Agenda

• Technical Backgrounds
• Introduce HPB
• Deeper Implementation
• Performance Improvement
• Current Status
• Conclusion
Technical Backgrounds:
Brief Information about UFS
What is UFS?

- **Universal Flash Storage (UFS)**
  - Simple, high performance, mass storage device with a serial interface
  - Provides Higher performance & Lower power consumption
  - Widely used in commercial embedded products (e.g., smartphone)

- **Linux UFS Subsystem**
  - Subsystem that supports UFS storage devices
  - Implemented under the SCSI Subsystem
### Performance: eMMC vs UFS

**• Performance Comparison with eMMC**

<table>
<thead>
<tr>
<th>Products</th>
<th>Interface</th>
<th>Sequential Read</th>
<th>Sequential Write</th>
<th>Random Read</th>
<th>Rand Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>eMMC</td>
<td>eMMC 4.5</td>
<td>140MB/s</td>
<td>50MB/s</td>
<td>7,000 IOPS</td>
<td>2,000 IOPS</td>
</tr>
<tr>
<td></td>
<td>eMMC 5.0</td>
<td>250MB/s</td>
<td>90MB/s</td>
<td>7,000 IOPS</td>
<td>7,000 IOPS</td>
</tr>
<tr>
<td></td>
<td>eMMC 5.1</td>
<td>250MB/s</td>
<td>125MB/s</td>
<td>11,000 IOPS</td>
<td>13,000 IOPS</td>
</tr>
<tr>
<td>UFS</td>
<td>128GB eUFS 2.0</td>
<td>350MB/s</td>
<td>150MB/s</td>
<td>19,000 IOPS</td>
<td>14,000 IOPS</td>
</tr>
<tr>
<td></td>
<td>256GB eUFS 2.0</td>
<td>850MB/s</td>
<td>260MB/s</td>
<td>45,000 IOPS</td>
<td>40,000 IOPS</td>
</tr>
<tr>
<td></td>
<td>256GB UFS Card</td>
<td>530MB/s</td>
<td>170MB/s</td>
<td>40,000 IOPS</td>
<td>35,000 IOPS</td>
</tr>
<tr>
<td></td>
<td>512GB eUFS 2.1</td>
<td>860MB/s</td>
<td>255MB/s</td>
<td>42,000 IOPS</td>
<td>42,000 IOPS</td>
</tr>
<tr>
<td></td>
<td>1TB eUFS 2.1</td>
<td>1000MB/s</td>
<td>260MB/s</td>
<td>58,000 IOPS</td>
<td>50,000 IOPS</td>
</tr>
<tr>
<td></td>
<td>512GB eUFS 3.1</td>
<td>2100MB/s</td>
<td>410MB/s</td>
<td>63,000 IOPS</td>
<td>68,000 IOPS</td>
</tr>
</tbody>
</table>

Architectural Overview

- **UFS Top Level Architecture**
  - UFS communication is a layered communication architecture
  - Based of SCSI SAM architectural model [SAM]

*JESD220E UFS3.1, JEDEC*
Architectural Overview

- UFS System Model

* JESD220E UFS3.1, JEDEC
UFS Subsystem

- **UFS Subsystem Implementation**
  - UFS is implemented under SCSI layer
  - UFS driver uses the SCSI command set

- **Location of UFS Subsystem in Linux**
  - UFS directory: drivers/scsi/ufs/
  - Core drivers: drivers/scsi/ufshcd.{ch}
  - Platforms: drivers/scsi/ufshcd-pltfrm.{ch}
  - Controller specifics: driver/scsi/ufs-*.{ch}
    - e.g., ufs-exynos.{ch}
Transactions in UFS

- **UFS Transport Protocol (UTP) Layer**
  - UTP uses a SCSI Architectural model (SAM)
  - **Client-Server** or **Request-Response** model
  - UFS transactions consist of packets called **UFS Protocol Information Unit (UPIU)**

![SAM client-server model diagram]

- Host System
- Target Device
- **Device Service Request**
- **Device Service Response**
- **Task Management Request**
- **Task Management Response**
Transactions in UFS

- **UFS Protocol Information Unit (UPIU)**
  - All UPIU consists of a single constant 12 bytes header segment
  - Transaction specific segment
  - Possibly one or more extended header segments
  - Zero or more data segments

---

**Table 10.1 — UPIU Transaction Codes**

<table>
<thead>
<tr>
<th>Initiator To Target</th>
<th>Transaction Code</th>
<th>Target to Initiator</th>
<th>Transaction Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP OUT</td>
<td>00 0000b</td>
<td>NOP IN</td>
<td>10 0000b</td>
</tr>
<tr>
<td>COMMAND</td>
<td>00 0001b</td>
<td>RESPONSE</td>
<td>10 0001b</td>
</tr>
<tr>
<td>DATA OUT</td>
<td>00 0010b</td>
<td>DATA IN</td>
<td>10 0010b</td>
</tr>
<tr>
<td>TASK MANAGEMENT REQUEST</td>
<td>00 0100b</td>
<td>TASK MANAGEMENT RESPONSE</td>
<td>10 0100b</td>
</tr>
<tr>
<td>Reserved</td>
<td>01 0001b</td>
<td>READY TO TRANSFER</td>
<td>11 0001b</td>
</tr>
<tr>
<td>QUERY REQUEST</td>
<td>01 0110b</td>
<td>QUERY RESPONSE</td>
<td>11 0110b</td>
</tr>
<tr>
<td>Reserved</td>
<td>01 1111b</td>
<td>REJECT UPIU</td>
<td>11 1111b</td>
</tr>
<tr>
<td>Reserved</td>
<td>Others</td>
<td>Reserved</td>
<td>Others</td>
</tr>
</tbody>
</table>

*NOTE 1: Bit 5 of the Transaction Code indicates the direction of flow and the originator of the UPIU: when equal '0' the originator is the Initiator device, when equal '1' the originator is the Target device.*

* JESD220E UFS3.1, JEDEC
Scope of Improvement: The Read Latency

• UFS is a Flash Memory Storage
  – NAND Flash device uses Flash Translation Layer (FTL)
  – To translate logical address of I/O request to flash memory physical address
  – Logical → Physical (L2P) mapping entries are managed by FTL
  – Device must read the mapping entry first to reach the actual data
  – This mapping table is maintained in the NAND flash memory.

• The ‘Read Latency’
  – Normally UFS devices are having SRAM to cache these entries
  – Because of high cost of SRAM, it only save partial data of whole entries
Scope of Improvement: The Read Latency

- Read Latency of UFS Device

- Flash memory access latency doubled
  - when the mapping entries are not cached in SRAM

Diagram:
- Host System
  - Host Controller Interface
  - CPU + Logic
  - NAND Flash memory

Steps:
1. Fetch read command
2. Request L2P entry
3. Read L2P entry
4. Request user data
5. Transfer user data

Latency:
- tR(map)
- tR(data)

Time: t

Legend:
- READ requested
- READ completed

Note:
- UFS Device
- Flash memory access latency doubled
- Mapping entries not cached in SRAM
Introduce HPB:
What is the Host Performance Booster?
What is HPB?

The Host Performance Booster is an Extension Feature of UFS Subsystem:
to Improve the overall performance through reducing the read latency

* UNIVERSAL FLASH STORAGE (UFS) HOST PERFORMANCE BOOSTER (HPB) EXTENSION, VERSION 2.0, JEDEC
Concept of HPB

What exactly HPB does?
- HPB caches the L2P mapping entries in the Host memory
- The capacity of Host memory is **BIG enough** to save the mapping entries
- Accessing the Host memory is a lot faster than accessing NAND

\[
8B \times 33,554,432 \text{ blocks} = 256\text{MB} \\
\]

256MB needed for whole 128GB mapping entries

* Assuming the logical block size of UFS device is 4KB
Concept of HPB

- Read Latency of UFS Device with HPB

When the mapping entries are not cached in SRAM, read latency reduced.

- Read Cached L2P entry
- Fetch read command
- Request L2P entry
- Read L2P entry
- Request user data
- Transfer user data

READ requested

(0) Read Cached L2P entry
(1) Fetch read command
(2) Request L2P entry
(3) Read L2P entry
(4) Request user data
(5) Transfer user data

READ completed
Overall Behavior

- Caching Requested

The HPB data structure is used to map logical block addresses (LBAs) to physical page numbers (PPNs). Each HPB Region consists of PPNs for contiguous LBAs so that the host can lookup PPN for the requesting LBA from the host memory.

- HPB Entry Table (LBA #0~)
  - Index
  - PPN

- HPB Entry Table (LBA #100~)
  - Index
  - PPN

- HPB Region Lookup Table
  - HPB Region #
  - State
    - 0: valid
    - 1: invalid
    - 2: invalid
    - ...

Host memory

- HPB Region 0
- HPB Region 1

System Area (Available to OS)

File system

HPB Host driver

UFS booting

UFS Device

Any SCSI cmd

Command

Response

Each HPB Region consists of PPNs for contiguous LBAs so that the host can lookup PPN for the requesting LBA from the host memory.
Overall Behavior

- Caching Mapping Entries

Host memory

HPB Region 0
HPB Region 1

System Area (Available to OS)

HPB data structure (Host driver)

<table>
<thead>
<tr>
<th>Index</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x12347134</td>
</tr>
<tr>
<td>1</td>
<td>0x73915734</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

HPB Entry Table (LBA #0~)

<table>
<thead>
<tr>
<th>Index</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Each HPB Region consists of PPNs for contiguous LBAs so that host can lookup PPN for the requesting LBA from the host memory.

HPB Region Lookup Table

<table>
<thead>
<tr>
<th>HPB Region #</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>valid</td>
</tr>
<tr>
<td>1</td>
<td>invalid</td>
</tr>
<tr>
<td>2</td>
<td>invalid</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

File system

HPB Host driver

UFS booting

UFS Device

Any SCSI cmd

Command

Response

Request mapping entries

SAMSUNG
THE LINUX FOUNDATION
Overall Behavior

- Caching Mapping Entries

### HPB Data Structure (Host Driver)

<table>
<thead>
<tr>
<th>Index</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x12347134</td>
</tr>
<tr>
<td>1</td>
<td>0x73915734</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HPB Region Lookup Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPB Region #</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Each HPB Region consists of PPNs for contiguous LBAs so that host can lookup PPN for the requesting LBA from the host memory.
Overall Behavior

• Caching Mapping Entries

- **Host memory**
  - HPB Region 0
  - HPB Region 1

- **System Area (Available to OS)**

---

### HPB data structure (Host driver)

- **HPB Entry Table (LBA #0~)**
  - Index | PPN
  - 0     | 0x12347134
  - 1     | 0x73915734
  - ...

- **HPB Entry Table (LBA #100~)**
  - Index | PPN
  - 0     | 0x13472915
  - 1     | 0x91741312
  - ...

Each HPB Region consists of PPNs for contiguous LBAs so that host can lookup PPN for the requesting LBA from the host memory.

---

**HPB Host driver**

- **HPB Region Lookup Table**
  - HPB Region # | State
  - 0           | valid
  - 1           | **valid**
  - 2           | invalid
  - ...

---

**UFS Device**

- **UFS booting**
  - Any SCSI cmd
  - Command
  - Response
  - Request mapping entries
  - Mapping entries

---

Now, HPB host driver can use HPB Read CMD using HPB data structure.
Overall Behavior

• Sending Command

Host memory

<table>
<thead>
<tr>
<th>HPB Region 0</th>
<th>HPB Region 1</th>
</tr>
</thead>
</table>

System Area (Available to OS)

HPB data structure (Host driver)

<table>
<thead>
<tr>
<th>HPB Entry Table (LBA #0~)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HPB Entry Table (LBA #100~)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

HPB Region Lookup Table

<table>
<thead>
<tr>
<th>HPB Region #</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>valid</td>
</tr>
<tr>
<td>1</td>
<td>valid</td>
</tr>
<tr>
<td>2</td>
<td>invalid</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Each HPB Region consists of PPNs for contiguous LBAs so that host can lookup PPN for the requesting LBA from the host memory.
Overall Behavior

- Sending Command

<table>
<thead>
<tr>
<th>Host memory</th>
<th>HPB Region 0</th>
<th>HPB Region 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Area (Available to OS)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

HPB data structure (Host driver)

<table>
<thead>
<tr>
<th>Index</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x12347134</td>
</tr>
<tr>
<td>1</td>
<td>0x73915734</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

HPB Entry Table (LBA #0~)

<table>
<thead>
<tr>
<th>Index</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x13472915</td>
</tr>
<tr>
<td>1</td>
<td>0x91741312</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Each HPB Region consists of PPNs for contiguous LBAs so that the host can lookup PPN for the requesting LBA from the host memory.

HPB Region Lookup Table

<table>
<thead>
<tr>
<th>HPB Region #</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>valid</td>
</tr>
<tr>
<td>1</td>
<td>valid</td>
</tr>
<tr>
<td>2</td>
<td>invalid</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Host memory

- HPB data structure
- HPB Region Lookup Table

File system

- HPB Host driver initialization
- Any SCSI cmd
- Command
- Response
- Loops
- Request mapping entries
- Mapping entries

UFS Device

- HPB READ (LBA #101) with PPN (PPN #91741312)
- HPB host driver

UFS booting
Overall Behavior

- Sending Command

Host memory

<table>
<thead>
<tr>
<th>HPB Region 0</th>
<th>Host memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPB Region 1</td>
<td>HPB data structure (Host driver)</td>
</tr>
<tr>
<td>System Area (Available to OS)</td>
<td></td>
</tr>
</tbody>
</table>

Each HPB Region consists of PPNs for contiguous LBAs so that host can lookup PPN for the requesting LBA from the host memory.
Deeper Implementation:
For further understanding
Data Structure

**UFSHCDC**
- ufs_hba
  - mmio_base
  - ...
- ufshpb_dev_info
  - num_lu
  - rgn_size
  - srgn_size
  - ...

**UFSPHPB**
- ufshpb_dev_info
  - num_lu
  - rgn_size
  - srgn_size
  - ...
- ufshpb_map_ctx
  - *m_page
  - *ppn_dirty
- ufshpb_lu
  - lun
  - *sdev_ufs_lu
  - rgn_state_lock
  - *rgn_tbl
  - hpb_state
  - rsp_list_lock
  - lh_act_srgn
  - lh_inact_rgn
  - ...
  - map_work
  - lru_info
  - ...
- ufshpb_region
  - *hpb
  - srgn_tbl
  - rgn_ida
  - rgn_idx
  - list_inact_rgn
  - list_lru_rgn
  - rgn_flags
- ufshpb_subregion
  - *mctx
  - srgn_state
  - rgn_ida
  - rgn_idx
  - list_act_rgn
- ufshpb_subregion
- ufshpb_region
- ufshpb_region

* Some of data structures of HPB is not included in the picture

Described in driver/scsi/ufs/ufshcd.h

Described in driver/scsi/ufs/ufshpb.h

Cache Management
State change of HPB

- **READ -> HPB_READ:**
  - **HPB_PRESENT** & **HPB_RGN_ACTIVE/PINNED** & **HPB_SRGN_CLEAN**
  - The cached entry is not **DIRTY**
Region Management

- **Activate/Inactivate Information**
  - Active/Inactivate region is informed by device through the region number
  - Information is received at each end of transaction through Response UPIU
Region Management

- **Activate Region**
  - Active subregion is informed by Device

* RB: READ BUFFER command: used in HPB for requesting mapping entry
** WB: WRITE BUFFER command: used in HPB as prefetching command
Region Management

- **Inactivate Region – Victimized by LRU**
  - Active regions are managed by LRU algorithm (Active LRU list)
    - If the list is full, the last used region has to be selected as **victim**
    - Then, selected victim will be **inactivated**
    - Total number of active region is not changed

Max Active region : 2048

```
0 → 1 → 2 → 2044 → 2045 → 2046 → 2047
```

Active LRU list : 2048

```
0 → 1 → 2 → 2044 → 2045 → 2046
```

Active LRU list : 2047

```
0 → 1 → 2 → 2044 → 2045 → 2046 → 2047
```

Active LRU list : 2048

UFS Device

Region Activation Informed
Region Management

**Inactivate Region – Informed by Device**

- Also, device can inform the region which should be inactivated
  - HPB finds the region which is informed in Active LRU list
  - Informed region is **deleted directly** from list and **inactivated**
  - Total number of active region is reduced

Max Active region : 2048

---

**Diagram:**

- UFS Device

  - Region **Inactivation** Informed

  - Inactive Region
    - 0 → 1 → 2
    - 2044 → 2045 → 2046 → 2047

  - Active LRU list : 2048
    - 0 → 1 → 2
      - 2044 → 2045 → 2046 → 2047

  - Active LRU list : 2047
      - 0 → 1 → 2
        - 2044 → 2045 → 2046

---
Performance Improvement:
Is it work?
Benchmark & UX experience

UFS HPB Benchmark (RR)

UFS HPB UX (App Launch Time)

<table>
<thead>
<tr>
<th>CYCLE</th>
<th>UFS [S]</th>
<th>HPB [S]</th>
<th>DIFF [S]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>272.4</td>
<td>264.9</td>
<td>7.5</td>
</tr>
<tr>
<td>2</td>
<td>250.4</td>
<td>248.2</td>
<td>2.2</td>
</tr>
<tr>
<td>3</td>
<td>226.2</td>
<td>215.6</td>
<td>10.6</td>
</tr>
<tr>
<td>4</td>
<td>230.6</td>
<td>214.8</td>
<td>15.8</td>
</tr>
<tr>
<td>5</td>
<td>232.0</td>
<td>218.1</td>
<td>13.9</td>
</tr>
<tr>
<td>6</td>
<td>231.9</td>
<td>212.6</td>
<td>19.3</td>
</tr>
</tbody>
</table>

* Measured with UFS 3.1 samples manufactured by Samsung
Chunk-Range Result

* Measured with UFS 3.1 samples manufactured by Samsung
** Measured through IOzone test: https://www.iozone.org/
Current Status:
What is going on?
HPB Linux Upstream Status

• Upstream Started in Q2 of 2020
  – v40 of HPB 2.0 support (Daejun Park, Samsung)
  – Host Control Mode for HPB (Avri Altman, WDC)
  – Committed to SCSI tree in July 2021!

• Credits and Contribution: Thanks!
  – Avri Altman (WDC)
  – Bart Van Assche (acm)
  – Bean Huo (Micron)
  – Can Guo (Qualcomm)
  – Greg Kroah-Hartman (Linux Foundation)
  – Stanly Chu (Mediatek)
Timeline

- 18th: Micron posted the HPB v1.0 on Mainline
- 5th: Samsung posted the RFC patch for HPB v1.0
- 8th: Samsung posted the HPB v2.0 on AOSP
- 27th: WDC posted the Host Control Mode for HPB v2.0
- 5th: Samsung posted the HPB v2.0
- 12th: PATCH v40 Committed
- 2nd: HPB merged to AOSP

2020 2021
Conclusion

• HPB is still changing:
  – PATCH v40 is committed at 2021-07-12
  – Lots of suggestions are included since last year

• Contribute!
  – Review it, Test it, Share the bug fixes, Please!