FRP: A Nonvolatile Memory Research Platform Targeting NAND Flash

John D. Davis and Lintao Zhang
Microsoft Research - Silicon Valley Lab
[joda,lintaoz]@microsoft.com

Abstract
In this paper, we introduce the Flash Research Platform (FRP), a non-volatile memory research platform that targets Solid State Storage (SSS) and NAND Flash research. We are developing the FRP because we believe that solid state storage devices such as Solid State Disks (SSDs) will be a ubiquitous persistent storage medium within 5 years for personal computers and servers. Solid state technologies present great challenges and opportunities to revolutionize storage system architecture. This architectural change will affect the entire application stack of the computation platform and could enable new applications at a radically lower infrastructure cost than currently possible.

Various Flash-based storage devices have been proposed. However, most of the existing and proposed solutions still regard Flash (and other solid state storage devices such as Phase Change Memory (PCM) and Magnetoresistive Random Access Memory (MRAM)) as a way to build faster hard disk drive equivalents (i.e., SSDs). Although this is a valid and sufficient approach for the short term, there are many more options in the design space and many open questions need to be answered to best exploit these new technologies in the longer term.

To help answer these questions, we need a flexible platform for experimentation. We present a Flash Research Platform for SSS research. FRP leverages reconfigurable hardware to provide maximum flexibility for innovative architectural and algorithmic design of the next generation storage systems.

1. INTRODUCTION

NAND Flash in the form of Solid State Disks (SSDs) is being presented as the next generation persistent storage solution, supplanting traditional spinning media or hard disk drives (HDDs). NAND Flash has many advantages that make it a very attractive replacement option: fast random reads, shock resistant, lower power, and better reliability. However, there are also several drawbacks that require careful system design at both the hardware and software level, such as requiring erasing before programming (writing), difference between erase and program granularity, limited rewrite cycles, and complex failure modes that require non trivial ECC. Historically, NAND Flash has been used in less demanding devices like digital cameras and MP3 players. More recently, SSDs based on NAND Flash have been adopted for more demanding workloads such as laptops, desktops, and server platforms. These applications have very different characteristics and require much better performance and higher reliability.

For compatibility reasons, current SSDs export an HDD interface. The firmware in an SSD provides a Flash Translation Layer (FTL) to abstract away the Flash devices in order to provide the HDD abstraction and hide certain operations (such as erasing) from the operating system. This could potentially limit the file system from extracting the full performance from the storage device.

On the other hand, traditional computer software has mostly been designed and optimized for HDDs. It is conceivable that many of the design choices may need reevaluation in order to fully utilize solid state storage devices. Due to the vastly different characteristics of Flash and other Solid State Storage (SSS) devices compared to traditional HDDs, it is worthwhile to rethink the entire storage stack with a fresh mindset. Currently available commodity devices do not provide enough flexibility/visibility for innovation due to the proprietary nature of the design.

The Flash Research Platform (FRP) is an open platform that leverages reconfigurable hardware to enable SSS research. Coupled with host drivers and software, it facilitates the research on the entire storage system. As shown in Figure 1, the FRP uses the BEE3 system [3] with custom designed Flash DIMM (FDIMM) modules as the underlying reconfigurable hardware. The accompanying host software enables driver and application development. This full control over the combination of hardware and software enables the ability to implement ideas in the entire storage stack.

The FRP provides the facility to investigate SSD related issues, Flash device specific issues, or broader memory hierarchy issues. The specific contributions of this work are:

- An open non-volatile memory research platform.
- The FRP currently targets NAND Flash devices, but has the flexibility to target other non-volatile memories.

![Figure 1. Flash Research Platform hardware and software stack.](image)
- The ability to investigate all aspects of SSD design in hardware and software. The design provides the ability to implement components in either hardware or software, and provides a path to migrate components between them.
- A flexible hardware architecture that provides individual control of each Flash chip, independently, thus enabling a variety of hardware architecture and controller designs.
- A host software and driver framework that enables easy SSD prototyping.

The rest of the paper is organized as follows. Section 2 provides background on SSDs and discusses related work. In Section 3, we discuss the hardware implementation of the Flash Research Platform (FRP) with particular focus on the NAND Flash implementation. Section 4 provides details about our firmware and software for the current system. Section 5 reports the current system status and discusses the FRP research scope. Finally, Section 6 concludes the paper and provides future work.

2. BACKGROUND AND RELATED WORK

The generic SSD architecture is shown in Figure 2. It contains multiple Flash chips for the storage, some dynamic RAM for storing working data set, and a communication channel to the host computer. It also contains a management unit, usually consisting of an embedded CPU running custom software, to perform various management operations. Currently, most vendors treat the architectures of their SSD designs and the algorithms used for managing Flash devices as trade secrets, and very little information on how they work is publicly available.

One of the main functionalities an SSD controller provides is a Flash Translation Layer (FTL) that makes the Flash device look like a disk. As shown by various literature ([20] [4] [13]), by using different architecture and management algorithms, SSDs have drastically different performance characteristics. Several previous works have used experimentation or trace-based software simulators to infer SSD behavior and test their algorithms and designs [4][2], but the results are not verified in practice.

There is a large amount of research on the FTL algorithms as well as on other aspects of an SSD design (see a survey in [10]). Most of this literature uses software simulation to verify the techniques proposed. The lack of a flexible hardware research and prototyping platform may be one of the most significant reasons that these innovative designs are not realized in actual hardware. The impact of these designs on actual application performance is thus not easy to evaluate.

We believe that a flexible prototyping environment can greatly accelerate the adoption and innovation in the solid state storage research space. The FRP provides an open framework in hardware and software to implement SSD architectures, interfaces, and algorithms. In particular, the FRP could be used to validate the SSD simulator created to investigate Flash array organization and management [2]. It can also be used to implement new mechanisms such as a transactional interface on top of SSDs [17].

3. FRP HARDWARE

The FRP hardware is a combination of the BEE3 multi-FPGA research platform and a custom printed circuit board, a Flash Dual Inline Memory Module (FDIMM). The BEE3 was originally designed and built for computer architecture research for the RAMP consortium [1]. Because the BEE3 is a reconfigurable platform, it can also be repurposed for SSD and Flash memory research. The BEE3 has 16 DIMM slots that we can populate with either DRAM or FDIMMs. The DIMM slot also provides the ability to interface to other non-volatile memories for future research, leveraging the same BEE3 hardware and software. In the rest of this section, we describe the BEE3 and the FDIMM in more detail. We will also discuss the possibility of using other non-volatile memory alternatives.

3.1. BEE3

The BEE3 system is composed of two separate PCBs and we focus only on the main PCB for brevity. The BEE3 is the third generation Berkeley Emulation Engine and borrows much of its design from the BEE2 system [4]. The main BEE3 PCB has four Virtex 5 FPGAs. They can be populated with a choice of several FPGAs with different capacity and features including LX110T, LX155T, SX95T, FX70T, and FX100T.

Each FPGA has two DDR2 DRAM channels with two DIMMs per channel and supports up to 64 GB for the system. The FPGA multi-gigabit transceivers
provide two CX4 interfaces, which can implement two 10 Gigabit (10 GbE) XAUI interface. Each FPGA has one eight lane PCI-Express (PCI-E x8) interface that supplies endpoint functionality. Each FPGA also has an embedded 1 Gigabit Ethernet (1 GbE) MAC hard macro that is coupled to a Broadcom PHY chip. The 10 GbE, PCI-E x8, or 1 GbE interfaces provide a variety of ways to interconnect BEE3 systems and host computers. In addition to the host attached SSD, the rich interconnects of BEE3 enable research on Flash-based network attached storage systems. Figure 3 provides the overall high-level BEE3 system block diagram and interfaces with approximate pin bus widths for the main BEE3 PCB [3].

3.2. FDIMM and FRP Hardware Architecture

By leveraging the BEE3 system, we are taking advantage of the FPGA’s reconﬁguration ability and using a standard DIMM form factor for the Flash daughter card. The FDIMM is built using 4GB SLC Samsung NAND Flash packages (Writes: 20 MB/s, Reads: 40 MB/s) [19]. The FDIMM ﬁts in a standard 240-pin DDR2 DIMM slot and is slightly taller than a normal DIMM. Each FDIMM exports approximately 130 pins for the data and control interface to the DIMM slot connector, enough for eight Flash chips to be independently controlled by the FPGA.

The DDR2 I/O voltage interface is 1.8V, whereas the Flash component I/O voltage interface is 3.3V. We use a Xilinx CPLD as a voltage translator. The CPLD implements a synchronous pass-through device that could also provide some minimal additional logic or performance counters. The Flash we are using has a low operating frequency that is well suited for the CPLD. For higher speed devices such as the new generation ONFI2 Flash parts ([29]), other types of level changing devices may be needed. Figure 4 shows the front and back of the FDIMM hardware module and connectivity.

In the current configuration, an FDIMM has an aggregate write bandwidth of approximately 160 MB/s and aggregate read bandwidth of approximately 320 MB/s. This is on par with existing high-end SSDs [13]. The FRP can easily saturate the 2GB/s BEE3 PCI-E x8 interface when it is fully populated with 14 FDIMMs; the aggregate write bandwidth is 2,240 MB/s and the aggregate read bandwidth is 4,480 MB/s.

As shown in a Figure 5A, a small conﬁguration of the FRP uses a single FPGA on the BEE3 and can accommodate up to 8GB RAM and 64GB Flash. Using all 4 FGPA’s, the maximum system capacity would be 8 GB of DRAM with 448 GB of NAND Flash (14 FDIMMs), as illustrated in Figure 5B. The system can further increase the capacity by using different DIMM designs, such as dual rank FDIMMs and/or MLC NAND Flash devices. Between these two system conﬁguration points, other hybrid DRAM and Flash or Flash-only conﬁgurations are possible.

We implemented the most ﬂexible FDIMM design using independent point-to-point control and data interconnect between the Flash packages (FP) and the FPGA as shown in Figure 6A. However, other Flash conﬁgurations are possible that require fewer I/O pins. For example, the majority of the control signals (labeled S-Cntrl) can be shared and only the chip enable (CE) signals require distribution or all control pins can be shared, as shown in Figures 6B and 6C, respectively. This enables one or more FPs to operate
in concert or forces all the FPs to complete the same command and increases the read and write block size. It is also possible to share the data pins by time division multiplexing the data bus with individual chip or rank enables.

3.3. Using Other Non-Volatile Memories

By using the BEE3 reconfigurable hardware platform, we can adapt it for other non-volatile memory research. We can easily switch between SLC and MLC NAND by changing the type of device that populates the FDIMM. Furthermore, different DIMMs can be manufactured for investigating NOR Flash devices. This would be similar to the Spansion EcoRAM components that are currently available [9]. There are several non-volatile memories that are on the verge of being commercially available. They currently are not yet competitive with Flash devices but have a lot of potential to become popular in the future. A reconfigurable platform is an ideal research vehicle for prototyping a storage system that use such devices. This includes Magnetoresistive Random Access Memory (MRAM) [8], Silicon-Oxide-Nitride-Oxide-Silicon (SONOS) [6], Phase Change Memory (PCM, or PRAM) [16], Nano-RAM (NRAM) [15] and programmable metallization cell, or (PMC) [14].

4. FRP SOFTWARE

The FRP design provides a flexible substrate for innovation in gateware, firmware, drivers, and applications that utilize the underlying storage media. For the FRP, there are three layers of software to develop, as shown in Figure 7A: gateware and firmware, device drivers, and applications.

Traditionally, code running on the hardware, generally stored in read-only memory, is often called firmware while code running on the host machine is called software. The gateware refers to the Hardware Description Language (HDL), typically Verilog or VHDL, used to program the FPGAs. The FRP provides the unique opportunity to move components from software to firmware run by an FPGA controller, or even to specialized hardware in the form of gateware, as shown by the grey boxes in the various layers in Figure 7B.

4.1. FPGA Gateware layer

The FPGA gateware is the code that programs the FPGA, transforming the FPGA into an instance of the specified hardware design. We use Xilinx’s software development environment (ISE 10.1 and EDK 10.1) to compile Verilog or VHDL into a bitstream that is used to program the FPGAs. The FPGA can be thought of as two distinct planes, a control plane and a sea of resources plane (Block RAMs, DSPs, 6-input look-up tables, and an on-chip interconnect network). The bitstream programs the control plane. This in turn specifies the connectivity and operation of the sea of resources plane. The result is the ability to integrate various digital systems and build complex systems, like the FRP.

The FRP integrates various components: DRAM memory controller, I/O interfaces (1GbE, 10 GbE, or PCI-E x8), Flash memory controllers, and embedded CPU, all the components in the lowest layer of Figure 7A. We leverage pre-existing gateware for the DDR2 DRAM controller [7]. The I/O modules for the 1GbE, PCI-E x8, and 10 GbE are based on application notes and the related sample designs and building blocks available from Xilinx [25]-[28]. The PCI-E interface provides DMA functionality, which frees the host machine from managing, at a fine granularity, data movement in the FRP system.

There are many CPU cores available for FPGAs. Xilinx provides its own 32-bit Microblaze processor that integrates well with other Xilinx modules and supports full OS functionality [24]. Xilinx Virtex 5 FXT devices also contain PowerPC hard IP cores that can provide higher performance than soft cores such as the Microblaze [23].
In addition to these initial components, there are several components that have been developed internally. The Flash package controller implements many of the Flash commands (read, program, erase, etc.) outlined in the datasheet [19]. We dedicate a simple Flash controller to each Flash channel. We designed this Flash controller as a finite state machine (FSM), but the Flash command sequences could also be implemented using a simple microcontroller and associated program. This may sounds like overkill, but it provides the maximum flexibility for quick prototyping. Likewise, there are several interface modules and queues required for data management that are not shown in Figure 7.

The FRP also facilitates the development of additional gateware for specialized compute engines or functions that operate on the data retrieved or the data to be written to the Flash packages. For example, error correction code (ECC) logic can be implemented in gateware to offload the computation from the host CPU. The ECC implementation can be flexible in that varying strengths of ECC could be implemented to recover from varying number of bit errors. It is also possible to implement other capabilities in the gateware such as encryption/decryption, compression, data filtering, and data manipulation.

4.2. Device Driver

The device driver is the OS dependent layer that interacts with the user applications and the FRP hardware. The device driver must be executed on the host, usually in the OS kernel space, although user-level driver development is also possible.

Traditionally, the device driver for SSDs provides a disk-like interface (a block device) to the applications. To be consistent with SSDs, our device driver exports a HDD interface, as shown in Figure 7. Thus, the FRP emulates HDD functionality and applications are unaware of the difference, making it compatible with all software.

It is also possible for the device driver to expose special APIs that are not traditionally provided by hard disk interfaces. One can augment the hard disk with operations such as compression and encryption. Moreover, there are many proposals that provide Flash specific operations, such as marking a logical sector as free (called TRIM operation) [21] or physically deleting the data of a logical sector (secure delete) [9]. Recently, transactional semantics have been proposed to augment the Flash-based disks [17].

The application interface is only one part of the FRP device driver layer. The external I/O device driver is also required to facilitate communication between the host CPU and BEE3 module. We leverage existing network device drivers for the communication with Ethernet, and create our own PCI-E device driver for performance reasons.

We have also developed the system shown in Figure 7C, which enables driver development in user-level applications, speeding up development time and improving the driver debugging experience. The user-level application sees the same block device interface as before. That block device driver makes a callback to
a user-level application that can implement various commands and data manipulations and then interface to a communication driver to interface to the FPGA through 1 GbE, 10 GbE, or PCI-E. The split driver implementation is especially useful for quickly implementing new functionalities in the Flash management layer.

4.3. The FRP Management Architecture

The FRP management software specifies the hardware-software contract for the FRP. The management layer software abstracts away the details of the Flash devices operations, such as block erase, and provides a well defined hardware independent interface for read/write/flush or APIs like trim and secure erase.

The FRP management software implements various algorithms used in the Flash Translation Layer (FTL) such as cache management, garbage collection, wear leveling and logical to physical sector remapping [10]. The FRP management software translates various tasks into the gateware layer commands and monitors the execution of these commands on the actual hardware. As Figure 7B illustrates, the FRP management software can be executed on an embedded CPU in the FPGA, or on the host CPU in the device driver or even as a user-level application. Currently, we organize the management layer as a user-level application similar to Figure 7C, trading off performance for ease of implementation.

Figure 8 shows the major components of FRP’s management software architecture. It contains several data structures collectively reflecting the state of the FRP system. These include a map that keeps track of the logical to physical page translations, a buffer manager for cache management, a block manager to keep track of the status of all the blocks (recall that a block is the smallest unit of erase), a log manager that transform random writes into log-structured writes, and a metadata manager that keep track of other system metadata.

The task manager of the management layer executes an infinite loop, leveraging the information available from the data structures to carry out user requests such as reads and writes. Additional tasks, such as wear leveling and garbage collection, as well as background data scrubbing, are also being scheduled by the task manager. It is possible to choose various algorithms for certain components, such as the cache replacement policy in the buffer manager or the mapping technique used in logical to physical mapping. However, we think the overall architecture of the management layer is flexible to accommodate many of these variations.

5. FRP DISCUSSION

The Flash research platform is early in its development and we believe it will enables many research directions. First, we describe the FRP status and then discuss the research scope of the system.

5.1. FRP Status

The FRP hardware has been manufactured, tested, and is fully functional and is shown in Figure 9. As mentioned before, the current FDIMM has 8 Samsung SLC NAND Flash chips and all the chips can be independently controlled by the FPGA. We are exploring other FDIMM designs including using devices that have the next generation Flash interfaces such as ONFI2 [29] and Toggle-Mode.

The FRP software is still under active development. We have developed the first instance of the FRP, a single FPGA system as shown in Figure 5A. We have implemented the gateware for this configuration using 1 GbE as the communication channel between FRP and the host. As a first step, we demonstrated the ability to read/write and erase individual Flash chips. We also implemented the management layer software on the host computer and we are in the process of linking the management software with the actual Flash hardware.

We are also developing the gateware for the PCI-E communication channel. Initial experiments show that we can achieve over 1 GB/s throughput for our actual implementation (compared with theoretical bandwidth of 2 GB/s). The PCI-E controller under development contains scatter-gather DMA capability. This will greatly improve system bandwidth and latency. Currently, we do not have an embedded CPU in the FPGA. All management functions are carried out in the device driver layer. We plan to implement a CPU on the FPGA to offload the management layer in the future.

We are still developing the Windows device driver layer. We separated the device driver into two parts as shown in Figure 7C for ease of development. This will certainly incur performance overhead. In the future, when the software becomes stable, we plan to use a
unified device driver similar to Figure 7B. The current device driver makes the FRP look like a hard disk to the user applications. We expect to expand the interface to include richer functionalities.

Finally, the application layer software received the least amount of development. However, we believe that this may actually provide the most fertile ground for innovation when the FRP becomes stable.

5.2. FRP Research Scope
The main benefits of the FRP are the increased system visibility and real prototype interaction. The FRP provides the ability to see into the hardware, firmware, device driver, and application layers. This is something no other SSD provides and enables both in depth investigation and understanding opposed to research that derives results based on inference and deduction. By implementing the entire stack, we will have complete control over the device management in all layers and not just interact with the exposed interface as with an SSD or other black-box systems.

The FRP will enable a wide range of monitoring and logging. Hardware counters of various types can be added to the FPGA and streamed out over RS232 or other I/O channels. The device driver can be instrumented as well as applications. Within the FPGA, performance counters can monitor read and write cache utilization and performance, hot pages or most frequently accessed pages, off-load engine performance, and bit error tracking, to name a few. These performance counters can be implemented as needed. The FDIMM design also allows for limited power measurement of the Flash packages. The Flash packages use an isolated voltage plane that also supplies the I/O voltage for the PLD. Controller or full system power would still need to be extrapolated for a target storage system.

Given the FRP flexibility and visibility, we will be able to investigate a wide range of topics. These topics span low-level issues like characterizing the NAND Flash devices and observing and understanding device failure to broader questions related to how NAND Flash fits in the traditional memory hierarchy. By building a system that mimics an SSD, we are developing the framework to investigate NAND Flash and non-volatile memory in general.

We built the FRP to interact with real systems. However, FRP system performance has been initially sacrificed for rapid bring up. By implementing most of the system functionality in software, the overall system performance will have many opportunities for optimization by moving functionality down into hardware. As a result, development and debugging may occur at very different levels depending on the user’s proficiency.

The main FRP system limitations are related to the overall system performance, limited power monitoring, and system cost. The base BEE3 system is expensive as well as the individual FDIMM modules. Currently, system performance is limited by developer effort. As mentioned previously, full system power measurements using the entire BEE3 system are not representative of an SSD, for instance. Likewise, the FRP has limited storage capacity using the FDIMMs and BEE3 DRAM slots and limited FPGA compute capacity. One can use a single BEE3 to build a system with terabytes of storage, but petabytes are currently beyond the realm of an FRP system or reasonable collection of systems. Finally, developing the FRP has required expertise in both hardware and software development and as one can imagine, initial development and the associated learning curve is nontrivial.

6. CONCLUSIONS AND FUTURE WORK
SSDs are going to revolutionize the storage architecture of computer systems. Due to the difference between SSDs and traditional storage medium, many well-established principles in our hardware and software design need to be re-examined. We have built a hardware platform, the FRP, for storage architecture research and in particular NAND Flash SSDs. We believe this hardware design is flexible enough to accommodate a wide variety of research. FRP is not limited to NAND Flash devices. We can leverage and reuse the FRP hardware and software for other non-volatile memories, like high-speed NAND Flash, NOR Flash, or PCM. These new solid state storage devices, though not as prevalent as NAND Flash, have the potential to have bigger impact in the longer term.
While we can leverage a lot of components from various sources, there is still significant effort required to integrate and interface to these components. Regardless, the goal of having an open research platform for SSD and NAND Flash research is in sight.

There is a significant amount of future work that can be pursued with the FRP. Our immediate plans are to create a stable FRP that can scale to use all the FPGAs in the BEE3. Following that milestone, we plan to implement transactional flash [17], as well as exploring other abstractions for solid state storage devices. We plan to investigate the possibility of migrating certain computations (such as encryption) to the disk controller hardware. We also plan to implement a utility to stress test Flash devices to learn more about the failure mode of these devices.

REFERENCES


