An Introduction to SPI-NOR Subsystem

By
Vignesh R
Texas Instruments India
vigneshr@ti.com

About me

- Software Engineer at Texas Instruments India
- Part of Linux team that works on supporting various TI SoCs in mainline kernel
- I work on supporting peripheral drivers on TI SoCs, mainly QSPI, UART, Touchscreen and USB
- This presentation is mainly based on my experience of getting QSPI controllers on TI platforms to work in mainline kernel

What’s in the presentation?

- SPI-NOR flash and types
- Communication with SPI-NOR flashes
- SPI-NOR framework
- SPI-NOR controllers and types
- Writing a controller driver
- Ongoing work and what’s missing

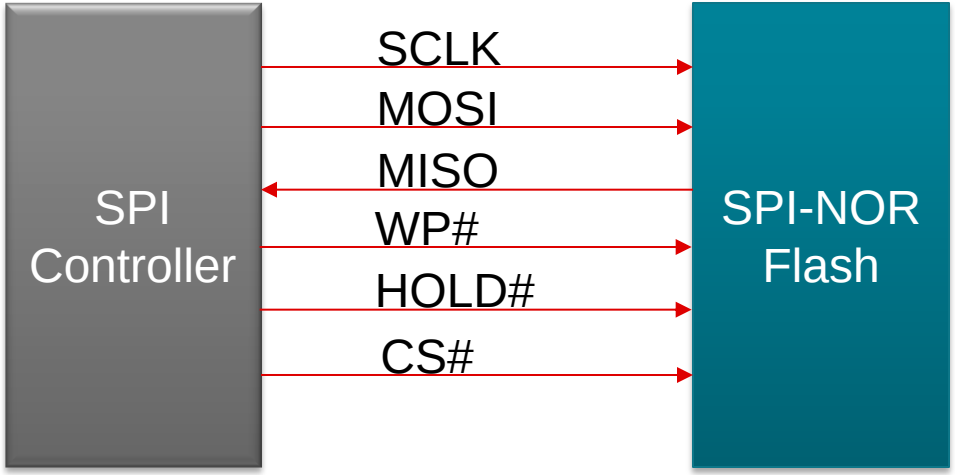
What is a SPI-NOR Flash?

- Array of storage cells that behave like NOR gate → NOR flash
- Two types of NOR flash:
  - Parallel NOR
  - Serial NOR
- Serial NOR flash that is interfaced to SoC via SPI bus and follows SPI protocol → SPI-NOR Flash
- Reduced pin counts compared to parallel NOR

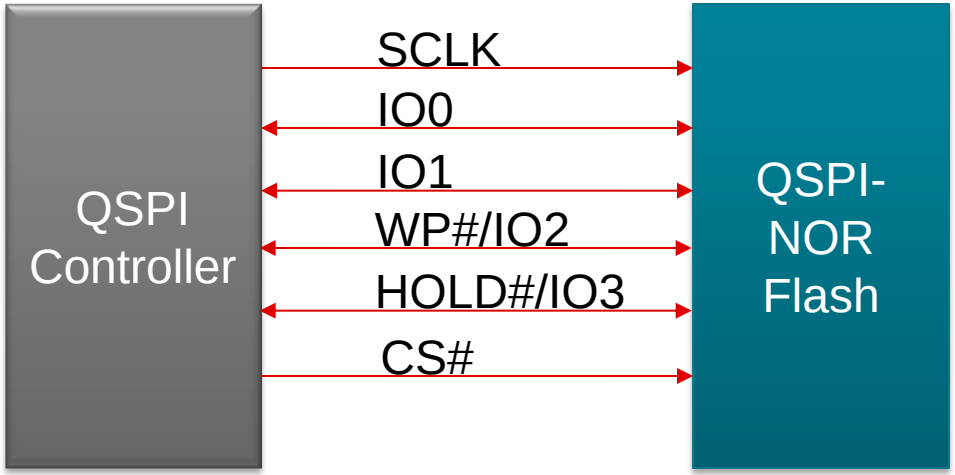
Why SPI-NOR flash?

Property	NAND	eMMC	SPI-NOR
Density	Upto 128GB	Upto 128GB	Upto 512MB
Bus width	x8/x16	x4/x8	x1/x2/x4/x8
Read speed	Slow random access	Similar to NAND	Fast random access
Write	Fast writes	Fast writes	Slower
Setup Requirements	ECC and bad block management	Needs tuning (for higher speed)	No overhead

Typical SPI-NOR flash



Multi IO Flash



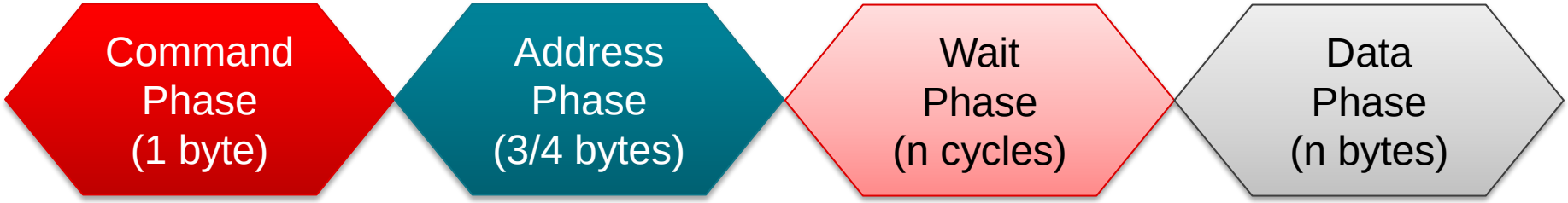
There are:

Dual IO, Quad IO and Octal IO flashes

SPI-NOR Flash Hardware

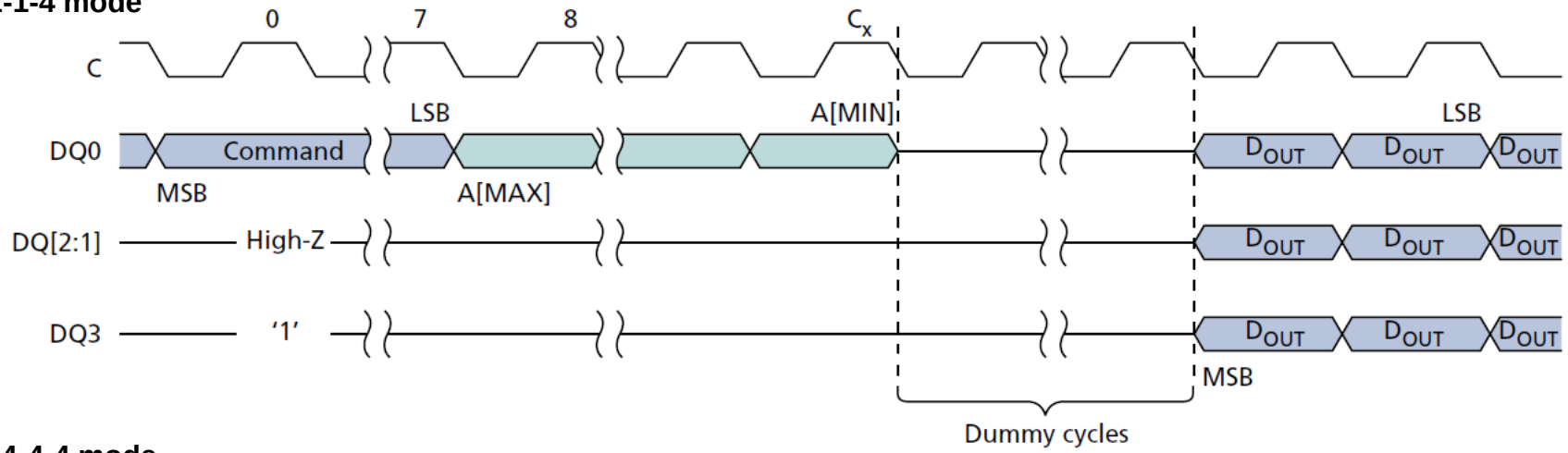
- Flash is composed of Sectors and Pages
- Smallest erasable block size is called Sector
  - May be 4/32/64/256 KB
- Sectors sub-divided into Pages
  - May be 256/512 bytes
  - Flash program is usually in page size chunks (though not necessary)
- Need to send Write Enable(WREN) command before a write or erase operation
- Most flashes support Read ID(RDID) command for flash detection

Communication Protocol

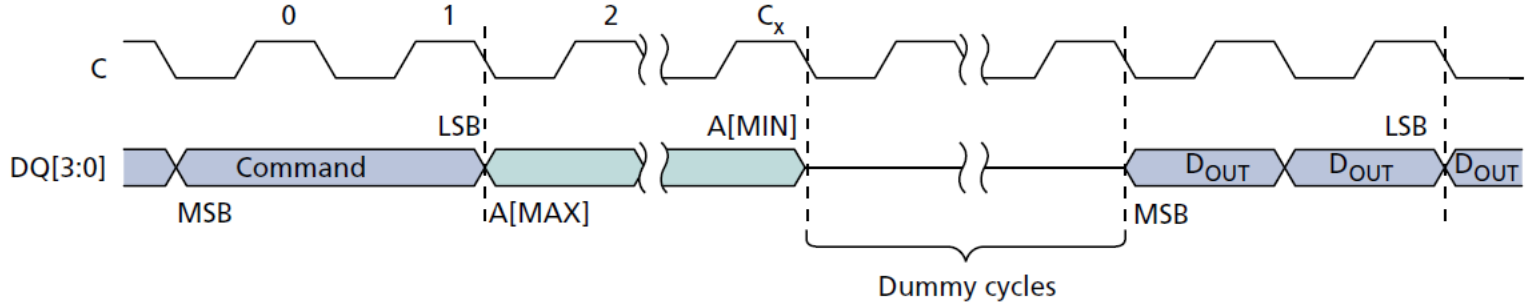


Multi IO Read: 1-1-4 and 4-4-4

1-1-4 mode



4-4-4 mode



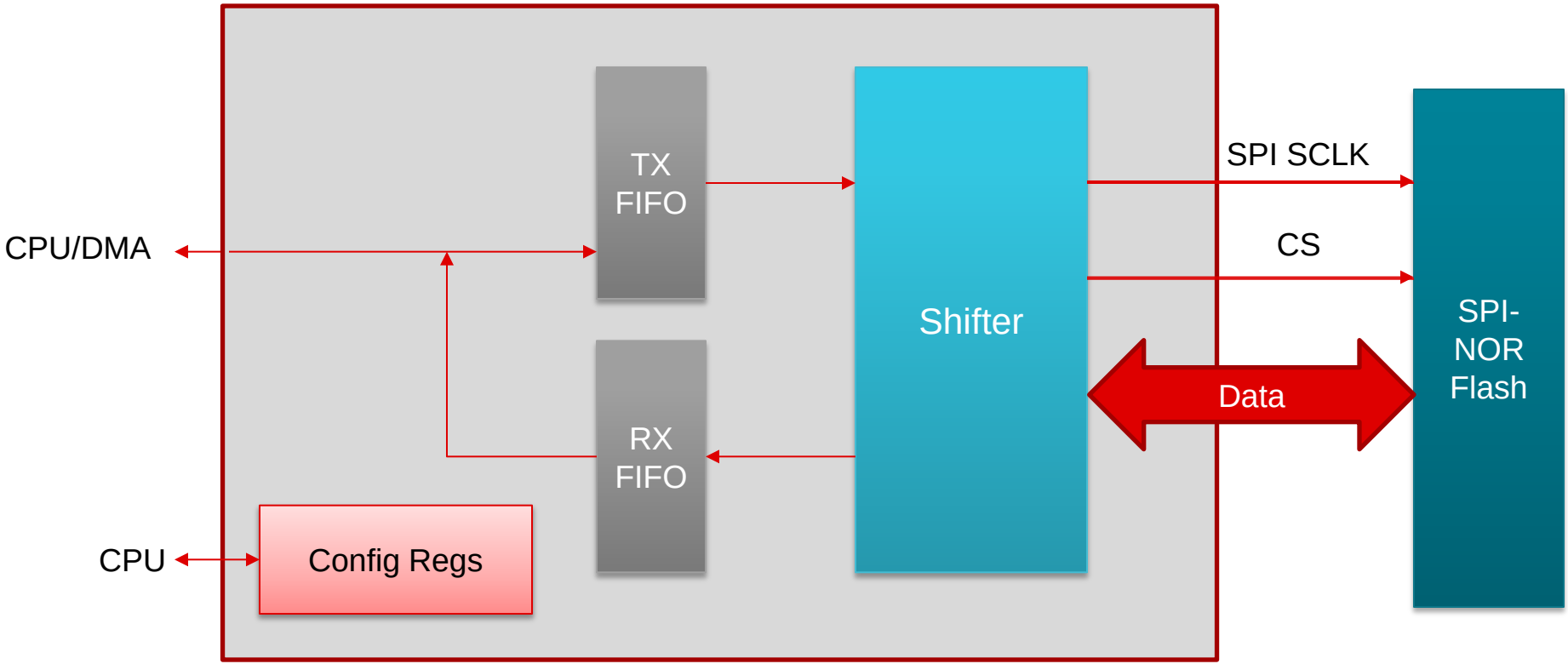
Types of Controllers

- Traditional SPI controllers
  - Provide direct access to SPI bus
  - Are not aware of the connected SPI slave device
  - Normally does not have deep FIFOs
- SPI-NOR Controllers
  - Aware of flash communication protocol (command, address and data phase)
  - Low latency access to flash, read pre-fetch and large HW buffer
  - May not provide direct SPI bus access
- Specialized SPI Controllers
  - Support both traditional SPI devices and Flashes
  - Typically, provides accelerated SPI-NOR access

SPI-NOR Framework

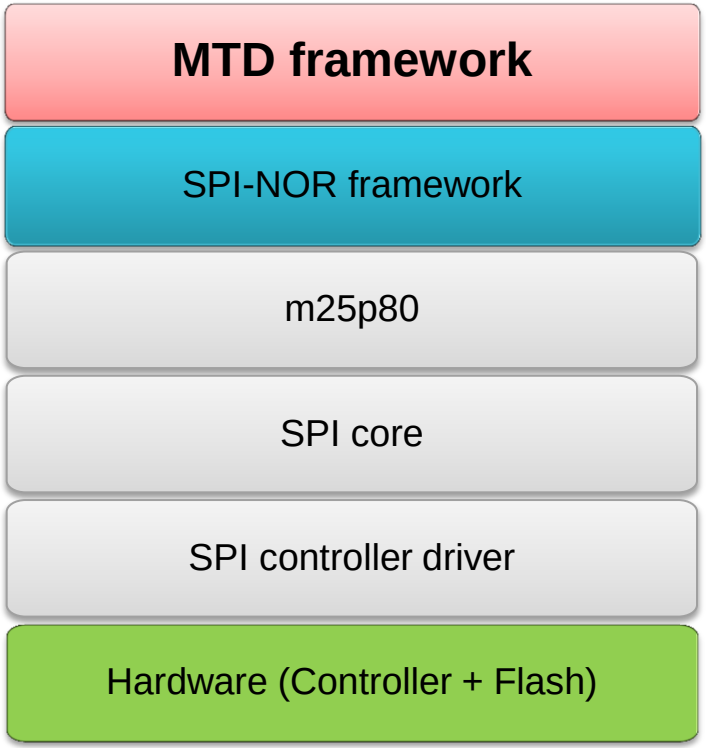
- Merged in v3.16
- Under Memory Technology Devices(MTD) Subsystem:
  - `drivers/mtd/spi-nor/spi-nor.c`
- Derived from pre-existing m25p80 flash driver code
- Why SPI-NOR framework?
  - Support controllers that only support flash slave devices
  - Support SPI-NOR/Specialized SPI controller hardware
    - Know flash specific data like opcodes, address width, mode of operation etc
  - Detect connected flash and choose suitable protocol for read/write/erase

Traditional SPI Controller



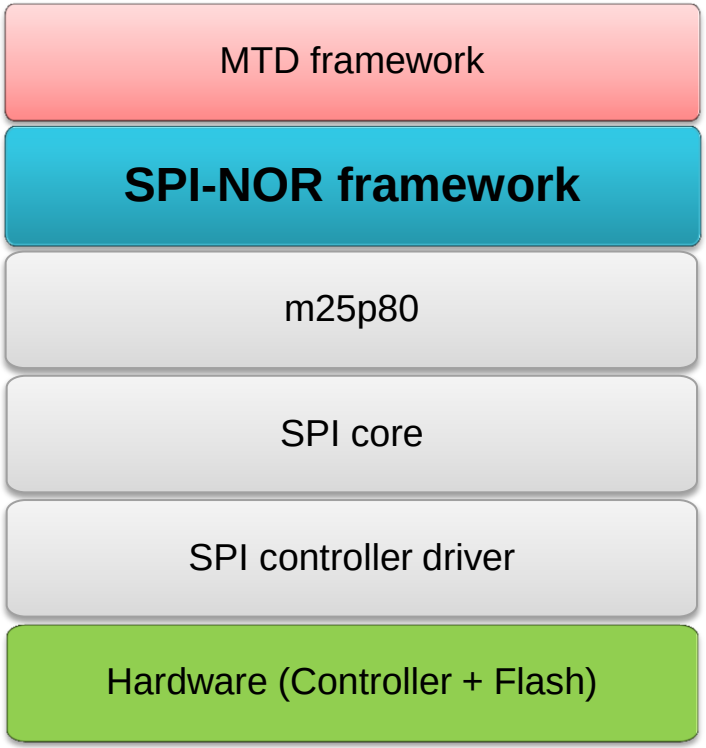
Accessing flash via SPI framework

- MTD layer abstracts all type of raw flash based devices like NAND, NOR and similar devices.
- Provides char(/dev/mtdX) and block(/dev/mtdblockX) device support
- Abstracts flash specific properties like sector, page and ECC handling
- Wear and bad block handling using UBI
- Handles partitioning of flash storage space
- `/proc/mtd` lists all devices



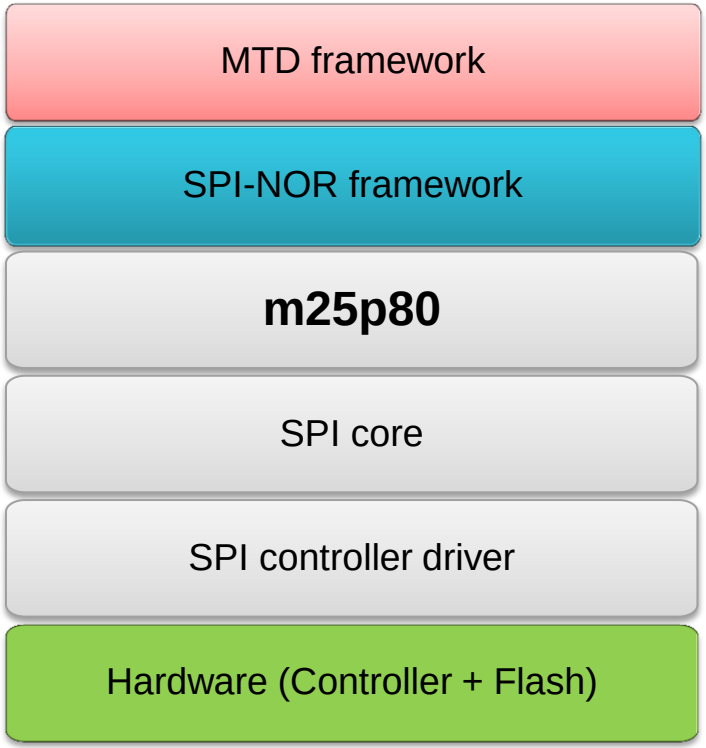
Accessing flash via SPI framework

- Handle SPI-NOR specific abstractions
  - Implement read, write and erase of flash
  - Detect and configure connected flash
  - Provide flash size, erase size and page size information to MTD layer
- Provides interface for dedicated SPI-NOR controllers drivers
  - Provide opcode, address width, dummy cycles information
- Support Multi IO flashes



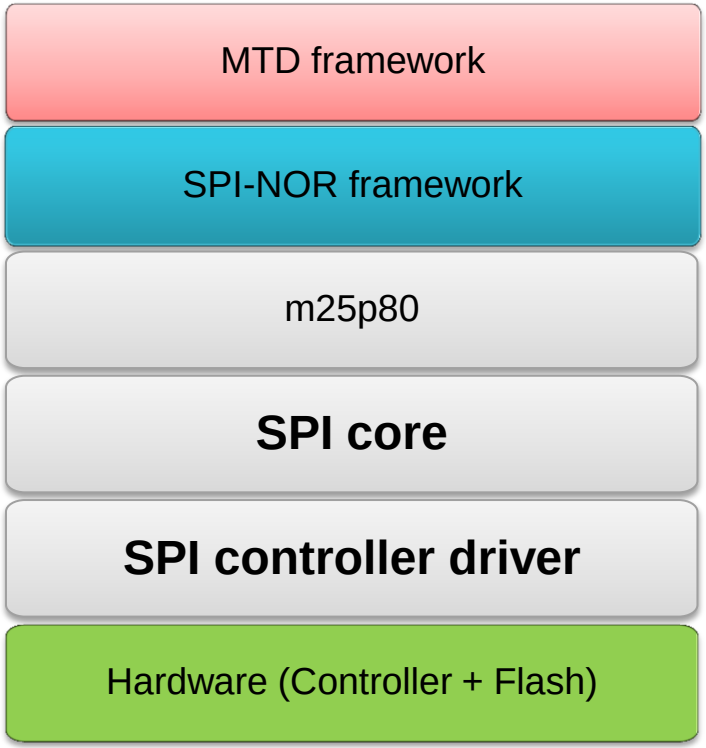
Accessing flash via SPI framework

- Translation layer between SPI-NOR framework and SPI core
- Convert command, address and data phases into `spi_transfer` structs based on data that is supplied by SPI-NOR (via `spi_nor` struct)
- Generate `spi_message` objects for `spi_transfer` and submit to SPI core for read/write or other flash operations

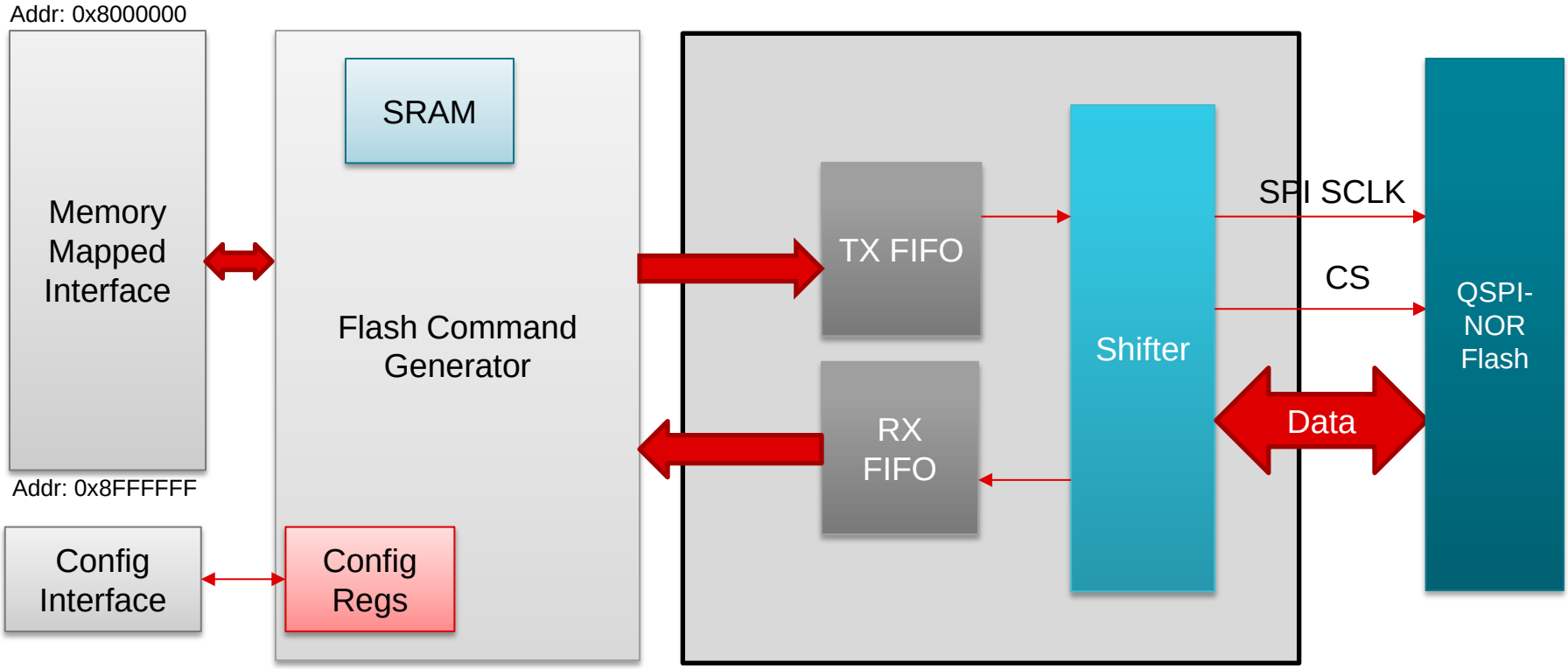


Accessing flash via SPI framework

- SPI core validates, queues and sends SPI messages from upper layer to controller drivers
- SPI controller driver writes data to TX FIFO and reads data from RX FIFO
- Does not distinguish transfers as command or data or address

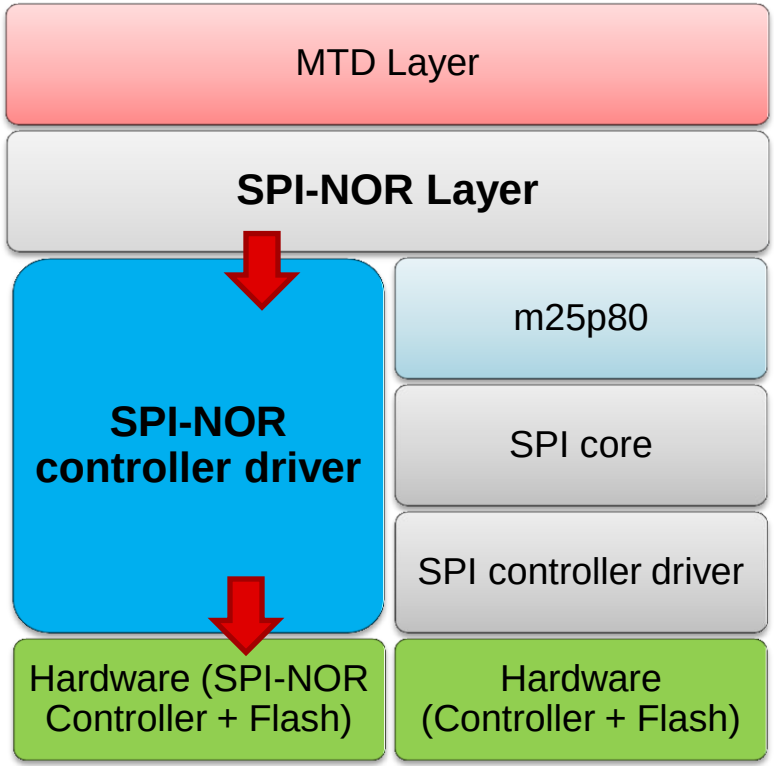


SPI-NOR controller-MMIO interface

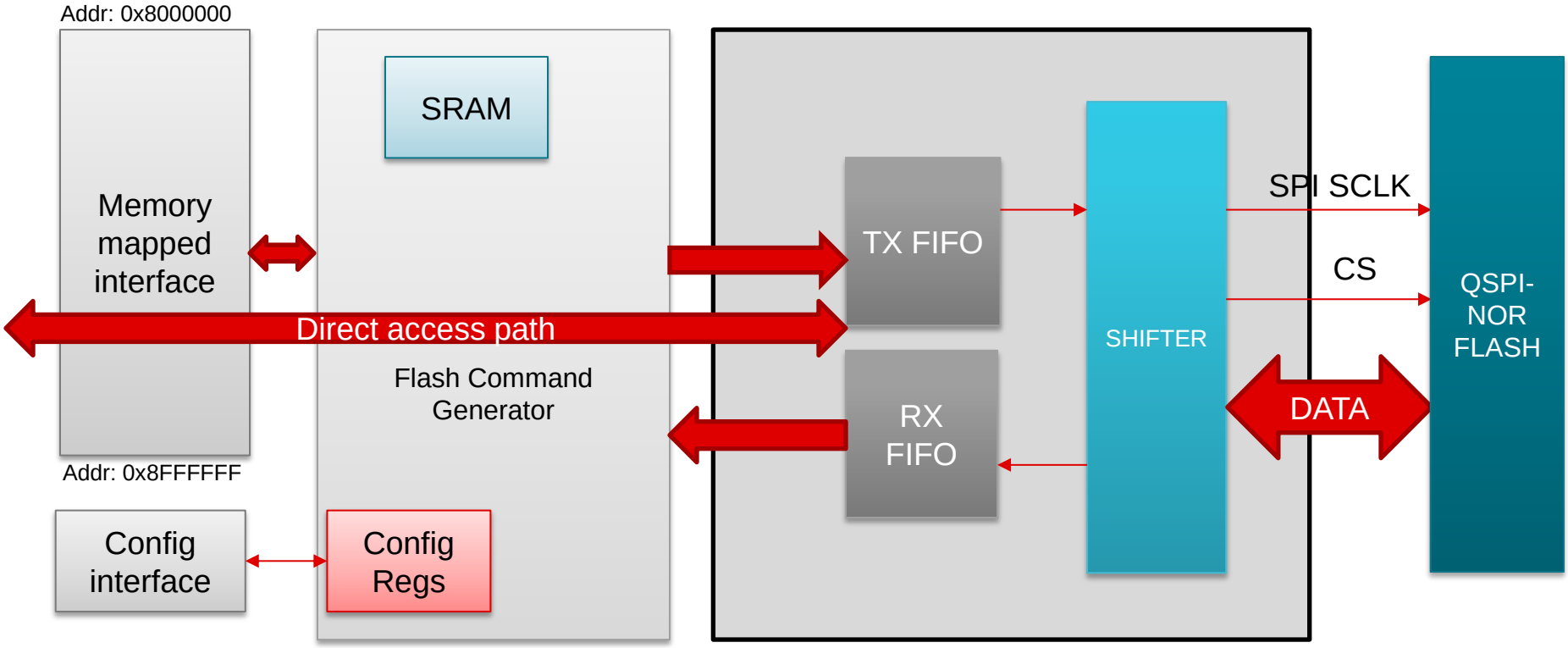


Accessing flash via SPI-NOR framework

- SPI-NOR layer provides information about the connected flash
- Passes `spi_nor` struct:
  - Size, page size, erase size, opcode, address width, dummy cycles and mode
- Controller configures IP registers
- Controller configures flash registers as requested by framework
- Controller drivers implements reads and writes
  - MMIO interface or from internal HW buffer

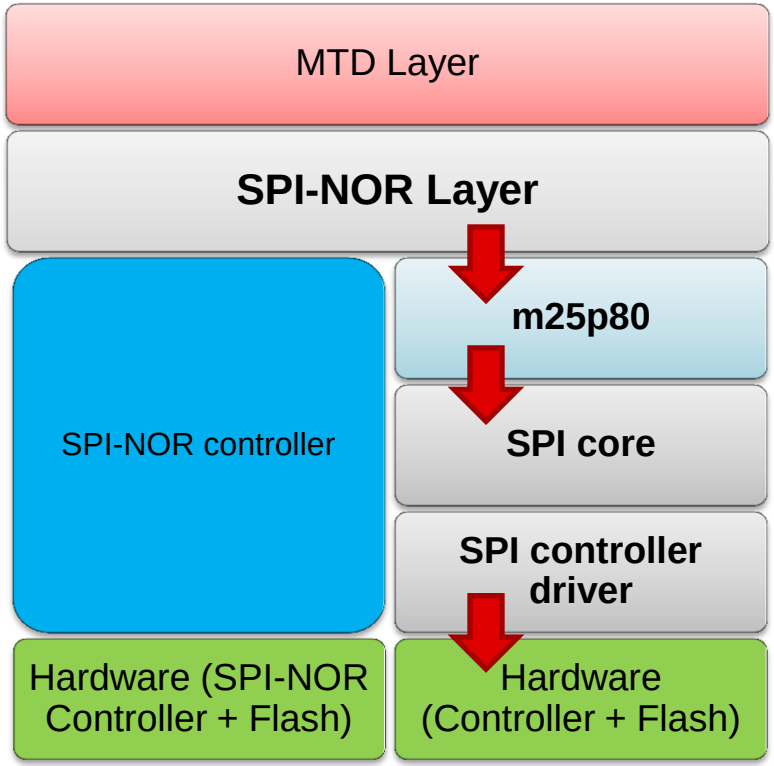


Specialized SPI controller-MMIO interface



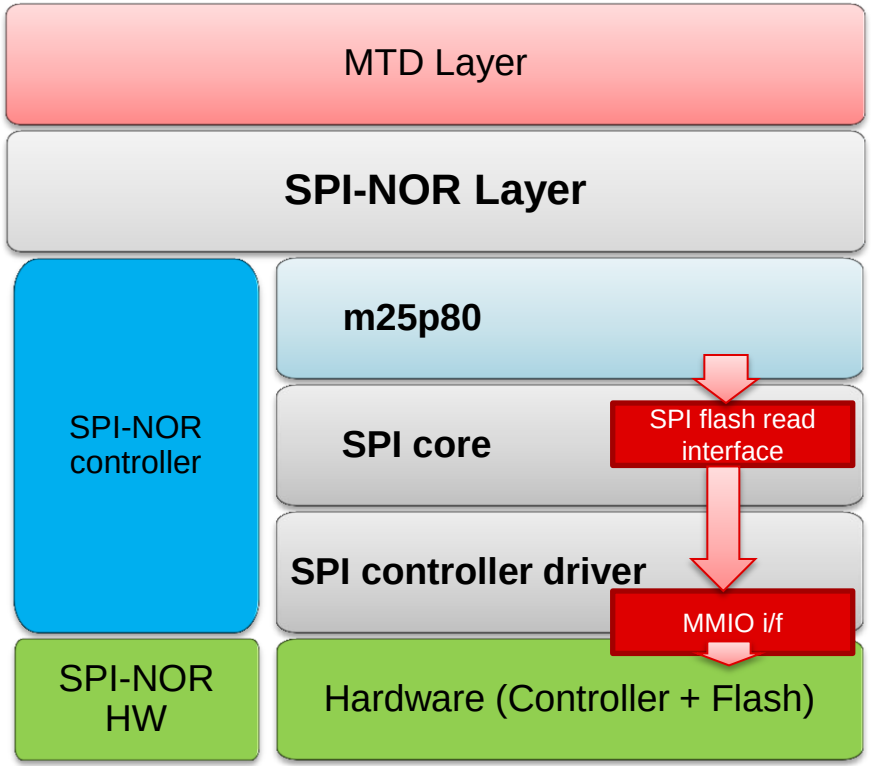
Specialized SPI controllers with MMIO support

- SPI flash is configured using m25p80 and regular SPI interface
- Usually writes and erase operations are also done via SPI regular interface using `spi_message` struct



Specialized SPI controllers with MMIO support

- Flash read operation is done via MMIO interface.
- m25p80 driver calls spi_flash_read() API of SPI core
- Drivers of SPI controller with MMIO interface implement spi_flash_read()
- `spi_flash_read_message` struct provides info related to flash



Where to put your driver?

- Supports any type of SPI device and direct access to bus
  - Use SPI framework
- Supports only SPI-NOR flashes and optimized for low latency flash access
  - Use SPI-NOR framework
- Supports all SPI devices and has special interface for flash
  - Use SPI framework and also implement spi_flash_read() interface

Writing a SPI-NOR controller driver

- At least following four callbacks need to be implemented:

```
int (*read_reg)(struct spi_nor *nor, u8 opcode, u8 *buf, int len);

int (*write_reg)(struct spi_nor *nor, u8 opcode, u8 *buf, int len);

ssize_t (*read)(struct spi_nor *nor, loff_t from, size_t len,
                u_char *read_buf);

ssize_t (*write)(struct spi_nor *nor, loff_t to, size_t len,
                const u_char *write_buf);
```

  - Call `spi_nor_scan()` to ask SPI-NOR framework to discover connected flash
  - Then call `mtd_device_register()`

Cadence QSPI DT fragment

```
qspi: qspi@2940000 {
        compatible =  "cdns,qspi-nor";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0x02940000 0x1000>,
              <0x24000000 0x4000000>;
        interrupts = <GIC_SPI 198 IRQ_TYPE_EDGE_RISING>;
        flash0:flash@0 {
                compatible = "jedec,spi-nor";
                reg = <0>;
                spi-max-frequency = <96000000>;
        };
        flash1: flash@1 {
                 …
        };
 };
```

Performance Comparison

Parameter	SPI transfers	SPI-NOR controller driver	SPI core's flash read interface
Read Speed	800 KB/s	4MB/s	4MB/s
CPU Load	~70%	~100%	~100%
Read with DMA	No HW support	No support in framework	20MB/s (15% CPU load)
Write Speed	400KB/s	400KB/s	400KB/s

Using TI QSPI controller on DRA7 SoCs under different framework with SPI bus rate of 64MHz

Ongoing work

- Choosing the right opcode based on controller and flash capabilities
  - Making sure communication with flash is stateless
  - Use opcodes that support 4 byte addressing
- Choosing 1-1-4 or 1-4-4 or 4-4-4 mode
  - Quad Enable (QE) bit behavior is different on different flashes
    - Spansion supports (1-1-4 and 4-4-4) but Micron supports only (4-4-4)
- Handling different sector sizes
  - A Flash may support 4K and 32K/64K/256K sectors
- Serial Flash Discoverable Parameters(JESD216) and Basic Flash Parameter Table Support (merged in v4.14)
- Octal mode and DTR mode support

Adding DMA support

- Flash filesystems and mtdblock are not written with DMA in mind
  - Uses vmalloc'd buffers
  - Known to cause problems with VIVT caches
  - Buffers backed by LPAE memory are not accessible by DMA engines
- One solution is to use bounce buffers
  - Drivers like TI QSPI use bounce buffers
- SPI core has its own vmalloc buffer to sg_list mapping logic
  - Individual framework/drivers have own implementation
- Can DMA Mapping APIs be modified to map vmalloc'd for DMA for wider community benefit?
  - Provide bounce buffer, if mapping is not possible

References

- Various Flash datasheets: Micron, Spansion and Macronix
- JEDEC Standards: JESD216, JESD216A and JESD216B (www.jedec.org)
- http://www.linux-mtd.infradead.org/index.html
- MTD mailing list (linux-mtd@lists.infradead.org) archive.
- https://git.kernel.org/

Credits

- Texas Instruments Inc.
- The Linux Foundation

Q & A

Thank You!