ZEPHYR SD SUPPORT

PUBLIC

Daniel DeGrasse
Firmware Engineer, NXP Semiconductors

JUNE 8, 2022

SECURE CONNECTIONS
FOR A SMARTER WORLD

PUBLIC

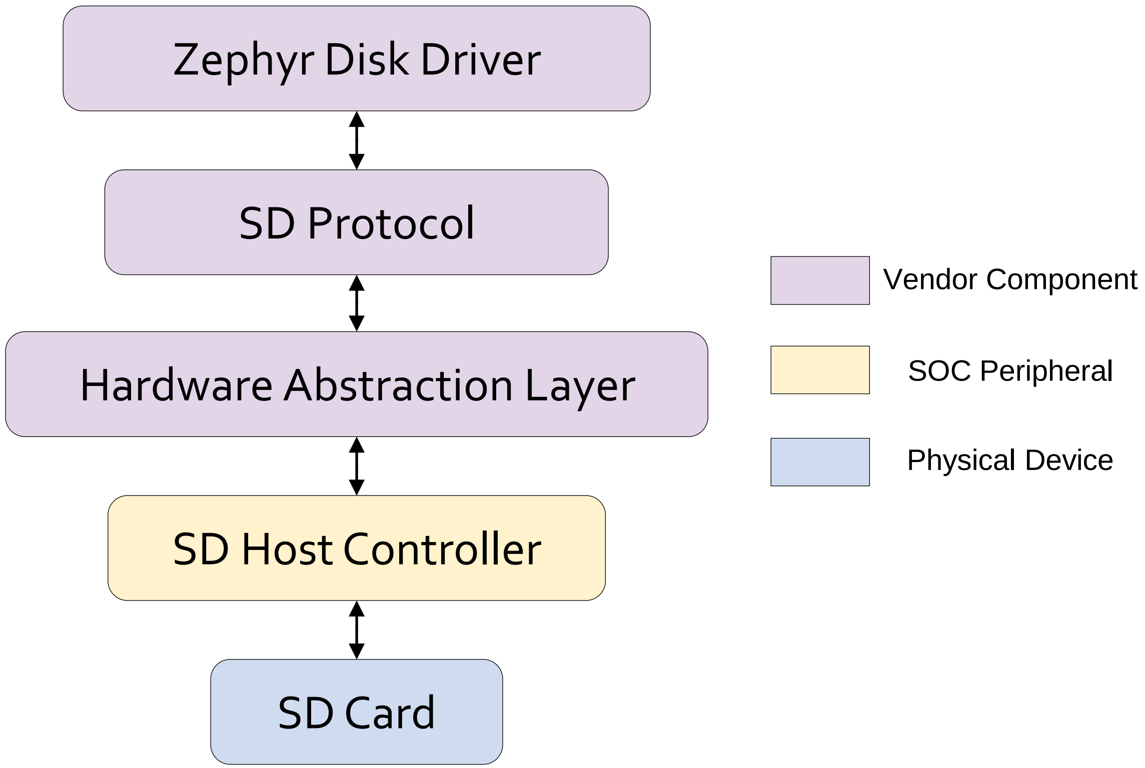
ABSTRACT

- Current stable API for SD devices
- Experimental SD host controller API
- SD protocol and host controller API overview
- Vendor implementation tips for SD host controllers
- Overview of changes required to use stack in application
- Future work on SD stack and host controller API

STABLE STATE OF SD SUPPORT

- All SD devices implement disk driver API
- Vendor Implements SD protocol stack
- Code size can be smaller
- No shared components
- Every vendor must maintain entire protocol stack
  - NXP Driver: ~3000 SLOC
- New SD host controller means new SD protocol stack
- No API available for SDIO support

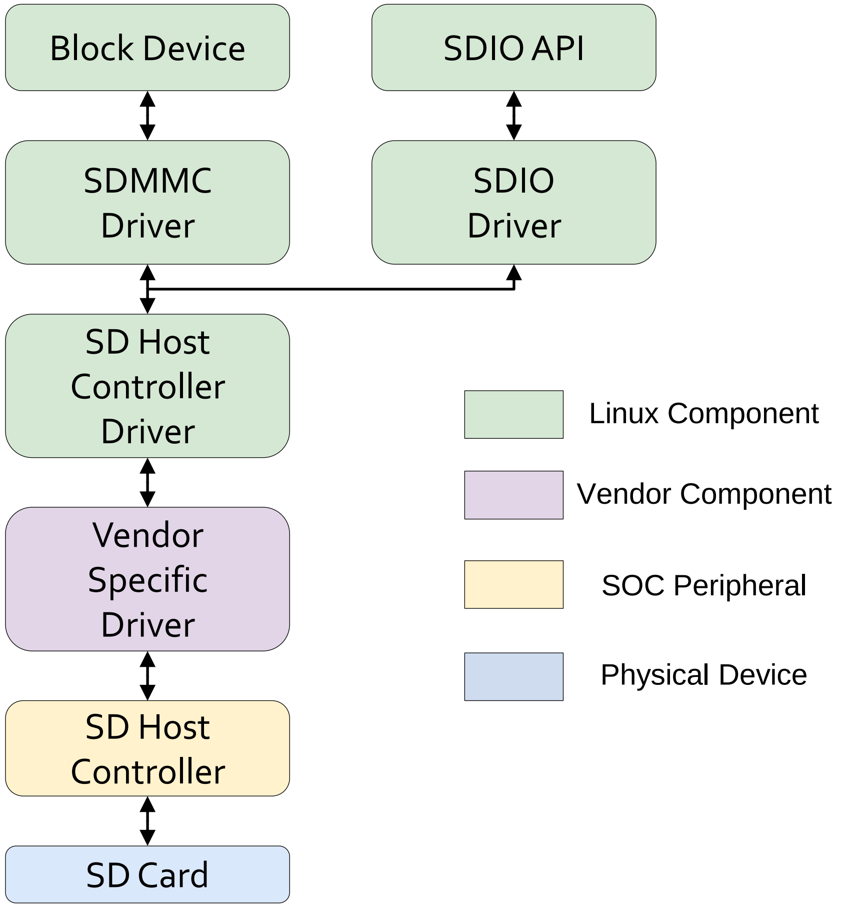
Stable SDMMC Architecture



LINUX STACK COMPARISON

- Generic API for SDIO devices, as well as SDMMC cards
- Shared generic protocol stack
- SDIO support built in
- Common API for all host controller drivers
- Much lower vendor requirements to support a new SD host controller

Linux SD Architecture



Block Device

SDIO API

SDMMC Driver

SDIO Driver

SD Host Controller Driver

Vendor Specific Driver

SD Host Controller

SD Card

🟩 Linux Component

🟪 Vendor Component

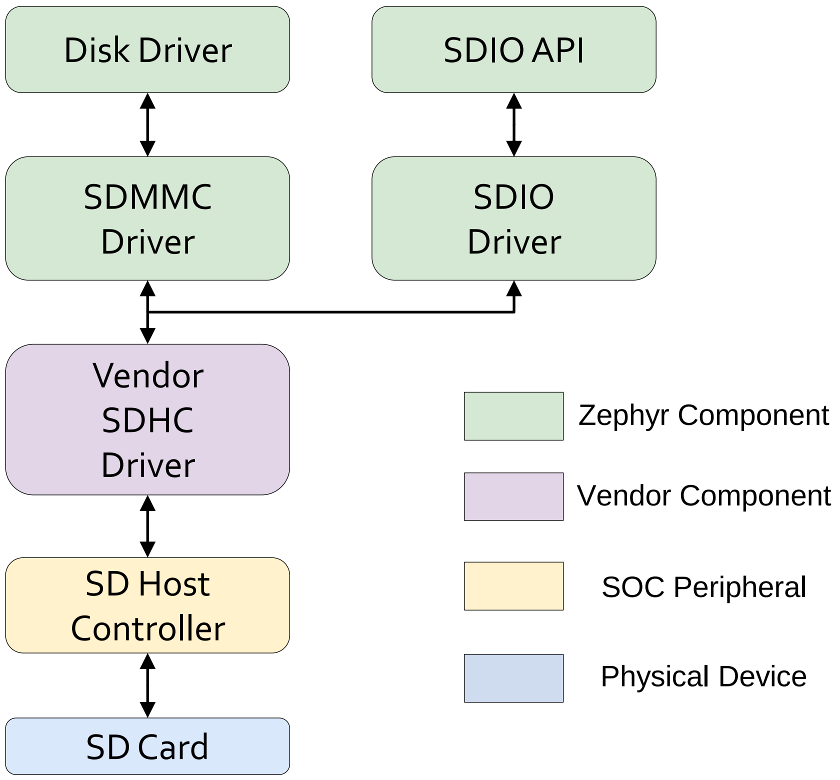
🟨 SOC Peripheral

🟦 Physical Device

EXPERIMENTAL ZEPHYR STACK

- Generic SD protocol stack
- API to interact with SD host controllers
- Vendors must implement SD host controller driver
- Reduces vendor implementation requirements
  - ~900 SLOC for NXP SDHC
- Provides clear path towards SDIO support in Zephyr
- Designed to provide benefits of a layered architecture

Experimental Zephyr Architecture



Disk Driver

SDIO API

SDMMC Driver

SDIO Driver

Vendor SDHC Driver

SD Host Controller

SD Card

🟩 Zephyr Component

🟪 Vendor Component

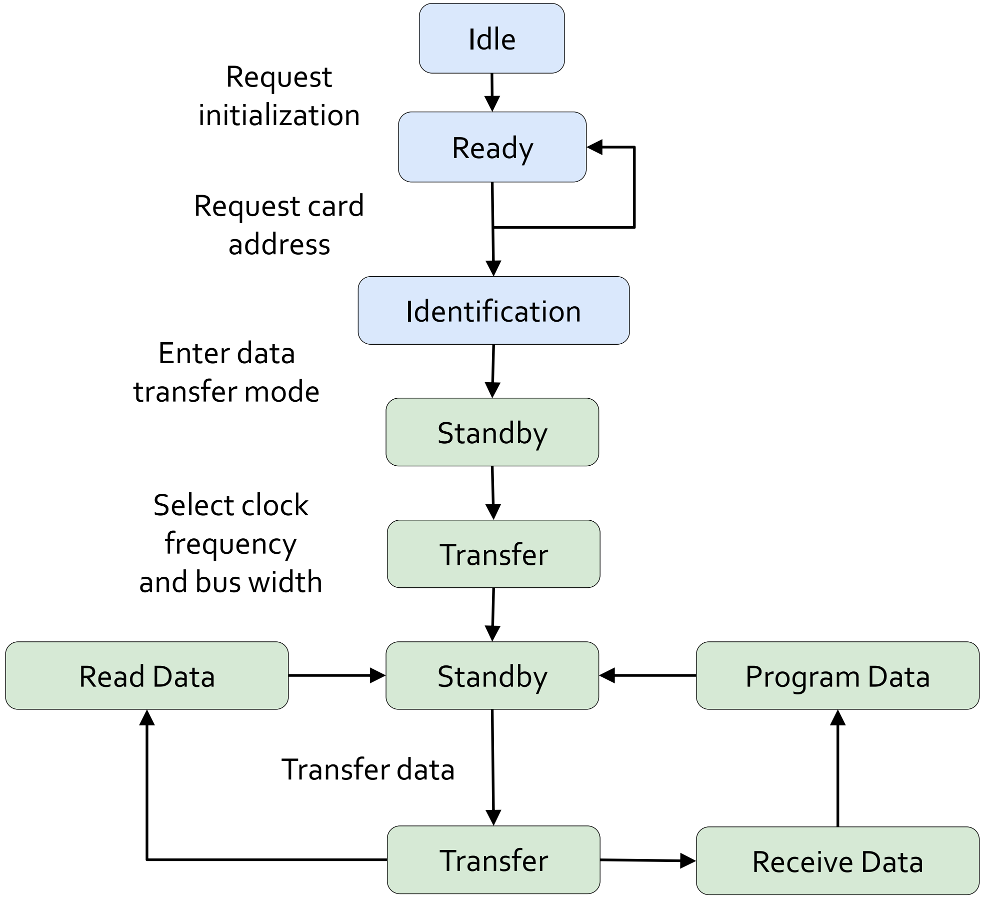
🟨 SOC Peripheral

🟦 Physical Device

SD PROTOCOL OVERVIEW

- Stateful protocol based around commands and responses
- Card is configured in initialization states
- Once in transfer state, card frequency and bus width can be raised
- Data transfer can only start from standby state

SDMMC Initialization Process



SD HOST CONTROLLER API

```
struct sdhc_driver_api {
        int (*reset)(const struct device *dev);
        int (*request)(const struct device *dev,
                struct sdhc_command *cmd,
                struct sdhc_data *data);
        int (*set_io)(const struct device *dev,
                struct sdhc_io *ios);
        int (*get_card_present)(const struct device *dev);
        int (*execute_tuning)(const struct device *dev);
        int (*card_busy)(const struct device *dev);
        int (*get_host_props)(const struct device *dev,
                struct sdhc_host_props *props);

};
```

```
struct sdhc_command {
        uint32_t    opcode;
        uint32_t    arg;
        uint32_t    response[4];
        uint32_t    response_type;
        uint32_t    retries;
        int         timeout_ms;
};
```

```
struct sdhc_data {
        unsigned int block_addr;
        unsigned int block_size;
        unsigned int blocks;
        unsigned int bytes_xfered;
        void         *data;
        int          timeout_ms
};
```

```
struct sdhc_io {
        enum sdhc_clock_speed   clock;
        enum sdhc_bus_mode      bus_mode;
        enum sdhc_power         power_mode;
        enum sdhc_bus_width     bus_width;
        enum sdhc_timing_mode   timing;
        enum sd_driver_type     driver_type;
        enum sd_voltage         signal_voltage;
};
```

SD HOST CONTROLLER IMPLEMENTATION

- Blocking API
- Use DMA/Interrupts where possible
- Use host capabilities field to control stack behavior
- Kconfig symbols can be used to compile out portions of stack
- CMD12 (stop transmission) and CMD23 (set block count) are responsibility of host controller
- API shall be thread safe
- CMD0 can be used to identify card initialization

CASE STUDY: LPC SDIF

- New driver in drivers/sdhc
  - PR 45447
  - Must select `SDHC_SUPPORTS_NATIVE_MODE`
  - Don’t select `SDHC_SUPPORTS_UHS` to reduce stack size
- Implement APIs based on complexity
  - `reset`
  - `get_host_props`
  - `set_io`
  - `get_card_present`
  - `request`
  - No `execute_tuning` implementation- no UHS support
- Testing
  - SDHC driver test
  - SDMMC subsystem test

SD SUBSYSTEM USAGE

- No user-facing changes required-SD subsystem integrates with disk driver API
- Binding like the following should be added under SD host controller implementing SDHC API
- Enables disk driver shim that uses SD subsystem as backend

```
&sdhc0 {
    mmc {
        compatible = "zephyr,sdmmc-disk";
        status = "okay";
        label = "SDMMC_0";
    };
}
```

FUTURE WORK

- SDIO support is planned using the SD host controller API
- Continued work on code size
  - Portions of stack can be compiled based on what host controller supports
- Improvements to SD host controller API
  - Callback for card insertion
- Potential to create opt-in generic SD host controller driver, like what Linux offers
- More vendor support
  - Supporting SD stack will allow vendors to leverage common stack, reduce support requirements, and enable SDIO on their platforms

IN SUMMARY

- New SD host controller API layer enables generic SD protocol stack
- SDIO support can leverage same host controller API
- Generic protocol reduces vendor support requirements
- Minimal application changes required- protocol stack is a drop-in replacement for stable disk driver API
- Vendors supporting stack will get all these benefits
- Additional Info
  - SD subsystem RFC
  - Reference Implementation of LPC SD host controller
  - SD card specification

SECURE CONNECTIONS
FOR A SMARTER WORLD

Backup Slides

SECURE CONNECTIONS
FOR A SMARTER WORLD

PUBLIC

PROPOSED SDIO API

- SDIO cards have multiple functions
- Card I/O occurs using a function number, and register address
- I/O can be performed using byte based or block-based transfers
- I/O can read/write to FIFO, or using increasing address
- Usage examples include WiFi or Bluetooth drivers

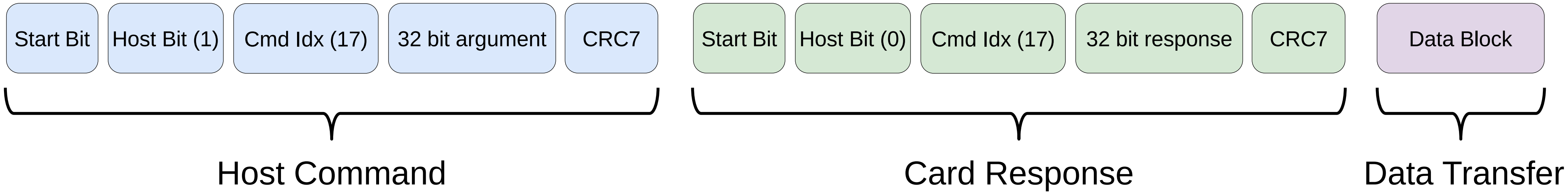
```
bool sd_is_card_present();

int sd_init()

int sdio_func_enable()

int sdio_fifo_read()

int sdio_fifo_write()

int sdio_read()

int sdio_write()

int sdio_read_byte()

int sdio_write_byte()
```

SD MEMORY CARD API

- Used by disk driver to implement a shim layer between SD subsystem and disk api
- Usage example:
  - Poll until sd card is present, then initialize it with sd_init
  - Check card properties using sdmmc_ioctl
  - Read and write data from card using api

```
bool sd_is_card_present(const struct device *sdhc_dev);

int sd_init(const struct device *sdhc_dev,
         struct sd_card *card);

int sdmmc_write_blocks(struct sd_card *card,
         const uint8_t *wbuf,
         uint32_t start_block,
         uint32_t num_blocks);

int sdmmc_read_blocks(struct sd_card *card,
         uint8_t *rbuf,
         uint32_t start_block,
         uint32_t num_blocks);

int sdmmc_ioctl(struct sd_card *card,
         uint8_t cmd,
         void *buf);
```

SD COMMAND EXAMPLE- SINGLE BLOCK READ



- Command IDX sent by host and card
- CRC7 used for error checking
- Response type varies based on command index
  - Response types have difference lengths and information
- Data block length depends on command index, and command argument
- Single block read will return up to 512 bytes, depending on the block size SD card has set.

ADDITIONAL API IMPROVEMENTS

- Async I/O/Callbacks
  - Disk I/O would need to be asynchronous as well
- Hot plugging
  - Broader Zephyr conversation- potentially compile in several SDIO device drivers at runtime
  - Disk subsystem does not support hot plug, but you can wait to mount the filesystem until the card is present