Plumbing SDXI into Linux: From DMA Engine to User-Space Offloads

Wei Huang

2025 Linux Plumber Conference

AMD
together we advance_

Overview

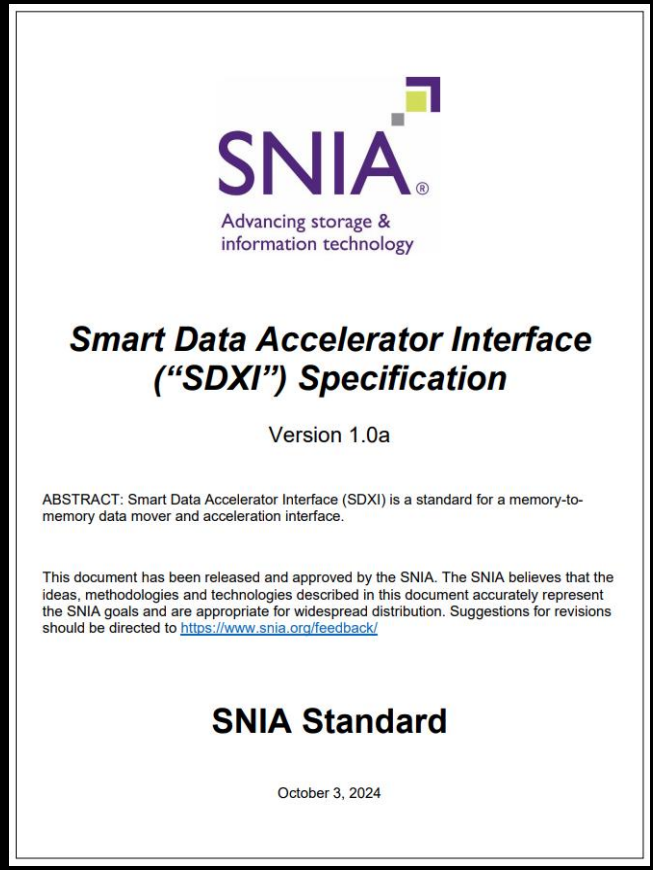
SDXI Primer
A vendor neutral, industry standard for offloading memory operations

Kernel Implementation
Two phases to support both kernel and user-mode offloading

Discussion
Next step and future design discussion

About SDXI

- SDXI is a vendor neutral standard for data operations.
  - Supports cross address space data operations without context switching
  - Developed with flexibility and future extensions
- Usage examples
  - Data center: Memory copying, zeroing, etc.
  - Networking: Data copying in packet processing pipelines
  - Virtualization: Migrate VM's physical memory across hosts



SNIA®
Advancing storage & information technology

Smart Data Accelerator Interface ("SDXI") Specification

Version 1.0a

ABSTRACT: Smart Data Accelerator Interface (SDXI) is a standard for a memory-to-memory data mover and acceleration interface.

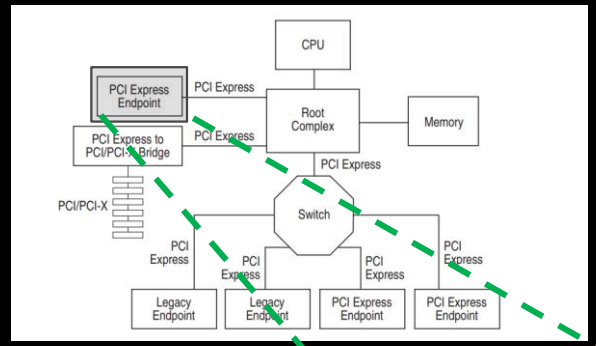
This document has been released and approved by the SNIA. The SNIA believes that the ideas, methodologies and technologies described in this document accurately represent the SNIA goals and are appropriate for widespread distribution. Suggestions for revisions should be directed to https://www.snia.org/feedback/

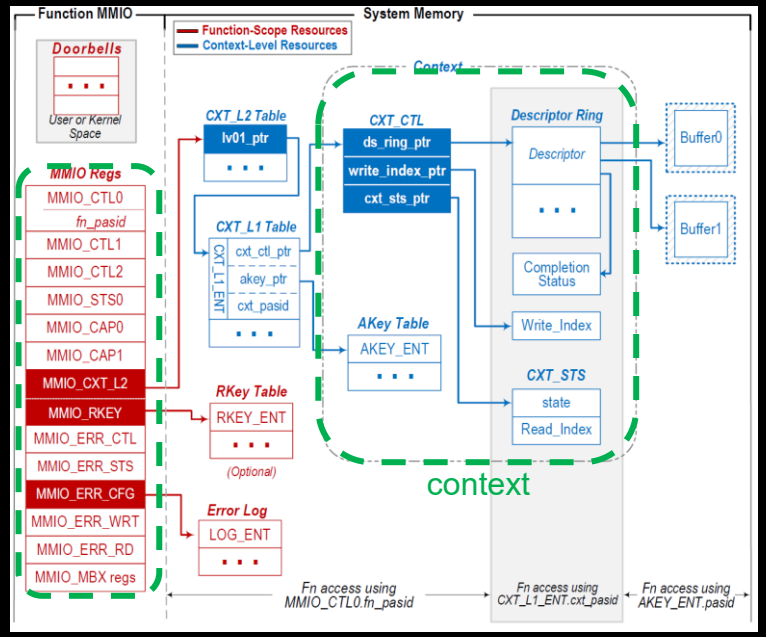
SNIA Standard

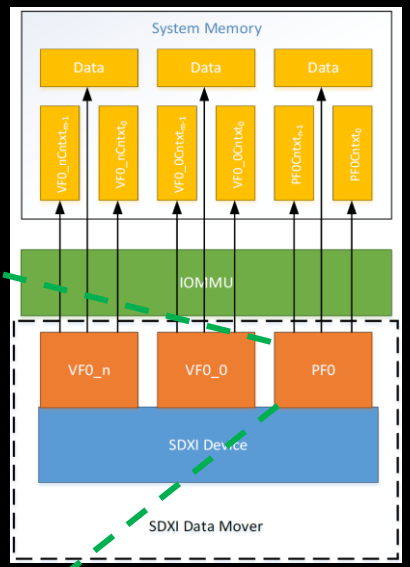
October 3, 2024

SDXI – Overview

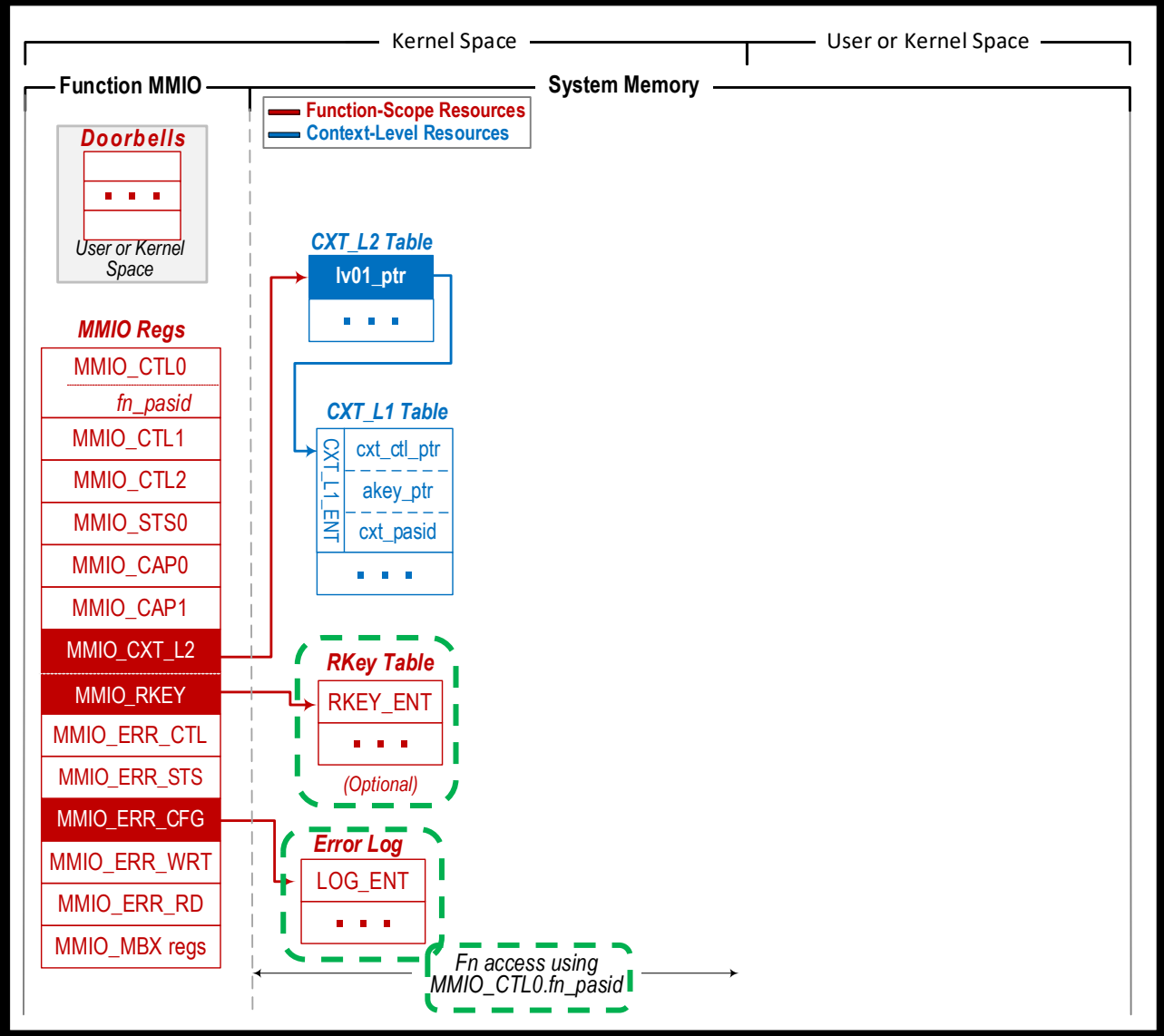
- Exposed as PCIe end-point device w/ SR-IOV support
- Each function operates as an independent entity
  - Capability and configure registers are accessible via MMIOs
- Each function supports multiple execution contexts
  - Context is the entity for command submission
- Can handle requests from different address spaces





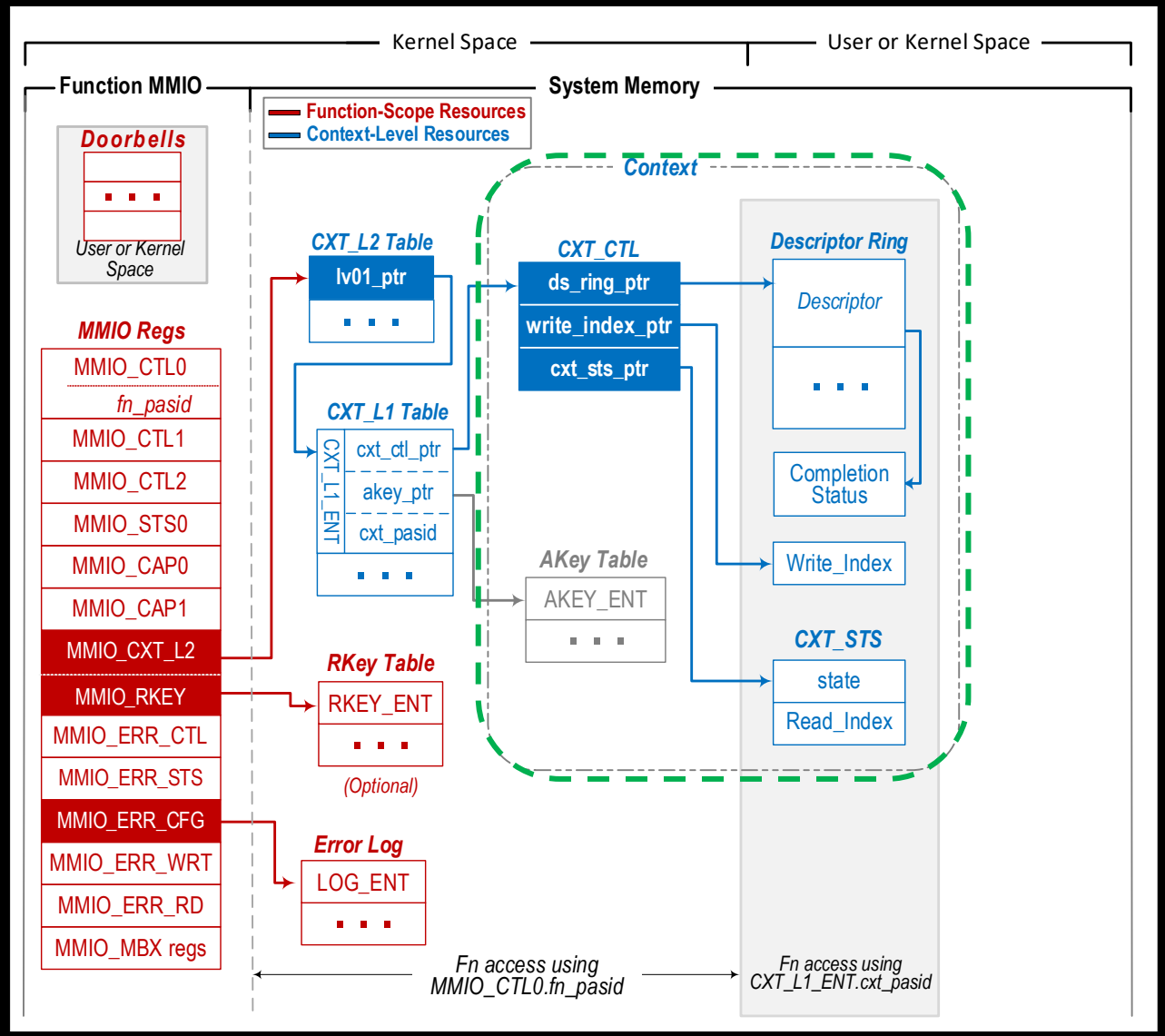


SDXI – Function



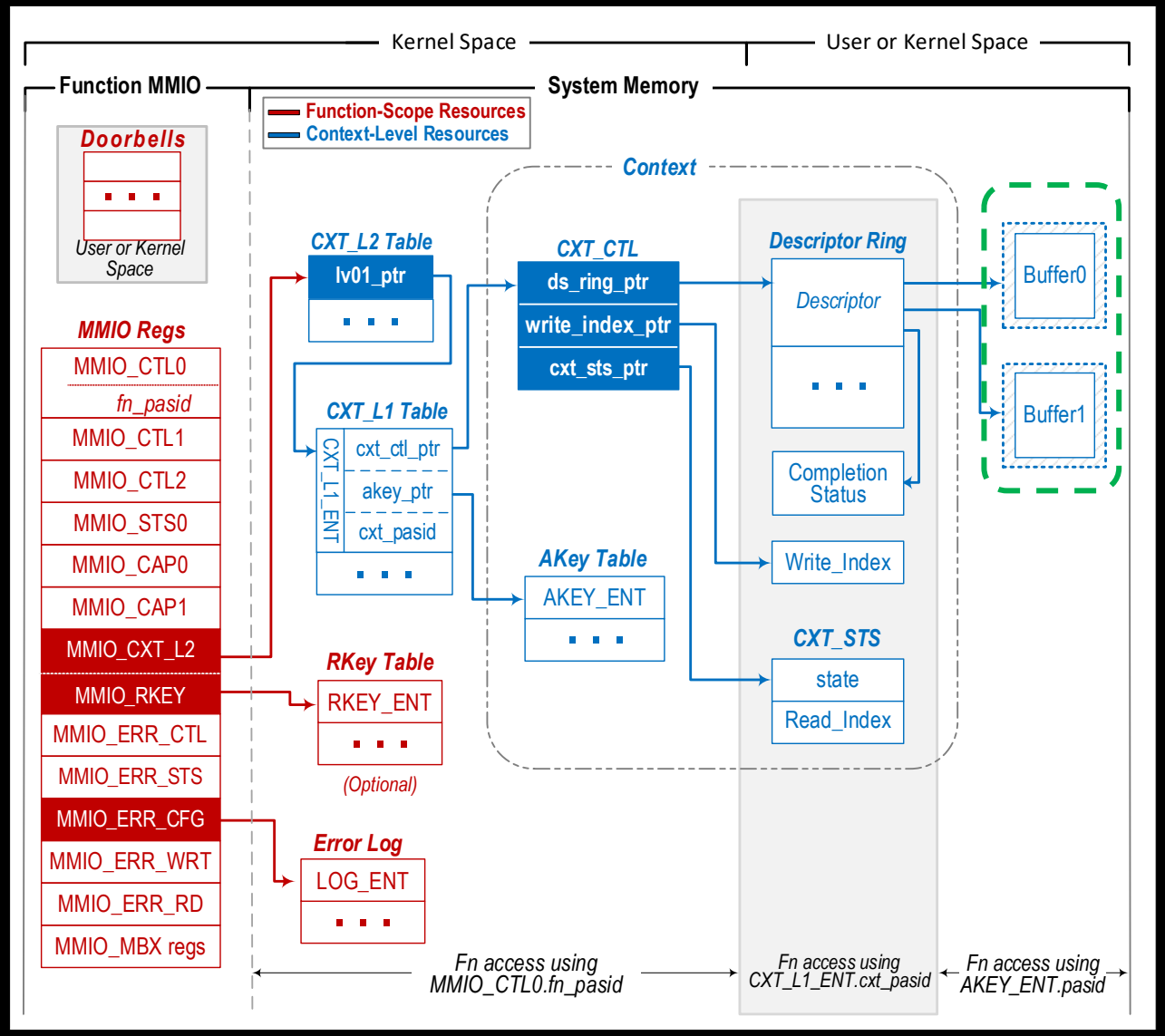
- SDXI functionality relies on MMIO & in-memory data structures
- Two-level table design supports many contexts
- Allow access control by tagging control structures with PASIDs
- The RKey Table controls remote accesses
- The Error Log records details of errors

SDXI – Context



- Context is a self-contained execution unit
- The Descriptor Ring is used for command submission.
- Context #0 is reserved for admin commands
- Developed with a simple submission flow

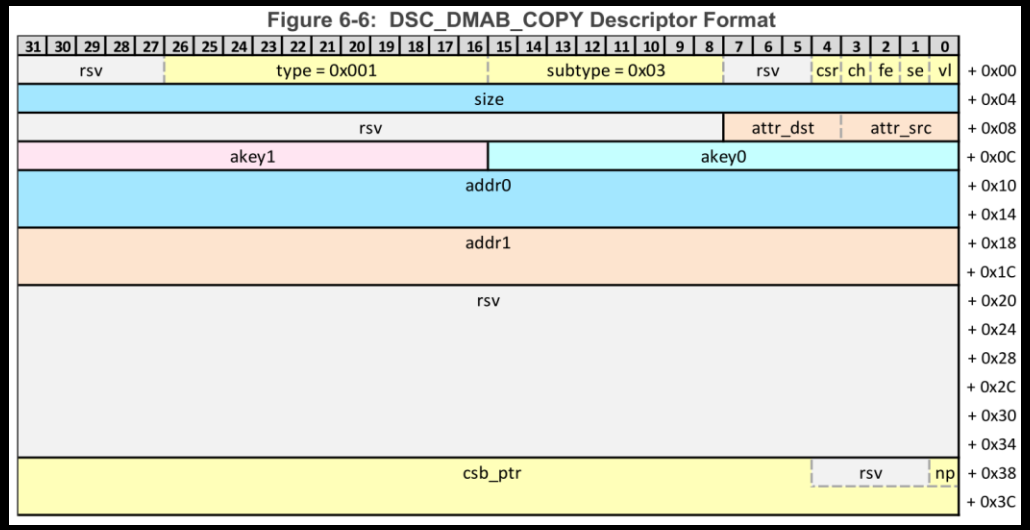
SDXI – Address Space



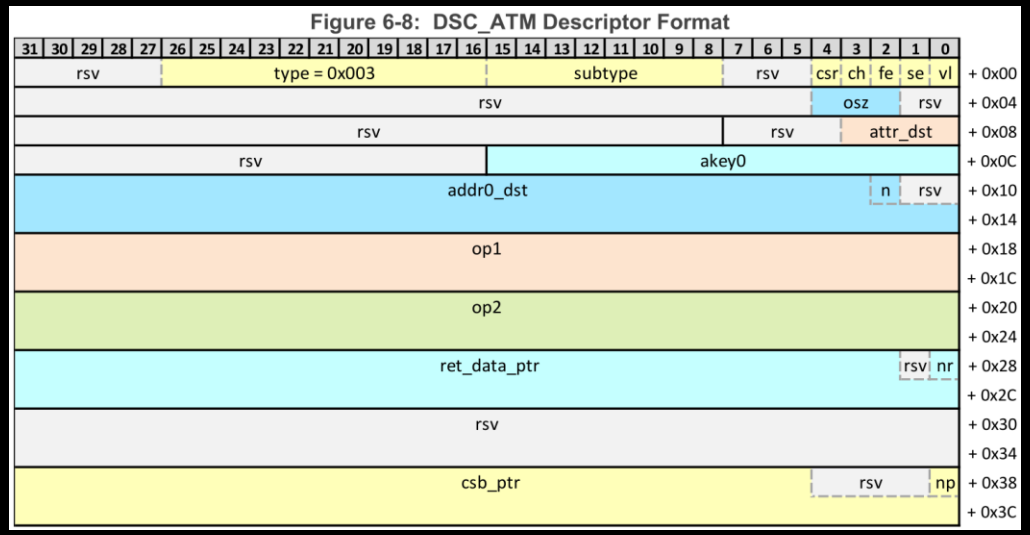
- Targeted memory buffers are limited to the context's own address space by default.
- External address spaces can be accessed via a <akey_entry, address> pair.
- The receiving SDXI function will validate the remote request via its local RKey Table.

SDXI – Descriptors

- An SDXI descriptor is a 64-byte data for operations.
- Descriptors are classified as different groups:
  - DMABaseGrp: `NOP`, `WRT_IMM`, `COPY`, `REPCOPY`
  - AtomicGrp: `SWAP`, `UADD`, `USUB`, `AND`, `OR`, etc.
  - AdminGrp: `UPD_FN`, `UPD_AKEY`, `START_NM`, `ADM_INTR`, etc.
  - IntrGrp: `INTR`
  - VendorGrp: Extensible, vendor-defined operations



DMA Copy Descriptor

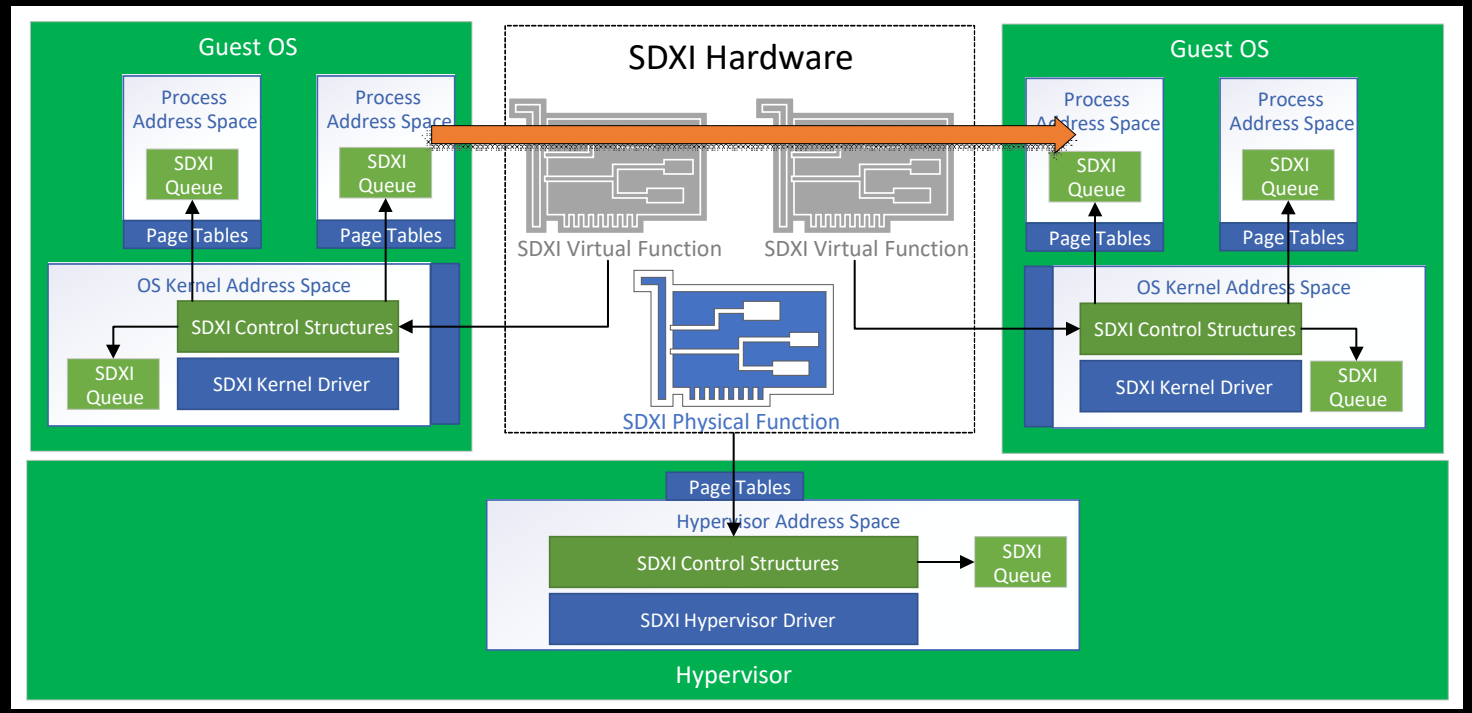


Atomic Descriptor

Use Example

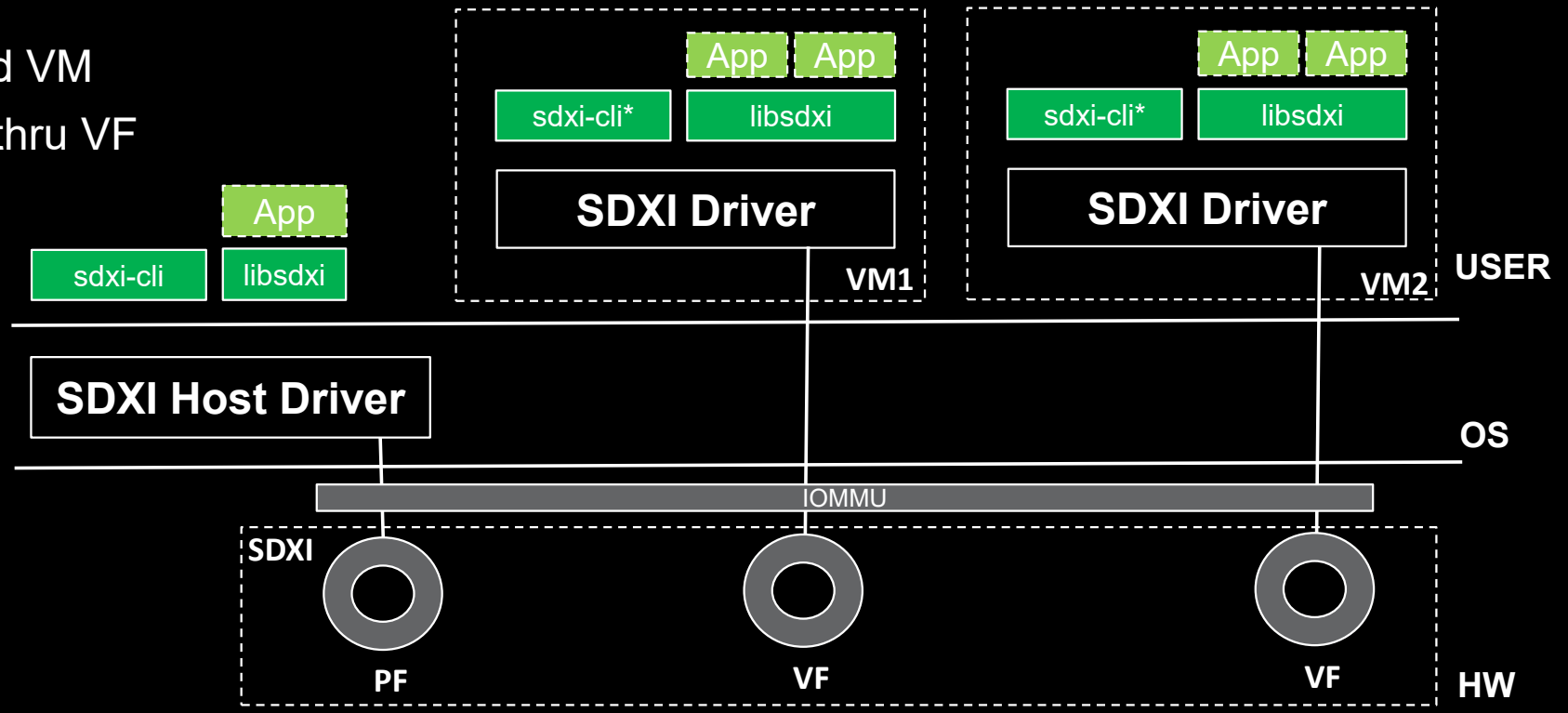
Process memory copying from Guest VM 1 to Guest VM 2 across passthru SDXI functions

- Guest VM 1 sets an AKey entry to indicate target address space
  - AKey entry = <Func_1, RKey_Index>
- Guest VM 2 sets a RKey entry to allow remote memory access
  - RKey entry = <Func_0, PASID of B>



SDXI Software Design

- Applicable to both bare-metal and VM
- Each VM is provided with a passthru VF
  - SDXI Function <=> VM
  - Context <=> Process
- Software components
  - sdxi driver
  - libsdxi
  - sdxi-cli



Now – As DMA Engine

- SDXI as a new DMA engine in Linux
  - Initial SDXI 1.0 support as dma-engine provider
- Features
  - Memory-to-memory data-mover using SDXI
  - Spec-compliant SDXI control and descriptor formats
  - Single-threaded, polled-mode operation
- Known limitations for improvement
  - No interrupt-driven multi-queue support yet (WIP)
  - Relatively simple context & descriptor management (locking, etc.)
- Status
  - RFC submitted and under view

Subject [PATCH RFC 00/13] dmaengine: Smart Data Accelerator Interface (SDXI)
Date Fri, 05 Sep 2025 13:48:23 -0500

The Smart Data Accelerator Interface (SDXI) is a vendor-neutral
architecture for memory-to-memory data movement offload designed for
kernel bypass and virtualization.

General information on SDXI may be found at:
https://www.snia.org/sdxi

This submission adds a driver with basic support for PCIe-hosted SDXI 1.0
implementations and includes a DMA engine provider.

It is very much a work in progress. Among other issues, the DMA
provider code only supports single-threaded polled mode, and context
management should use better data structures.

While we're addressing those shortcomings, we'd appreciate any
feedback on:

* Where the code should live. SDXI entails a fair amount of code for
  context and descriptor management, and we expect to eventually add a
  character device ABI for user space access. Should all of this go in
  drivers/dma/sdxi?

* Whether the DMA engine provider should use virt-dma/vchan. SDXI
  submission queues can be almost arbitrarily large, and I'm not sure
  putting a software queue in front of that makes sense.

Planned future SDXI work (out of scope for this series):

* Character device for user space access. We are evaluating the uacce
  framework for this.

* Support for operation types to be added in future SDXI revisions.

* Greater configurability for control structures, e.g. descriptor ring
  size.

Next – User Space Support

- A device driver provides full user-space access to SDXI HW
- Proposed ABI interface
  - `/dev/sdxi`
  - Two API classes
    - Privileged: device discovery & configuration, RKey management
    - User-space: Context lifecycle, AKey management, etc.
- ABI interface proposal

Category	IOCTL Commands	Description
Privileged	SDXI_PRIV_GET|SET_DEV_ATTR	read/write device attributes
Privileged	SDXI_PRIV_SET_DEV_RKEY	configure RKey entries
User	SDXI_GET_API_VER	returns API version to manage compatibility
User	SDXI_GET_SYS_INFO	returns list of SDXI devices and capabilities
User	SDXI_CREATE_CXT	create a per-process context
User	SDXI_SET_CXT_ATTR	update context attributes (QoS, security, ring size, etc.)
User	SDXI_SET_CXT_AKEY	manage per-context AKey entries
User	SDXI_CLOSE_CXT	close and clean up a context

Next – libsdxi

- SDXI user library for fast app development
  - `sdxi_cxt_create()` brings up /dev/sdxi
  - `sdxi_cxt_close()` tears them down
  - Descriptor helpers pack
- Status
  - Available in github for download

```
$ ./memcopy
SDXI memory copy test ...
    memory buffer src = 0x55a216dc8000
    memory buffer dst = 0x55a216dca000
Memory copy ==> SUCCESS
```

```c
#include "libsdxi.h"

#define TEST_BUFFER_ALIGN       4096
#define TEST_BUFFER_SIZE        4096

int main(int argc, char *argv[])
{
    int ret;
    sdxi_cxt_h cxt;
    sdxi_status res;

    res = sdxi_cxt_create("dma_copy_cxt", &cxt);
    sdxi_status_ok_or_die(res, "Failed to create SDXI context");

    /* init source and dest buffers */
    posix_memalign_check((void **)&src, TEST_BUFFER_ALIGN, EST_BUFFER_SIZE);
    posix_memalign_check((void **)&dst, TEST_BUFFER_ALIGN, TEST_BUFFER_SIZE);

    printf("SDXI memory copy test ...\n");
    printf("    memory buffer src = %p\n", src);
    printf("    memory buffer dst = %p\n", dst);

    memset(src, 0xAF, TEST_BUFFER_SIZE);
    memset(dst, 0xBE, TEST_BUFFER_SIZE);

    /* submit to sdxi */
    res = sdxi_mem_copy(cxt, dst, src, TEST_BUFFER_SIZE);
    sdxi_status_ok_or_die(res, "sdxi_mem_copy returned failure");

    /* compare buffer contents */
    if (memcmp(src, dst, TEST_BUFFER_SIZE) == 0) {
        fprintf(stderr, "Memory copy ==> SUCCESS\n");
        ret = 0;
    } else {
        fprintf(stderr, "Memory copy ==> FAIL\n");
        ret = 1;
    }

    sdxi_cxt_close(cxt);
    return ret;
}
```

Conversation

Item #1: DMA engine support upstream

- Added IRQ competition support, better locking mechanism, etc.
- Cleaning up with community feedback/comments
- **Q: Any additional items before upstream?**

Item #2: User space support

- How much do you like/hate a new UABI via IOCTL?
- **Q: Direction in user-space context management?**

Item #3: Tooling support

- libsdxi and sdxi-cli will be made soon
- **Q: How to accelerate upstream adoption, e.g. QEMU SDXI model?**

Item #3: Use cases

- Tested with memory zero'ing, kernel folio plugin, NTB support etc.
- **Q: Any suggestions?**