Introduction to UFS Subsystem

Mohammad Faiz Abbas Rizvi
ELC 2019
faiz_abbas@ti.com
About Me

• Software Engineer at Texas Instruments India since 2017.
• Part of the Linux Team that works on supporting various TI SoCs in mainline kernel and u-boot.
• I work on supporting peripheral drivers on TI SoCs, mainly MMC, CAN and now UFS.
• This presentation is a result of my experience adding the UFS subsystem to U-boot.
What is Universal Flash Storage?

It is primarily a Managed Flash.
Managed Flash

• A non-volatile flash array combined with a memory controller

• The controller manages the memory – bad block management, ECC, wear leveling etc.

• The controller communicates with the outside world with a defined communication protocol like MMC, SATA, SPI, Hyperflash etc.
UFS Features

• It’s a high performance serial interface designed for low power devices.
• Marketed as a replacement for SD card and eMMC for smartphones, ultra portable PCs and other embedded devices.
• Theoretical bidirectional full duplex transfer speeds up to 1.45 GBps
• Can be a removable card or embedded on the board

Image courtesy: dpreview.com
How is UFS better?

<table>
<thead>
<tr>
<th>Property</th>
<th>eMMC 5.1</th>
<th>UFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O speed</td>
<td>Upto 400 MBps</td>
<td>Upto 1.45 GBps</td>
</tr>
<tr>
<td>I/O Voltage</td>
<td>1.2V-3.3V</td>
<td>0.2V-0.4V differential</td>
</tr>
<tr>
<td>Random Read</td>
<td>13000 IOPS(^1)</td>
<td>68000 IOPS(^1)</td>
</tr>
<tr>
<td>Protocol</td>
<td>Half duplex</td>
<td>Full duplex</td>
</tr>
<tr>
<td>Theoretical Device Capacity</td>
<td>128 GB</td>
<td>16 TB</td>
</tr>
</tbody>
</table>

\(^1\) [androidcentral.com](https://www.androidcentral.com)
UFS System Overview

UFS Host

UFS Interconnect Layer
- MIPI Unipro
- MIPI M-PHY

UFS Transport Layer
- UFS Driver

UFS Application Layer
- SCSI Driver
- Application

UFS Device

LU-N
- MIPI Unipro
- MIPI M-PHY

LU-0

Configuration Structures

MIPI Unipro
MIPI M-PHY
UFS System Overview

UFS Host

- UFS Application Layer
  - Application
  - SCSI Driver
  - UFS Driver

- UFS Transport Layer
  - MIPI Unipro
  - MIPI M-PHY

- UFS Interconnect Layer
  - MIPI M-PHY

UFS Device

- LU-0
- LU-N
- Configuration Structures

- MIPI Unipro
- MIPI M-PHY
UFS Application Layer

- Consists of the UFS SCSI command set based on SCSI Architecture Model (SAM)
- Transactions take place with fixed length Command Descriptor Blocks (CBD)
- Each Transaction follows the I_T_L_Q nexus meaning the CDB needs to identify an Initiator communicating with a specific **LUN** in a **Target** with a specific **Query**.

### Example 6 byte CDB format

<table>
<thead>
<tr>
<th>Byte</th>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Operation code = 03h</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>LUN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>Allocation length</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Control</td>
</tr>
</tbody>
</table>

Example 6 byte CDB format
UFS SCSI Commands

• A command descriptor block can be of 6, 10 or 16 bytes of length
• Some important commands:
  – READ
  – WRITE
  – READ CAPACITY - get the size of a logical unit
  – REPORT LUNS - get list of all logical units
  – TEST UNIT READY - check if a logical unit is ready for accepting requests
  – START STOP UNIT – switch Power mode of the device
  – INQUIRY – More information about a logical unit
What is a Logical unit?

• Externally addressable
• Storage entity
• Internal Task Queue
• LUs inside a UFS device can be configured in variety of ways:
  • Amount of physical memory allocated to each LU
  • Write Protection
  • Boot
  • Memory Type (default, system code, non-persistent, enhanced)
  • Priority access
  • RPMB
Logical unit (continued)

- A UFS device can have a maximum of 32 logical units in it.
- In addition, there might be 4 Well Known Logical Units (W-LUNS).

- **REPORT LUNS**
  - Target for REPORT LUNS SCSI Request
  - Returns number and configuration of all the LUNs on the device

- **UFS Device**
  - Target for INQUIRY SCSI Request
  - Configuration, Power Control, Formatting, flags, attributes

- **Boot**
  - Used for boot operation
  - Can be 1 or 2 boot partitions but only one active one

- **RPMB**
  - Replay Protected memory block
  - Need security protocol to access data

<table>
<thead>
<tr>
<th>Well known logical unit</th>
<th>LUN</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORT LUNS</td>
<td>0x81</td>
</tr>
<tr>
<td>UFS Device</td>
<td>D0h</td>
</tr>
<tr>
<td>Boot</td>
<td>B0h</td>
</tr>
<tr>
<td>RPMB</td>
<td>C4h</td>
</tr>
</tbody>
</table>

Well-known LUNS
Application Layer Transaction View

SCSI Driver

- Report LUNS
- Response
- Read Capacity
- Response

\[ x \] \[ N \]

Device

- LU-0 /dev/sda
- LU-1 /dev/sdb
- LU-N /dev/sdX
UFS System Overview

UFS Host

UFS Interconnect Layer
- MIPI Unipro
- MIPI M-PHY

UFS Transport Layer
- UFS Driver

UFS Application Layer
- SCSI Driver
- Application

UFS Device

LU-N
- LU-0

Configuration Structures

MIPI Unipro
MIPI M-PHY
UFS Interconnect Layer

- MIPI-PHY standard defines the physical layer implementation.
- MIPI-Unipro standard defines the data link layer implementation.

<table>
<thead>
<tr>
<th>HS-GEAR</th>
<th>Maximum Datarate</th>
</tr>
</thead>
<tbody>
<tr>
<td>HS-GEAR1</td>
<td>182 MBps</td>
</tr>
<tr>
<td>HS-GEAR2</td>
<td>364 MBps</td>
</tr>
<tr>
<td>HS-GEAR3</td>
<td>728 MBps</td>
</tr>
<tr>
<td>HS-GEAR4</td>
<td>1457 MBps</td>
</tr>
</tbody>
</table>

UFS data rates

UFS signals
Simplified UFS Power Modes

Each of these functions implements a START STOP UNIT Command
UFS Transport Layer

• Transactions consist of packets called UFS Protocol Information Units (UPIU).
• The host is the **Initiator** and the device is the **Target**.
• There are different types of UPIUs for handling SCSI commands, data operations, task management operations, query operations etc.
• Each transaction consists of:
  – One COMMAND UPIU
  – Zero or more DATA IN or DATA OUT UPIU
  – RESPONSE UPIU
• Each UPIU contains a 12 bytes header followed by optional fields depending on the type of UPIU.
UFS Transport Layer

UPIU HEADER

General UPIU Structure

<table>
<thead>
<tr>
<th>DW0</th>
<th>DW1</th>
<th>DW2</th>
<th>DW3</th>
<th>DW4</th>
<th>DWx</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Extra Header Segment 0</td>
<td>Extra Header Segment N</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Header E2ECRC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Data Segment 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Data Segment K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Data E2ECRC</td>
</tr>
</tbody>
</table>

Transaction Type | Flags | LUN | Task Tag |
-----------------|-------|-----|----------|
Total EHS Length | Device Information | Data Segment Length |
Reserved | Command Set Type | Query Function/Task Management Function | Response | Status |
Transaction Specification Fields
UPIU Structure

<table>
<thead>
<tr>
<th>Transaction Type</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Flags</td>
<td></td>
</tr>
<tr>
<td>LUN</td>
<td></td>
</tr>
<tr>
<td>Task Tag</td>
<td></td>
</tr>
<tr>
<td>Initiator ID</td>
<td></td>
</tr>
<tr>
<td>Command Set Type</td>
<td></td>
</tr>
<tr>
<td>Query/Task Manag. Fn</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td></td>
</tr>
<tr>
<td>Total length</td>
<td></td>
</tr>
<tr>
<td>Device Info</td>
<td></td>
</tr>
<tr>
<td>Data Length</td>
<td></td>
</tr>
<tr>
<td>UPIU Header</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initiator</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP OUT</td>
<td>NOP IN</td>
</tr>
<tr>
<td>Command</td>
<td>Response</td>
</tr>
<tr>
<td>Data Out</td>
<td>Data In</td>
</tr>
<tr>
<td>-</td>
<td>Ready to Transfer</td>
</tr>
<tr>
<td>Query Request</td>
<td>Query Response</td>
</tr>
<tr>
<td>-</td>
<td>Reject</td>
</tr>
</tbody>
</table>

Types of UPIUs
UTP Read Transaction

UAP

Initiator

Target

READ CDB

Command UPIU

Data In UPIU

Data In UPIU

Response UPIU
UTP Write Transaction

- UAP
- Initiator
- Target

WRITE CDB

Command UPIU

Ready to Transfer UPIU

Data Out UPIU

Ready to Transfer UPIU

Data Out UPIU

Response UPIU
UTP Host Memory Configuration

```c
struct utp_transfer_req_desc {
    /* DW 0-3 */
    struct request_desc_header header;

    /* DW 4-5 */
    __le32 command_desc_base_addr_lo;
    __le32 command_desc_base_addr_hi;

    /* DW 6 */
    __le16 response_upiu_length;
    __le16 response_upiu_offset;

    /* DW 7 */
    __le16 prd_table_length;
    __le16 prd_table_offset;
};
```

- **UTP Transfer Request Descriptor**
- **Command UPIU**
- **Response UPIU**
- **PRDT**
- **Data Buffer**

```c
struct utp_transfer_cmd_desc {
    u8 command_upiu[ALIGNED_UPIU_SIZE];
    u8 response_upiu[ALIGNED_UPIU_SIZE];
    struct ufshcd_sg_entry prd_table[SG_ALL];
};
```
UFS Host Memory Configuration

Transaction Type
Flags
LUN
Task Tag
Initiator ID
Command Set Type
Query/Task Manag. Fn
Response
Status
Total length
Device Info
Data Length

**UPIU Header**

- **Transaction Type**
- **Flags**
- **LUN**
- **Task Tag**
- **Initiator ID**
- **Command Set Type**
- **Query/Task Manag. Fn**
- **Response**
- **Status**
- **Total length**
- **Device Info**
- **Data Length**

**UTP Transfer Request Descriptor**

- **0**
  - Command UPIU
  - Response UPIU
  - PRDT
  - Data Buffer

- **1**
  - Command UPIU
  - Response UPIU
  - PRDT
  - Data Buffer

- **31**
  - Command UPIU
  - Response UPIU
  - PRDT
  - Data Buffer
UFS Task Management

<table>
<thead>
<tr>
<th>Transaction Type</th>
<th>Flags</th>
<th>LUN</th>
<th>Task Tag</th>
<th>Initiator ID</th>
<th>Command Set Type</th>
<th>Query/Task Manag. Fn</th>
<th>Response</th>
<th>Status</th>
<th>Total length</th>
<th>Device Info</th>
<th>Data Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**UPIU Header**

```
struct utp_task_req_desc {
    /* Dw 0-3 */
    struct request_desc_header header;

    /* Dw 4-11 - Task request UPIU structure */
    struct utp_uapiu_header req_header;
    _be32        input_param1;
    _be32        input_param2;
    _be32        input_param3;
    _be32        __reserved1[2];

    /* Dw 12-19 - Task Management Response UPIU structure */
    struct utp_uapiu_header rsp_header;
    _be32        output_param1;
    _be32        output_param2;
    _be32        __reserved2[3];
};

/* Task management functions */
enum {
    UFS_ABORT_TASK     = 0x01,
    UFS_ABORT_TASK_SET = 0x02,
    UFS_CLEAR_TASK_SET = 0x04,
    UFS_LOGICAL_RESET  = 0x08,
    UFS_QUERY_TASK     = 0x80,
    UFS_QUERY_TASK_SET = 0x81,
};
```
struct ufs_hba {
    void __iomem *mmio_base;

    /* Virtual memory reference */
    struct utp_transfer_cmd_desc *ucdl_base_addr;
    struct utp_transfer_req_desc *utrdl_base_addr;
    struct utp_task_req_desc *utmrdl_base_addr;

    ...
UFS Host Controller Interface

• Host Capabilities Registers: Version, Vendor ID, 64 bit support, number of slots etc.

• Operation and Runtime Registers: Interrupt handling (including aggregator), error status at each layer, host controller status etc.

• UTP Transfer Registers: Base addresses of UTPTRD table and UTPCMD table, doorbell registers, complete notifier etc.

• Task Management Registers: Base address of UTPTM table, doorbell, complete notifiers etc.

• UIC Command Registers: Interconnect layer command register and arguments.
Query Request

- Query request UPIU is used to access information about configuration, enumeration, device descriptors, flags, attributes etc.

<table>
<thead>
<tr>
<th>Query Function</th>
<th>Query OPCODE</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any value</td>
<td>00h</td>
<td>NOP</td>
</tr>
<tr>
<td>Std. Read</td>
<td>01h</td>
<td>Read Descriptor</td>
</tr>
<tr>
<td>Std. Write</td>
<td>02h</td>
<td>Write Descriptor</td>
</tr>
<tr>
<td>Std. Read</td>
<td>03h</td>
<td>Read Attribute</td>
</tr>
<tr>
<td>Std. Write</td>
<td>04h</td>
<td>Write Attribute</td>
</tr>
<tr>
<td>Std. Read</td>
<td>05h</td>
<td>Read Flag</td>
</tr>
<tr>
<td>Std. Write</td>
<td>06h</td>
<td>Set Flag</td>
</tr>
<tr>
<td>Std. Write</td>
<td>07h</td>
<td>Clear Flag</td>
</tr>
<tr>
<td>Std. Write</td>
<td>08h</td>
<td>Toggle Flag</td>
</tr>
</tbody>
</table>

Query functions
UFS Descriptors, Flags and Attributes

UFS Descriptor

• Independently addressable data structure describing something about the device

• Mostly read only except Configuration Descriptor

<table>
<thead>
<tr>
<th>Query Function</th>
<th>Query OPCODE</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any value</td>
<td>00h</td>
<td>NOP</td>
</tr>
<tr>
<td>Std. Read</td>
<td>01h</td>
<td>Read Descriptor</td>
</tr>
<tr>
<td>Std. Write</td>
<td>02h</td>
<td>Write Descriptor</td>
</tr>
</tbody>
</table>

Query UPIU opcode

<table>
<thead>
<tr>
<th>Descriptor IDN</th>
<th>Descriptor Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>Device</td>
</tr>
<tr>
<td>01h</td>
<td>Configuration</td>
</tr>
<tr>
<td>02h</td>
<td>Unit</td>
</tr>
<tr>
<td>03h</td>
<td>-</td>
</tr>
<tr>
<td>04h</td>
<td>Interconnect</td>
</tr>
<tr>
<td>05h</td>
<td>String</td>
</tr>
<tr>
<td>06</td>
<td>-</td>
</tr>
<tr>
<td>07h</td>
<td>Geometry</td>
</tr>
<tr>
<td>08h</td>
<td>Power</td>
</tr>
<tr>
<td>09h</td>
<td>Device Health</td>
</tr>
</tbody>
</table>

Types of Descriptors
UFS Descriptors, Flags and Attributes

UFS Flags

- A single Boolean value that can be set On or Off. Can be set, cleared or toggled.

<table>
<thead>
<tr>
<th>Flag IDN</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>01h</td>
<td>fDeviceInit</td>
</tr>
<tr>
<td>02h</td>
<td>fPermanentWPEn</td>
</tr>
<tr>
<td>03h</td>
<td>fPowerOnWPEn</td>
</tr>
<tr>
<td>04h</td>
<td>fBackgroundOpsEn</td>
</tr>
<tr>
<td>05h</td>
<td>fDeviceLifeSpanModeEn</td>
</tr>
<tr>
<td>06h</td>
<td>fPurgeEnable</td>
</tr>
<tr>
<td>07h</td>
<td>fRefreshEnable</td>
</tr>
<tr>
<td>08h</td>
<td>fPhyResourceRemoval</td>
</tr>
<tr>
<td>09h</td>
<td>fBusyRTC</td>
</tr>
<tr>
<td>0Bh</td>
<td>fPermanentlyDisableFwUpdate</td>
</tr>
</tbody>
</table>

Types of Descriptors

<table>
<thead>
<tr>
<th>Query Function</th>
<th>Query OPCODE</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Std. Read</td>
<td>05h</td>
<td>Read Flag</td>
</tr>
<tr>
<td>Std. Write</td>
<td>06h</td>
<td>Set Flag</td>
</tr>
<tr>
<td>Std. Write</td>
<td>07h</td>
<td>Clear Flag</td>
</tr>
<tr>
<td>Std. Write</td>
<td>08h</td>
<td>Toggle Flag</td>
</tr>
</tbody>
</table>

Query UPIU opcode
UFS Attributes

- A single parameter that represents a specific range of numerical values that can be set or read.

<table>
<thead>
<tr>
<th>Query Function</th>
<th>Query OPCODE</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Std. Read</td>
<td>03h</td>
<td>Read Attribute</td>
</tr>
<tr>
<td>Std. Write</td>
<td>04h</td>
<td>Write Attribute</td>
</tr>
</tbody>
</table>

Types of Descriptors

<table>
<thead>
<tr>
<th>Attribute IDN</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>bBootLunEn</td>
</tr>
<tr>
<td>02h</td>
<td>bCurrentPowerMode</td>
</tr>
<tr>
<td>03h</td>
<td>bActiveICCLevel</td>
</tr>
<tr>
<td>04h</td>
<td>bOutOfOrderDataEn</td>
</tr>
<tr>
<td>05h</td>
<td>bBackgroundOpStatus</td>
</tr>
<tr>
<td>06h</td>
<td>bPurgeStatus</td>
</tr>
<tr>
<td>07h</td>
<td>bMaxDataInSize</td>
</tr>
<tr>
<td>08h</td>
<td>bMaxDataOutSize</td>
</tr>
<tr>
<td>09h</td>
<td>dDynCapNeeded</td>
</tr>
<tr>
<td>0Ah</td>
<td>bRefClkFreq</td>
</tr>
</tbody>
</table>
Kernel Implementation

• All the drivers can be found under drivers/scsi/ufs/

• Documentation/scsi/ufs.txt – for information about device enumeration steps

• Documentation/devicetree/bindings/ufs/ - for device tree node implementation

• ufshcd_pltrfm_init() : Call this from your probe()
Kernel Implementation

[ 279.213830] cdns-ufshcd 4e84000.ufs: ufshcd_populate_vreg: Unable to find vdd-hba-supply regulator, assuming enabled
[ 279.224369] cdns-ufshcd 4e84000.ufs: ufshcd_populate_vreg: Unable to find vcc-supply regulator, assuming enabled
[ 279.234554] cdns-ufshcd 4e84000.ufs: ufshcd_populate_vreg: Unable to find vccq-supply regulator, assuming enabled
[ 279.244889] cdns-ufshcd 4e84000.ufs: ufshcd_populate_vreg: Unable to find vccq2-supply regulator, assuming enabled
[ 279.256684] scsi host1: ufshcd
[ 279.288184] cdns-ufshcd 4e84000.ufs: ufshcd_print_pwr_info:[RX, TX]: gear=[1, 1], lane[1, 1], pwr[SLOWSLOWAUTO_MODE, SLOWSLOWAUTO_MODE], rate = 0
[ 279.325695] cdns-ufshcd 4e84000.ufs: ufshcd_print_pwr_info:[RX, TX]: gear=[3, 3], lane[2, 2], pwr[FAST_MODE, FAST MODE], rate = 2
[ 279.337271] cdns-ufshcd 4e84000.ufs: ufshcd_find_max_sup_active_icc_level: Regulator capability was not set, actvIccLevel=0
[ 279.340037] scsi 1:0:0:0:49488: Well-known LUN TOSHIBA THGAFOG8T23BAILB 0300 PQ: 0 ANSI: 6
[ 279.357908] scsi 1:0:0:0:49476: Well-known LUN TOSHIBA THGAFOG8T23BAILB 0300 PQ: 0 ANSI: 6
[ 279.367150] cdns-ufshcd 4e84000.ufs: ufshcd_scsi_add_wluns: BOOT WLUN not found
[ 279.375452] scsi 1:0:0:0: Direct-Access TOSHIBA THGAFOG8T23BAILB 0300 PQ: 0 ANSI: 6
[ 279.383906] sd 1:0:0:0: Power-on or device reset occurred
[ 279.390310] sd 1:0:0:0: [sda] 8192 4096-byte logical blocks: (33.6 MB/32.0 MiB)
[ 279.397748] sd 1:0:0:0: [sda] Write Protect is off
[ 279.398383] scsi 1:0:0:1: Direct-Access TOSHIBA THGAFOG8T23BAILB 0300 PQ: 0 ANSI: 6
[ 279.410705] sd 1:0:0:0: [sda] Write cache: enabled, read cache: enabled, supports DPO and FUA
[ 279.419347] sd 1:0:0:0: [sda] Optimal transfer size 65536 bytes
[ 279.427141] sd 1:0:0:1: Power-on or device reset occurred
[ 279.432820] sd 1:0:0:1: [sdb] 7808000 4096-byte logical blocks: (32.0 GB/29.8 GiB)
[ 279.440598] sd 1:0:0:1: [sdb] Write Protect is off
[ 279.445666] sd 1:0:0:1: [sdb] Write cache: enabled, read cache: enabled, supports DPO and FUA
[ 279.454740] sd 1:0:0:1: [sdb] Optimal transfer size 65536 bytes
[ 279.460876] sd 1:0:0:0: [sda] Attached SCSI disk
[ 279.469225] sdb: sdb1
[ 279.473882] sd 1:0:0:1: [sdb] Attached SCSI disk
Configuring the device from user space

• UFS appears as a SCSI generic block device “/dev/bsg/ufs-bsg” that can accept an ioctl system call.

• User space can allocate `struct sg_io_v4` and send in the ioctl system call with request code SG_IO.

    int ioctl(int fd, SG_IO, struct sg_io_v4 *)
ufs-utils

https://github.com/westerndigitalcorporation/ufs-utils

ufs-utils < desc | attr | fl | err_hist | uic >

• Examples:
  – Read Configuration descriptor
    ufs-utils desc -t 1 -p /dev/bsg/ufs-bsg

  – Set Background operations flag
    ufs-utils fl -t 4 -e -p /dev/bsg/ufs-bsg
U-boot Implementation

• Available in 2020.01 release (got merged this Thursday!).
• drivers/ufs/ufs.c
• cmd/ufs.c
  – Contains:
    • basic command to initialize the ufs device, detect all LUNs and register them as scsi devices.
  – Does not contain:
    • commands to access/configure descriptors/flags/attributes

Patches Welcome 😊
References

• Universal Flash Storage 3.0

• UFS Host Controller Interface (UFSHCI) version 3.0

• https://git.kernel.org

Thank You