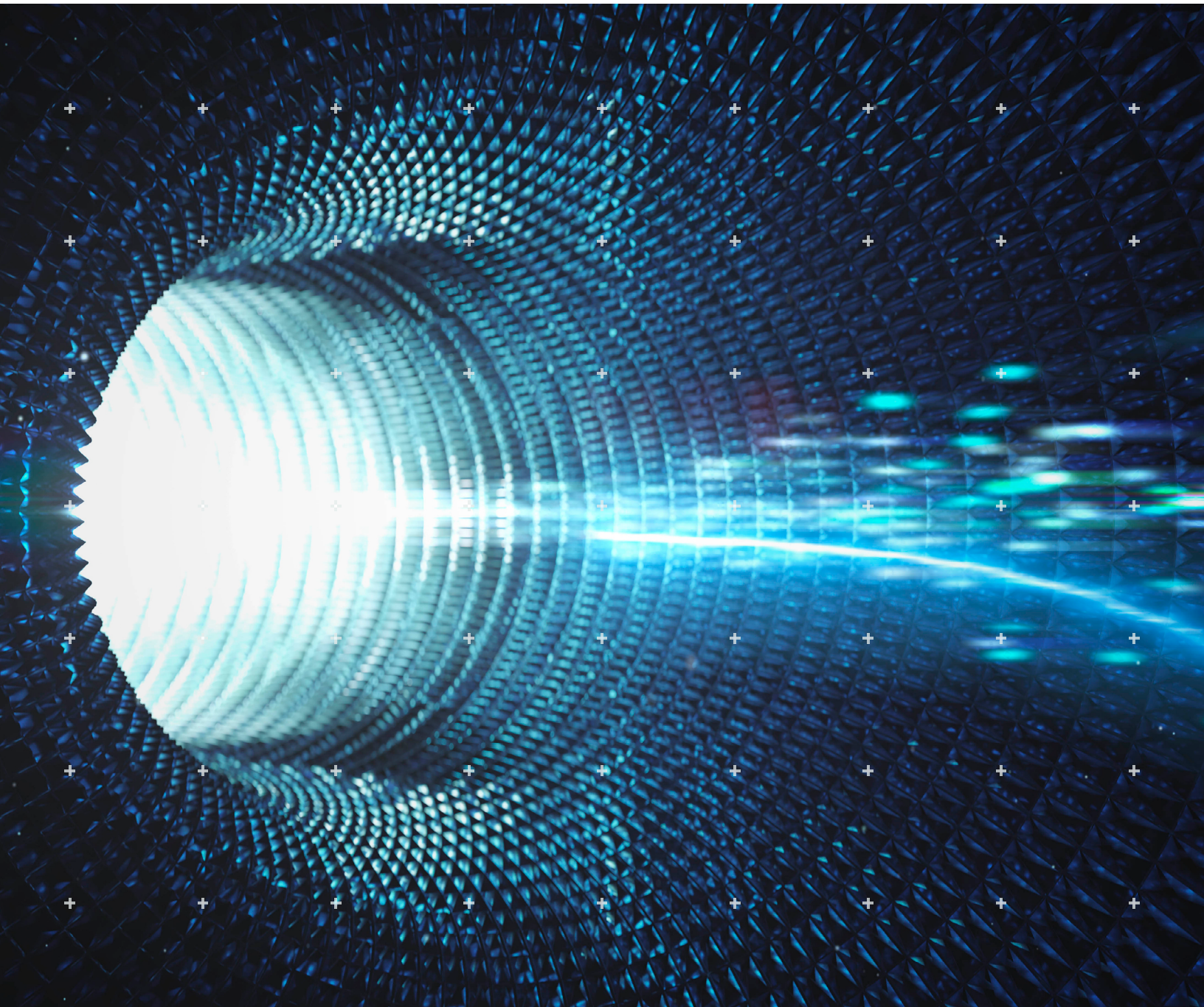
# Arm Storage Solution for SSD Controllers

arm

Solutions Brief



Data is being generated at a rapid pace and it is more critical than ever that storage devices combine high performance with energy efficiency. The Arm storage solution is a comprehensive blueprint for solid-state drive (SSD) applications that is proven and trusted by the industry's top companies. Arm combines security and energy efficiency across a range of performance points to help SSD designers innovate confidently and accelerate time to market with the best-fit solution.

## Why Arm: A Trusted Partner for Secure, Energy-efficient Storage Devices

**Faster Time to Market**

The combination of ready-to-run, open-source code from RTOS and Arm's integrated toolchain reduces fragmentation and provides a common foundation for fast development

**Energy-efficient Design**

Arm Cortex processors and system IP offer high-performance and energy-efficient solutions that are designed for complex computing tasks for storage-based devices

**Trusted Partner for Success**

Reduce risk and achieve success by choosing proven Arm technology that has been integrated into billions of storage devices to date

# Overview

Arm offers a broad portfolio that enables performance and efficiency optimization for SSD controllers. Cortex-R processors are most frequently used in enterprise, datacentre, and client SSDs, though some high-end enterprise and datacentre devices use Cortex-A processors. Very low-power client controllers opt for Cortex-M processors, and processors such as Cortex-M55, Cortex-M33, and Cortex-M0+ can also be used for helper functions in SSDs.

| Arm Cortex-A Processors | Arm Cortex-R Processors | Arm Cortex-M Processors |
|---|---|---|
| Designed for devices undertaking complex compute tasks | Optimized for high-performance, real-time applications | Built for discrete processing and microcontrollers |
| Typical processors for SSD controllers include: | | |
| + Cortex-A55<br>+ Cortex-A53 | + Cortex-R82<br>+ Cortex-R8<br>+ Cortex-R5 | + Cortex-M55 and Cortex-M33 with optional Arm Custom Instructions and coprocessor interface<br>+ Cortex-M7<br>+ Cortex-M0+ |

# Key Features

**Low latency, real-time performance with Cortex-R and Cortex-M processors**
Required to move data in the fastest time possible

**CoreSight and ecosystem of debug tools** Faster development with a unified and complete system and core debug available across all Arm processors

**Balanced performance, power, and area with Cortex-A processors**
Required for a front-end processor that needs increased performance over Cortex-R processors

**Machine learning (ML) features**
Improved ML performance for computational storage with Neon (Cortex-A series and Cortex-R82) and Helium technologies (Cortex-M55) and Ethos NPUs

**Low power**
Required for the SSD controller to fit power constraints

**Arm Custom Instructions**
Enables power reduction and increased performance by implementing a function in hardware that would otherwise take multiple cycles or instructions to complete

# Solution Block Diagram

**SSD Solution Featuring:**
- Arm Cortex Processors
- Arm Generic Interrupt Controller (GIC)
- Arm Interconnect
- Arm CoreSight debug
- Arm Memory Controller
- Arm DMA

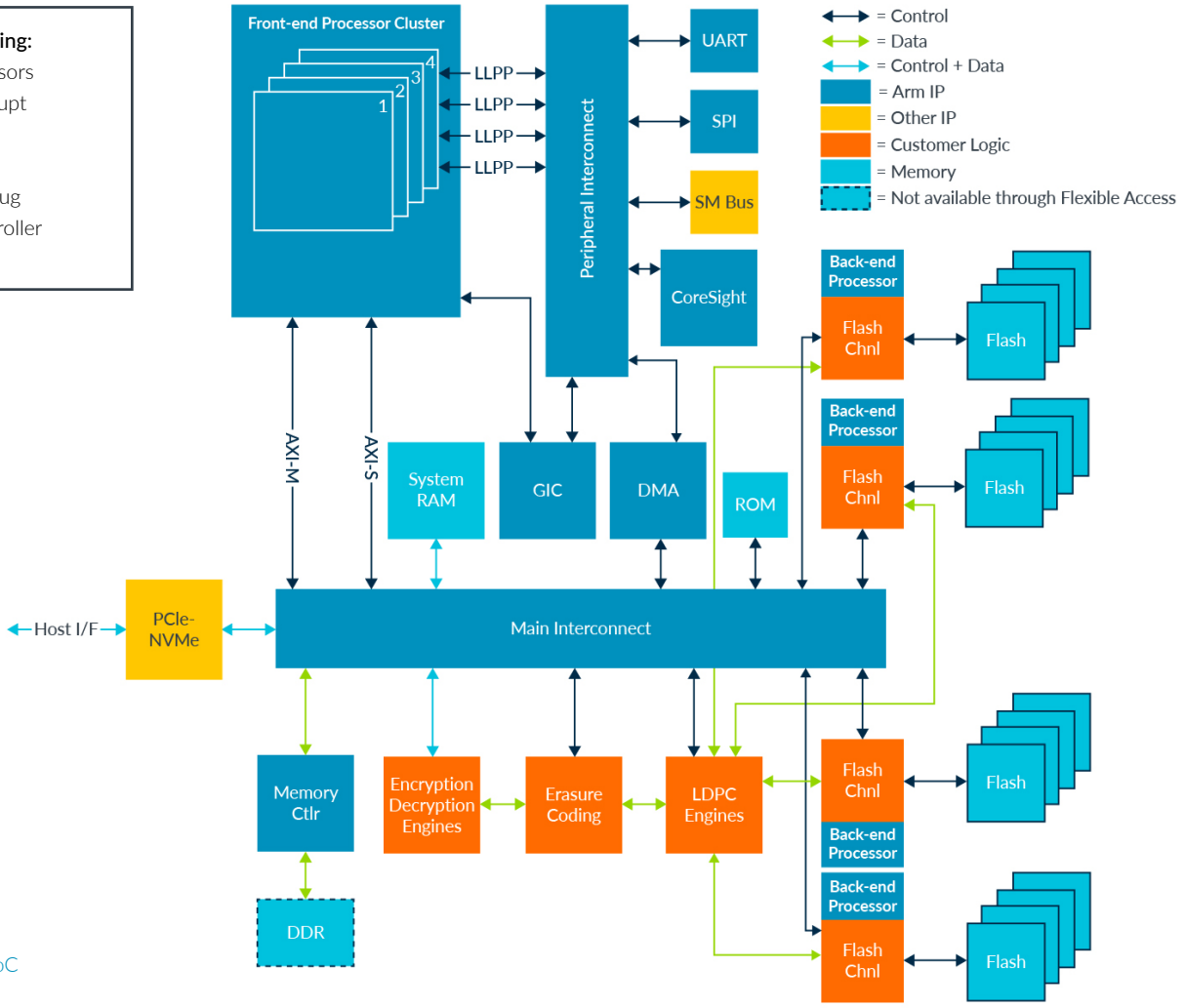



Figure 1 Typical SSD SoC

Front-end processors are usually Cortex-R processors but can sometimes be Cortex-A processors to support more intensive wear-levelling algorithms.

Wear-levelling is the process of systematically erasing and writing all NAND flash locations to ensure all locations wear out at about the same time. It may require moving unchanging data (cold data) already stored so a block can be erased and rewritten with rapidly changing data (hot data).

## Flash Translation Layer (FTL)

The **front-end processors** are responsible for maintaining the FTL tables that map logical addresses to physical addresses. As part of the FTL, the front-end processors also handle wear-levelling algorithms to determine where the data should be written, which data should move, and which locations require garbage collection.

## Read Requests and Write Requests

Read and write requests arrive from the host over the **PCIe/NVMe** interface. The requests arrive in the form of descriptors that describe the parameters of a transfer. Write requests also carry the data to be written. These descriptors are written to either the **DDR** memory or a system RAM using a **DMA** operation from the host. For write requests, the user data is transferred to the controller DDR using a separate DMA operation.

A front-end processor is notified about the arrival of this descriptor. It then parses the descriptor to modify some of the parameters. In particular, the Logical Block Address (LBA) provided by the host must be modified to the physical address in the NAND where the read data is stored. With the correct physical address, the descriptor can be dispatched to the appropriate **flash channel** using a DMA.

A NAND can be very slow, it might not be ideal for the processor to work directly with the NAND. Arm processors provide the flexibility to work with NAND from different manufacturers with different commands.

For read requests, the flash channel issues a read operation to the appropriate NAND die. When the data is available, it is moved from the NAND die buffer to the flash channel and then to the LDPC engines, as space is available. The **LDPC engines** perform ECC correction and data recovery from the media. If necessary, erasure coding operations are performed to recover data lost due to a block or page failure. Encrypted data is decrypted and stored in controller DDR. Finally, the user data is moved to host memory using a DMA operation to write into the DDR of the host, and then the host is notified that the data transfer completed.

For write requests, the user data in DDR is sent to the SSD data path of encryption, erasure coding, and LDPC encoding. When the encoded user data reaches the flash channel, the flash channel issues an operation to write the data to the NAND buffers. When all the data is in the NAND buffer, a program operation is issued to program the NAND. A completion response is generated by the flash channel that is sent back to the host.

The **back-end processors** are responsible for the manipulation of the NAND control sequences and the last modifications to the descriptor parameters needed to read or write the data. The back-end processors do not interface directly to the NAND. Instead, they build sequences of operations for a hardware state machine to execute.

Back-end processors are usually Cortex-R, but can sometimes be Cortex-M.

Cortex-R processors are ideal because of their high frequency options and real-time, deterministic, low-latency response. In lower power requirements and lower performance point SSD controllers, Cortex-M processors reduce controller power further.

With Cortex-R, there can be 1 per flash channel or up to 1 per 4 flash channels (including 1 per 2 flash channels). With Cortex-M, it is usually 1 per flash channel.

### Garbage Collection

Garbage collection is the process of collecting good data and preparing empty blocks to be erased.

When a file is saved multiple times, the data must be saved to a new physical address in the NAND each time. Obsolete file revisions are mixed with valid files in a block.

To erase the entire block for new data, the valid files must be read and then written to another location in NAND. Then, the entire block can be erased to recover the space consumed by the obsolete file revisions.

**Note:** The process is a combination of read and write, except that no data is returned to the host. In fact, it is completely transparent to the host.

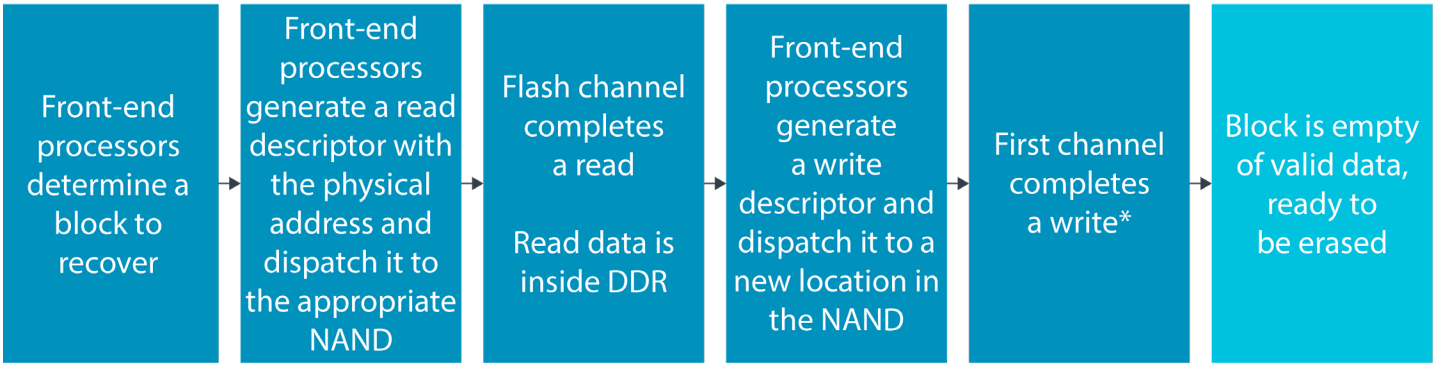



**Figure 2** Garbage collection process

**Multiple blocks are recovered at the same time to ensure that enough valid files are aggregated to write one page into the NAND.*

### Erase

The front-end processors generate a descriptor to erase the block. The descriptor is dispatched to the appropriate flash channel and the flash channel to the NAND. When complete, the back-end processors notify the front-end processors to update the Flash Translation Layer (FTL) tables to indicate that this block is now available.

## Introducing Computational Storage for SSDs

The volume of data companies generate is expected to grow by an astonishing 27 percent a year[1] and to be successful in this increasingly digital world, organizations need the infrastructure and technology to be capable of delivering and storing data and analytics in a fast, secure, and efficient way.

Computational storage adds computing capabilities to traditional storage devices, by putting processing power where it is needed and gives companies quick and easy access to vital information. The compute could include a Cortex-A series or Cortex-R82 Linux-capable processor, or a Neural Processing Unit (NPU) for ML workloads. Find out how computational storage could benefit your SSD by reading this guide to computational storage on Arm.

## Design Considerations

Designing an SoC for an SSD device requires a significant amount of investment and resource – from design verification to physical design and to the software development debug and validation – they all need to support the device.

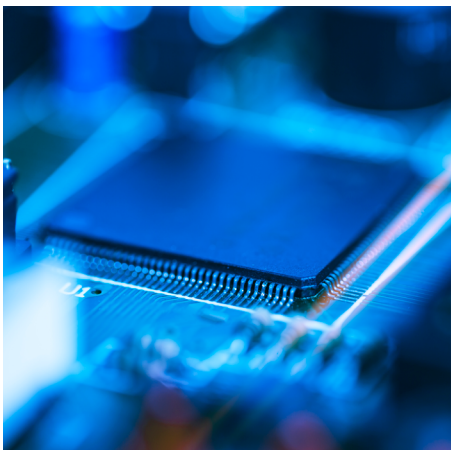
Consider the following tips to help simplify the process of designing an SSD:

- Narrow the Arm processor options based on the desired SSD performance, number of address bits, and real-time response requirements.
- Model your processor selection(s) and SSD controller with Fast and Cycle Accurate Models as the architecture is developed to increase confidence in the processor choice and SSD architecture.
- Configure and integrate the Arm IP RTL into the SSD controller. Simulate the configuration with the provided integration testbench.
- Synthesize the processor early to identify timing paths to the cache and TCMs to achieve the best balance between power, performance, and area.
- Rapidly develop and debug software, in parallel to the hardware development, using Fast Models to reduce time to market.

## Get Started with your SSD

Arm can help you design SSD controllers by providing the IP, tools and support to get started on your design with Arm Flexible Access. Arm Flexible Access offers a simple way to evaluate and fully design SoC storage solutions with a wide-ranging mix of Arm IP before committing to production.

Learn more about our storage solutions at storage.arm.com and contact us to discuss developing your next Arm-based SSD storage solution with one of our technical experts.



# Terms and Abbreviations

| | |
|---|---|
| DDR | Double Data Rate |
| DMA | Direct Memory Access |
| ECC | Error Correcting Code |
| FTL | Flash Translation Layer |
| FTL table | FTL tables are large tables that change dynamically as data is written and erased on the SSD. They map logical addresses to physical addresses |
| GIC | Generic Interrupt Controller |
| LBA | Logical Block Address |
| LDPC engine | Low-Density Parity Check engine. LDPC code provides powerful error-correction capability and is adapted for real-time performance and constrained-area devices |
| ML | Machine Learning |
| NPU | Neural Processing Unit |
| NVMe | Non-Volatile Memory express |
| PCIe | Peripheral Component Interconnect express |
| SMBus | System Management Bus |
| SoC | System-on-Chip |
| SPI | Serial Peripheral Interface |
| SSD | Solid-State Drive |
| TCM | Tightly Coupled Memory |
| UART | Universal Asynchronous Receiver-Transmitter |
| UPP | Universal Parallel Port |
| Wear levelling | Technique used to increase the lifetime of the memory by distributing the write operations in such a way that all NAND memory cells in the SSD are written to equally |

1 https://www.seagate.com/files/www-content/our-story/trends/files/idcseagate-dataage-whitepaper.pdf

arm