SOFTWARE-ENABLED FLASH™

Software-Enabled Flash™ Technology:

Understanding the Reference Flash Translation Layer in the SDK

TECHNICAL BRIEF

Software-Enabled Flash (SEF) technology is a combination of purpose-built hardware and an open source software stack (Figure 1) that provides developers with unprecedented application control over flash memory behaviors. Built for cloud-scale applications, it exposes the parallelism and digital nature of flash memory to applications while abstracting the low-level operations of flash memory. Tenant isolation, queueing control down to the individual Input/Output (I/O) operation, and copy offload are now possible within applications that use this technology.

The Software-Enabled Flash open source project, under The Linux Foundation®, is responsible for the host software stack that make this technology possible. It has already released and documented the Application Programming Interface (API) that serves as a wrapper around the low-level SEF device commands.

To make it even easier for applications to work with this new programming paradigm, the Software-Enabled Flash project is developing a Software Development Kit (SDK) that is intended to provide a higher-level set of application-focused features.

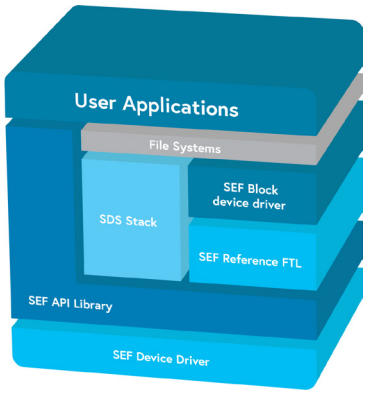

Figure 1: SEF Software Stack
(Used with permission from KIOXIA America, Inc.)

The SDK contains four major components:

1. Command-Line Interface (CLI) tool: *(used to manage the complete life cycle of a SEF device)*
2. Flexible I/O[1] (FIO) test tool: *(used to explore different isolation and latency configurations without requiring a ported application)*
3. Reference virtual I/O device driver: *(works with the open source QEMU[2] virtualization system enabling developers to use SEF technology in their virtualized applications without needing to change any application code)*
4. Reference Flash Translation Layer (FTL): *(provides a sample implementation of a standard block interface to applications running on a SEF device)*

The focus of this tech brief is on the reference FTL portion of the SDK.

The Reference FTL

The SDK provides an open source licensed reference FTL to accelerate user development of application-customized FTLs. Even though it is a reference codebase, it is a fully functional sample FTL. It stores application data on a user-created Quality of Service Domain (Figure 2), allowing all the isolation and queueing control of the raw SEF API. It provides such features as placement IDs and hardware offloaded garbage collection, as well as SEF features such as I/O queueing control and isolation. Developers can use this code to prototype their own applications and replace individual components as they are created.

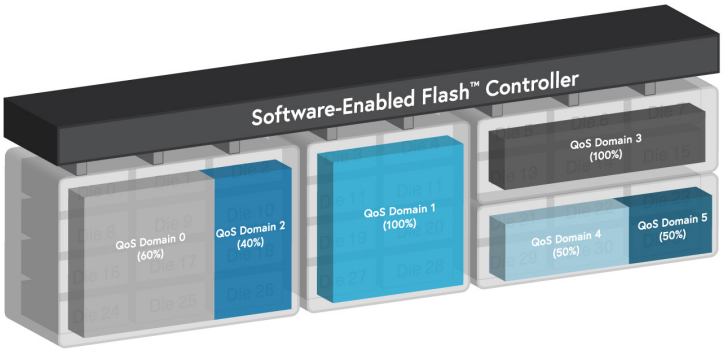

Figure 2: Example of SEF Quality of Service Domains configuration
(Used with permission from KIOXIA America, Inc.)

Documentation is provided in both the SDK header files as well as a separate PDF which includes the general theory of operation and explanations about how individual components work together. The guiding principles for the FTL include:

- Support all of the underlying SEF features such as isolation, queueing control and copy offload
- Protect data and ensure recovery after power loss
- Be written in straight C with well documented, modular code

Reference FTL Components

The reference FTL is constructed of multiple functional C-language units, each with their own defined API. This is intended to allow developers to easily replace individual portions while preserving the rest of the reference FTL stack. For example, a custom garbage collection algorithm could be implemented by a developer without requiring a new look up table or persistence layer.

The six components of the reference FTL include (Figure 3):

- SEF Block Layer
- Look Up Table
- Super Block Management
- Garbage Collection
- Metadata Persistence
- Instrumentation

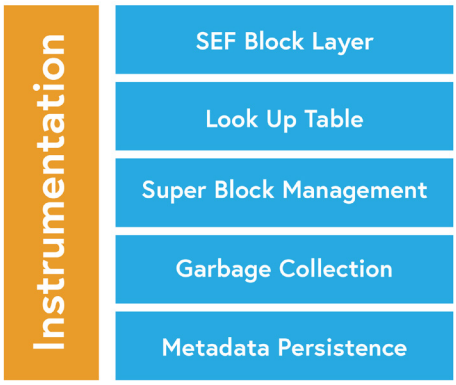

Figure 3: Reference FTL components
(Used with permission from KIOXIA America, Inc.)

More details covering each of these components now follows. For actual SDK interfaces, please refer to the SDK documentation and headers.

SEF Block Layer

The SEF Block Layer provides the interface between all applications and the reference FTL. This is an asynchronous-based API which allows multiple read, write or trim operations to occur. It takes a SEFMultiContext structure (Figure 4) that defines the I/O operation to perform and contains a callback to be executed on completion.

```
struct SEFMultiContext
{
 SEFBlockHandle blockHandle;
                 /**< SEF Block handle to be used for access to the block instance */
 struct SEFMultiContext *parent;
      /**< A pointer to an instance of SEFMultiContext used for compound operations */
 void (*completion)(struct SEFMultiContext *);
                /**< The function that is called when the transaction is completed */
 void *arg;               /**< A pointer that can be used by caller for any reason */
 uint64_t lba;                                          /**< Logical block address */
 uint32_t lbc;                                            /**< Logical block count */
 enum SEFBlockIOType ioType;           /**< The I/O Type that needs to be performed */
 uint8_t flags;                                  /**< I/O flags enum SEFBlockIOFlags */
 char reserved[2];
 struct iovec *iov;                       /**< A pointer to the scatter/gather list */
 int iovcnt;                /**< The number of elements in the scatter/gather list */
uint32_t iovOffset;                          /**< Starting byte offset into iov array */
struct SEFPlacementID placementID;                    /**< Placement ID for writes */
atomic_int transferred;
             /**< Counter denoting number of bytes transferred for the transaction */
 atomic_int count;                     /**< Reference count, I/O is completed -> 0 */
 atomic_int error;                            /**< First error for the transaction */
 int cancel;                                 /**< Set to indicate cancel in progress */
};
```

Figure 4: SEFMultiContext structure within the reference FTL Block Layer

Important Note: This SEFMultiContext I/O request structure uses the standard iovec structure[3] to support scatter-gather lists for reading or writing to host memory. It also supports placement IDs for write operations. The same structure is used for read, write and trim operations.

Look Up Table

The Look Up Table is a critical piece of any FTL and translates between Logical Block Addresses (LBAs) to the underlying flash Physical Addresses used by the SEF API. The full translation table for the Quality-of-Service Domain used is kept in host memory and requires 8 bytes for every 4,096 bytes of flash memory, or 2 gigabytes[4] (GB) of host RAM per terabyte[4] (TB) of flash memory. Developers can build their own look up table modules that, for example, only cache a portion of the table in host memory, or implements different/variable entry granularity.

Super Block Management

Super blocks[5] store all data in a Quality of Service Domain. While the SEF hardware unit is responsible for managing super blocks (such as ensuring even wear or allocating a defined virtual device), the FTL keeps a copy of all the super block states in host RAM. This capability enhances performance by avoiding a round-trip from the storage drive whenever a super block state needs to be accessed.

Super blocks in the FTL are tracked (Figure 5). These super block states correspond directly with those states in the SEF hardware unit. The internal representation allows the FTL to ensure the integrity of the on-flash data while multiple operations are underway. For example, a super block with read operations in flight will be marked in RAM so that it is not erased as a result of a garbage collection operation.

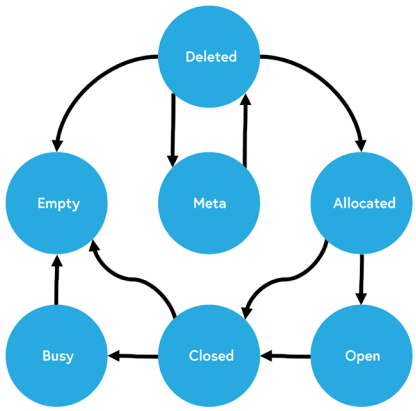

Figure 5: FTL super block states and transitions (Used with permission from KIOXIA America, Inc.)

Garbage Collection

Garbage Collection[6] (GC) is completely under host control with the reference FTL. It is normally run when flash memory is under write pressure, such as when the number of free super blocks that need to be written falls below a user-defined critical value (as is common with flash overprovisioning). However, under an explicit application request, GC can be initiated or deferred. The included GC module within the reference FTL runs asynchronously to the main FTL operation, and executes a pseudo-code loop (Figure 6):

```
while (still work to do)
{
 Get list of collectable superblocks (ones w/invalid data)
 Sort by # of invalid ADUs (~sectors)
 Determine placement id with most invalid data
 Allocate destination super blocks
 Send nameless copy bitmaps (from Super Block Tracking)
 Perform copy in-drive, no host CPU or DRAM or PCIe bus bandwidth
 Update Flash Translation with new mappings
 Discard read-out super blocks
}
```

Figure 6: Pseudo-code loop executed by the GC module in the reference FTL

The GC module determines the blocks that need to be moved, but also relies on the SEF copy offload capability to actually move the data. In this instance, neither the PCIe® bandwidth or DRAM bandwidth is used, nor is the host CPU involved other than to identify the blocks to move and to send a request to the SEF hardware unit.

Metadata Persistence

The Persistence module manages metadata persistence for the reference FTL. This includes data such as the look up table (and changes) and the state of different super blocks. It utilizes the standard SEF super blocks to store its data, along with a SEF feature called Root Pointers which enables the reference FTL to quickly find the latest copy of the metadata on start up.

Instrumentation

Since I/O operations to the SEF hardware unit do not travel the same path as those operations of standard block I/O, they cannot be tracked using existing system monitoring tools. To overcome this limitation, an instrumentation block permeates the entire reference FTL stack, and via this instrumentation layer, it is possible to track block I/O operations to the FTL, as well as lower level SEF operations such as super block allocations or physical address requests.

This Instrumentation layer can be dynamically enabled and disabled enabling developers to turn monitoring on and off as desired while an application runs. If monitoring is not required (i.e., in a fully-tested, deployed application), then no additional memory and CPU resources are used. When monitoring is enabled through a UNIX® socket, the Instrumentation layer can report data in human-readable

A Reference FTL Built for Developers

The SDK includes a reference FTL but as shown in this technical brief, "reference" does not mean "bare bones." It implements a fully functional FTL with support for all the SEF capabilities of the raw device. This brief only scratches the surface of the reference FTL. Since the code is fully open sourced, developers should dig into the header files and implementation details to better understand its operation as well as tuning it for specific application workloads.

For more information on the Software-Enabled Flash SDK and the entire Software-Enabled Flash technology ecosystem, go to https://softwareenabledflash.org where whitepapers, videos and infographics are available. Source code for the project is hosted on the GitHub® site - https://github.com/softwareenabledflash.

Notes:

[1] Flexible I/O (FIO) is a free and open source disk I/O tool used both for benchmark and stress/hardware verification. The software displays a variety of I/O performance results, including complete I/O latencies and percentiles.

[2] QEMU is an acronym for Quick Emulator and defined as a software module that supports full virtualization by providing emulation of various hardware devices. QEMU is a component of the hypervisor platform.

[3] The iovec structure defines one vector element. Normally, this structure is used as an array of multiple elements. For each transfer element, the pointer member iov_base points to a buffer that is receiving data for 'readv' or is transmitting data for 'writev'. The member iov_len in each case determines the maximum receive length and the actual write length, respectively.

[4] Definition of capacity – Software-Enabled Flash project contributing member, KIOXIA Corporation defines a kilobyte (KB) as 1,000 bytes, a megabyte (MB) as 1,000,000 bytes, a gigabyte (GB) as 1,000,000,000 bytes, a terabyte (TB) as 1,000,000,000,000 bytes, and a petabyte (PB) as 1,000,000,000,000,000 bytes. A computer operating system, however, reports storage capacity using powers of 2 for the definition of 1Gbit = 2^{30} bits = 1,073,741,824 bits, 1GB = 2^{30} bytes = 1,073,741,824 bytes, 1TB = 2^{40} bytes = 1,099,511,627,776 bytes, and 1PB = 2^{50} bytes = 1,125,899,906,842,624 bytes and therefore shows less storage capacity. Available storage capacity (including examples of various media files) will vary based on file size, formatting, settings, software and operating system, and/or pre-installed software applications, or media content. Actual formatted capacity may vary.

[5] A super block is a group of the same physical block addresses that are striped across multiple dies. Each die has thousands of blocks and a virtual device may have anywhere from one to hundreds of dies.

[6] Garbage collection (GC) is a memory recovery feature built into programming languages that automatically frees up memory space that has been allocated to data that is no longer needed by the application.