read/Write Execution Time with single RAM Disk

	Read			Write		
File Size (Bytes)	**64 B**	**512 B**	**1024 B**	**64 B**	**512 B**	**1024 B**
1 KB	9.28 μs	12.54 μs	19.62 μs	9.4 μs	28.3 μs	51.77 μs
10 KB	73.59 μs	45.8 μs	51.28 μs	52.3 μs	59.55 μs	83.02 μs
100 KB	709.84 μs	380.97 μs	366.32 μs	483 μs	391.56 μs	396.65 μs
1 MB	7.18 ms	3.76 ms	3.55 ms	4.9 ms	3.57 ms	3.54 ms
10 MB	71.8 ms	37.44 ms	35.35 ms	48.97 ms	35.48 ms	34.82 ms
100 MB	717.93 ms	374.33 ms	353.32 ms	489.65 ms	354.53 ms	347.69 ms

Table 4.2: HWFS Execution time for a 1 KB file, 64B block size

Operation	**Total**	**HWFS**	**RAMs**
Write	9.29 μs	5.54 μs	3.75 μs
Read	9.16 μs	4.32 μs	4.84 μs
Delete	5.27 μs	2.66 μs	2.61 μs

Results and Analysis



$$\text{eff} = \frac{\text{raw block transfer time}}{\text{filesystem block transfer time}}$$

$$\text{raw block transfer time} = \frac{\text{file size x clock cycle time}}{\text{bytes per clock cycle}}$$

Table 1. Statistics for HWFS resource utilization with different block sizes

Block Size	Slices	LUTs	F/Fs	BRAMs
64 B	759	1471	343	2
128 B	724	1369	345	2
256 B	749	1446	349	2
512 B	783	1502	353	2
1024 B	762	1463	356	3
4096 B	779	1476	364	10

Figure:HWFS Sequential Read/Write efficiency in simulation

Conclusion

- It enables the cores to get direct, high bandwidth access to large data sets.
- The design, correctly implements the four basic filesystem operations: open, read, write and remove.
- The design currently provides sequential access to a file. It can be enhanced to incorporate random reads and writes by implementing the lseek operation.
- Once the SATA IP core is purchased, the filesystem can be interfaced to it for measuring the actual File I/O performance.
- The hardware file system can then be replicated on each node of the 64-node cluster. This is important first step to develop and evaluate a parallel filesystem

Conclusion

- It enables the cores to get direct, high bandwidth access to large data sets.
- The design, correctly implements the four basic filesystem operations: open, read, write and remove.
- The design currently provides sequential access to a file. It can be enhanced to incorporate random reads and writes by implementing the lseek operation.
- Once the SATA IP core is purchased, the filesystem can be interfaced to it for measuring the actual File I/O performance.
- The hardware file system can then be replicated on each node of the 64-node cluster. This is important first step to develop and evaluate a parallel filesystem

References

Mendon, A.A.; Sass, R., "A Hardware Filesystem Implementation for High-Speed Secondary Storage," *Reconfigurable Computing and FPGAs, 2008. ReConFig '08. International Conference on* , vol., no., pp.283,288, 3-5 Dec. 2008

A. A. Mendon. Design and implementation of a hardware filesystem. Master's thesis, University of North Carolina at Charlotte, Aug. 2008.

J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers,* Inc., San Francisco, California, 1996.

http://www.cyberciti.biz/tips/understanding-unixlinux-filesystem-superblock.html