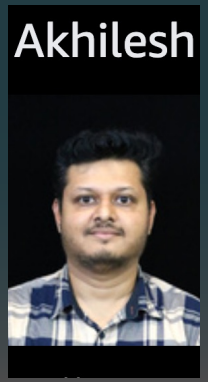
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA
Embedded Linux Conference

Practical Insights into interactive debugging of Linux MMC block device drivers

Akhilesh Patil, Amazon

About me



- ❑ Linux core BSP developer at Amazon Devices and Services.
- ❑ Working on storage drivers for next generation fire TV products
- ❑ Member of Silicon and Systems Group (SSG)
- ❑ Previously worked at Texas Instruments (TI) as Boot ROM developer for Sitara SoC
- ❑ LFX Mentorship Summer 25 graduate (kernel bug fixing)
- ❑ Electrical and Electronics engineer

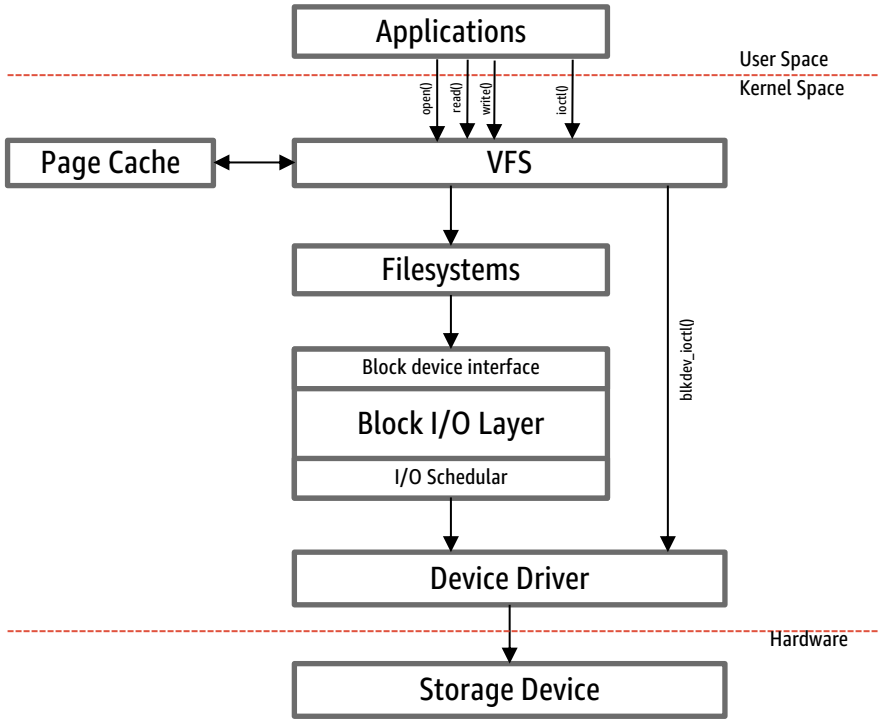
Outline

1. Linux block I/O layer and block devices
2. eMMC bus: Birds eye view
3. Kernel for block device driver debug interactively
4. Tapping I/O requests using golden breakpoints
5. Dive deep into physical signals and bus protocol
6. Leverage mmc_test module

Linux block I/O layer and block devices

Linux storage I/O

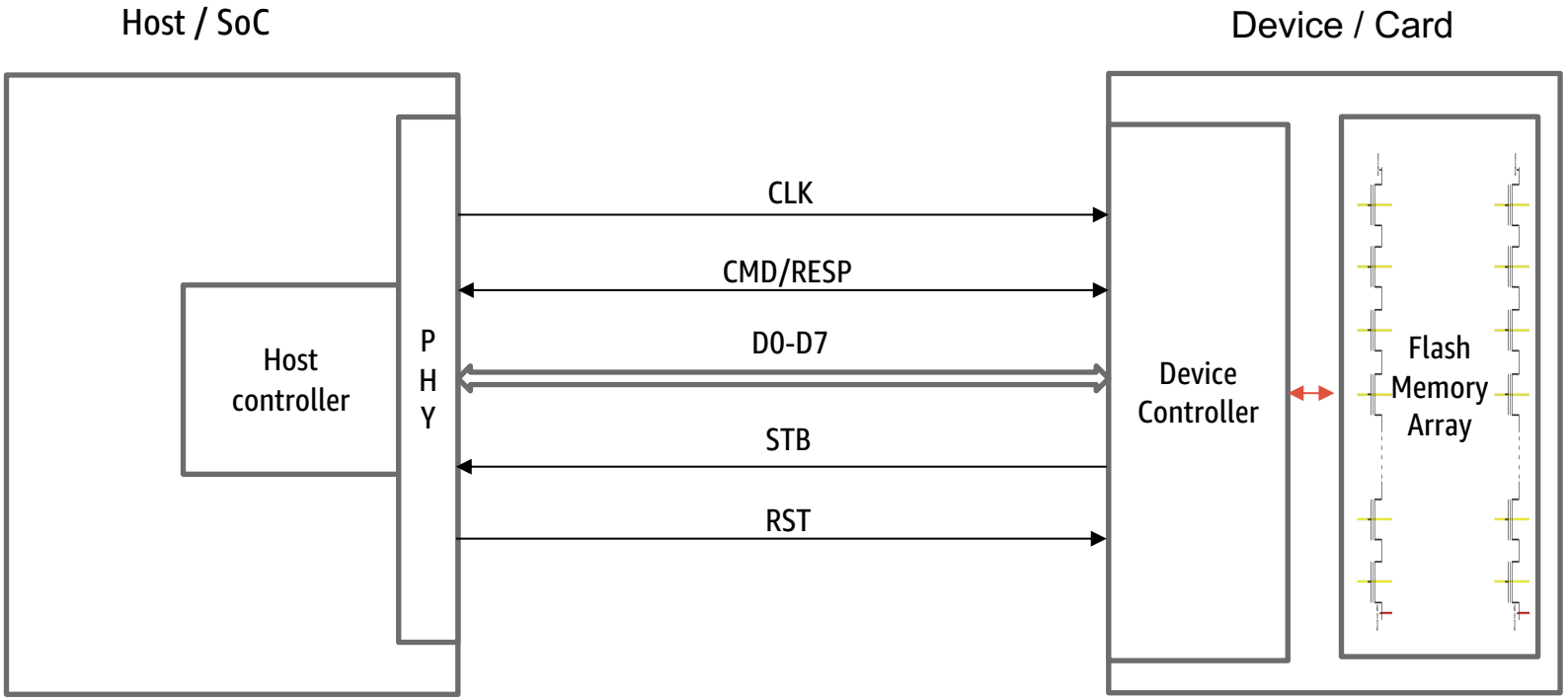
- ✓ Request Submission
- ✓ Queue Management
- ✓ Request Dispatch
- ✓ Completion Notification



- Direct I/O
- Buffered I/O
- Async I/O
- Sync I/O
- ioctl

eMMC bus: Birds eye view

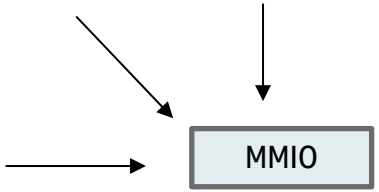
eMMC system: Host and device



Software Hardware Interface: Registers

Host controller		
SDHCI (Host Controller Interface)	Vendor	PHY Control
RSP_R	EMMC_CTRL_R	DLL_CTRL
ARGUMENT_R	CQHCI	TUNE_CTRL
CMD_R	CRYPTO_*	PAD_CTRL
XFER_MODE_R		
CLK_CTRL_R		
TOUT_CTRL_R		
ERROR_INT_STAT_R		
NORMAL_INT_STAT_R		
HOST_CTRL_R		
SW_RST_R		

drivers/mmc/host/sdhci-<vendor>.c



drivers/mmc/host/sdhci.c

Device Controller
Device registers (eMMC 5.1)

RCA → Relative Chip Address

CID → Chip Identification

DSR → Device State Register → CMD / RSP

CSD → Chip Specific Data

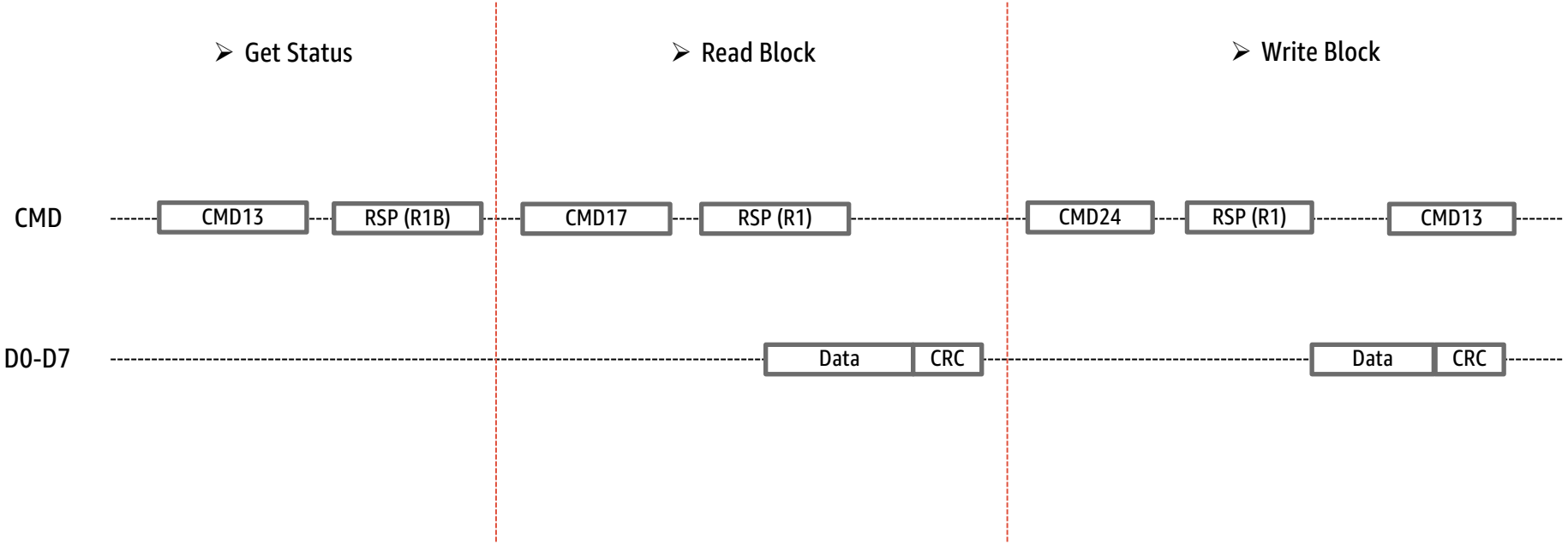
OCR → Operation Conditions Register

EXT_CSD → Extended CSD

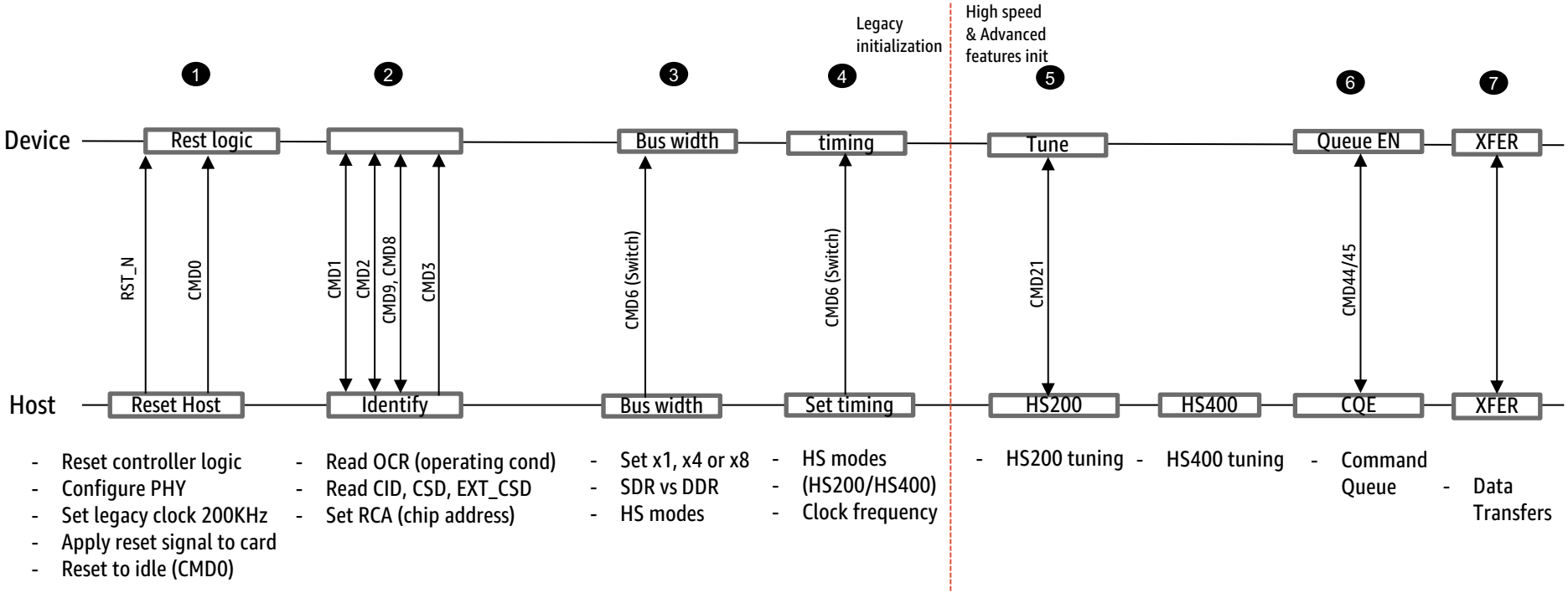
drivers/mmc/core/mmc.c
drivers/mmc/core/mmc_ops.c

✓ Monitoring these registers while debugging give deep peek into the system

Bus Protocol: Command & Response



Bus Initialization (Enumeration)



- Reset controller logic
- Configure PHY
- Set legacy clock 200KHz
- Apply reset signal to card
- Reset to idle (CMD0)

- Read OCR (operating cond)
- Read CID, CSD, EXT_CSD
- Set RCA (chip address)

- Set x1, x4 or x8
- SDR vs DDR
- HS modes

- HS modes (HS200/HS400)
- Clock frequency

- HS200 tuning

- HS400 tuning

- Command Queue

- Data Transfers

drivers/mmc/core/mmc.c : mmc_init_card()

✓ Most critical phase of eMMC → Most of the debugs and platform bring up happen here

Preparing Kernel for interactive block device driver debug

Stopping the kernel for JTAG debug

initramfs
(no fs mounts)

Singe Core
execution

Disable KASLR

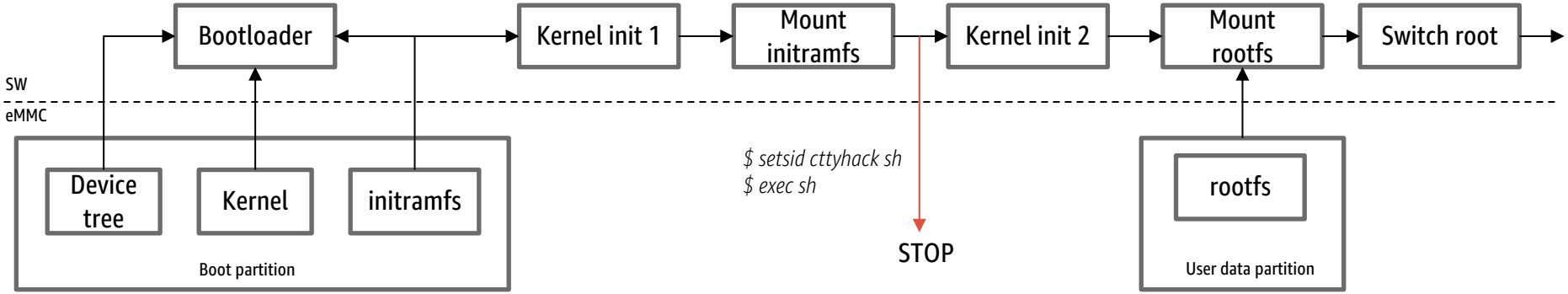
Disable
Softlockups

Disable
Watchdogs

Disable System
Firmware
timeouts

Disable cpuidle

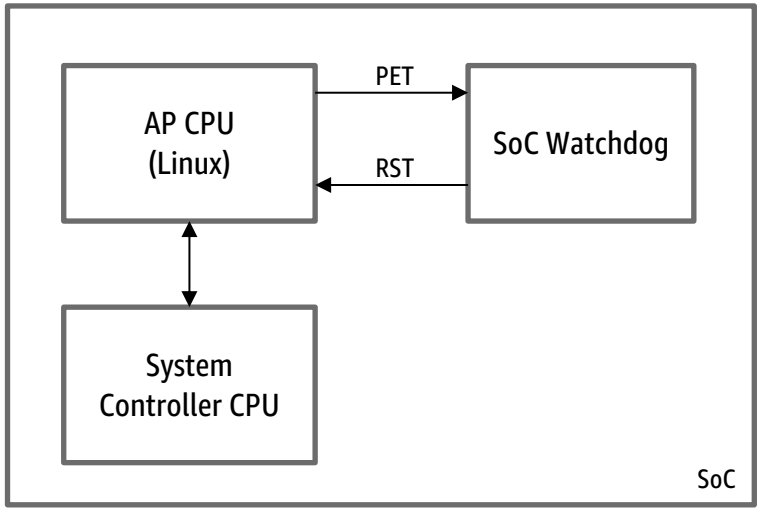
Stopping at initramfs



- ✓ Important utilities : util-linux-blkid, coreutils, mmc-utils
- ✓ No eMMC I/Os at this stage as system is not mounted
- ✓ No Aync or buffered I/Os
- ✓ No active filesystem mounts → zero I/O traffic

Create your own I/O traffic for debug

Disable Watchdog and SYSFW timeouts



- ✓ Unexpected reset when CPU is halted for debug
- ✓ WDT policy and implementation is SoC specific
- ✓ Hard disable WDT → Direct disable via debugger register writes
- ✓ Disable any SoC specific SYSFW handshake based timeouts and panics

Disable SMP, cpuidle, KASLR

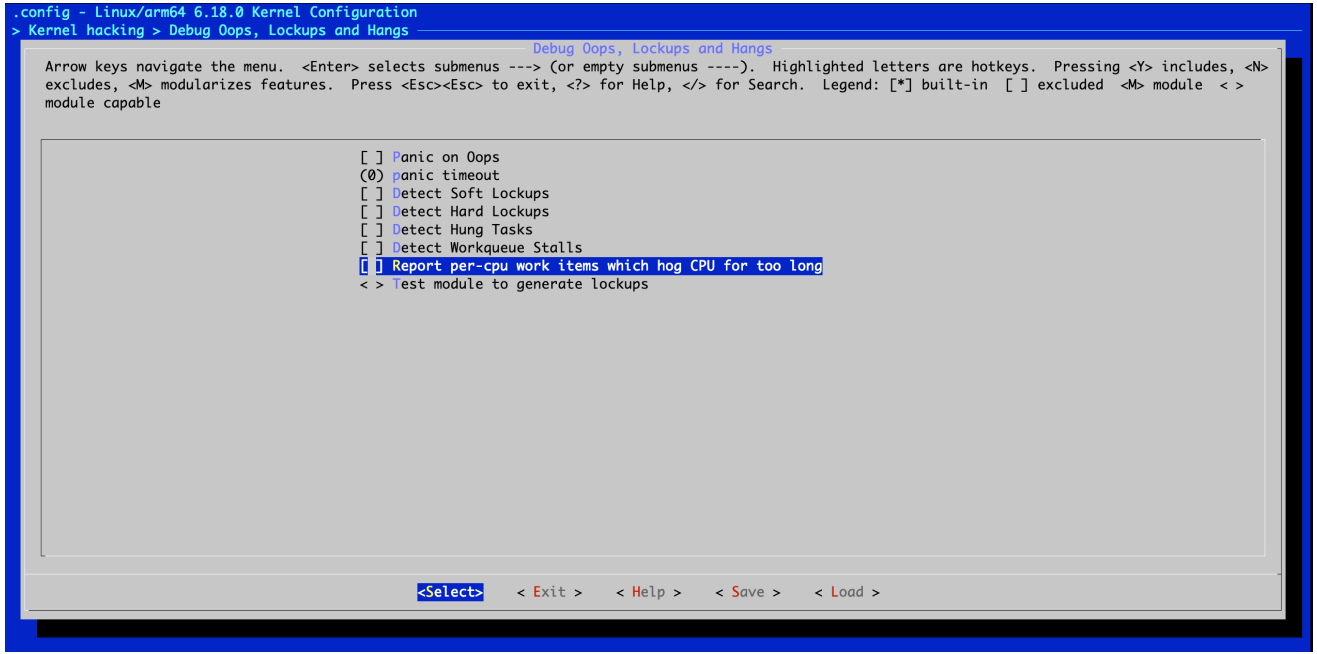
- ✓ Disable SMP to avoid complexities of multicore debug. 1 CPU → Single Queue
- ✓ Disable cpuidle → Debuggers may not work with cpuidle enabled
- ✓ Disable KASLR (Kernel Address Space Layout Randomization) for easy C source code level debugging (vmlinux symbols matching)

```
/ {
    chosen {
        bootargs = "console=ttyAMA0 earlycon \
                    root=/dev/ram0 rw rootfstype=ramfs \
                    nokaslr maxcpus=1 cpuidle.off=1 \
                    loglevel=8";
    };
};
```

Example bootargs in device tree

Disable lockups detectors

Kernel hacking → Debug Oops, Lockups and Hangs

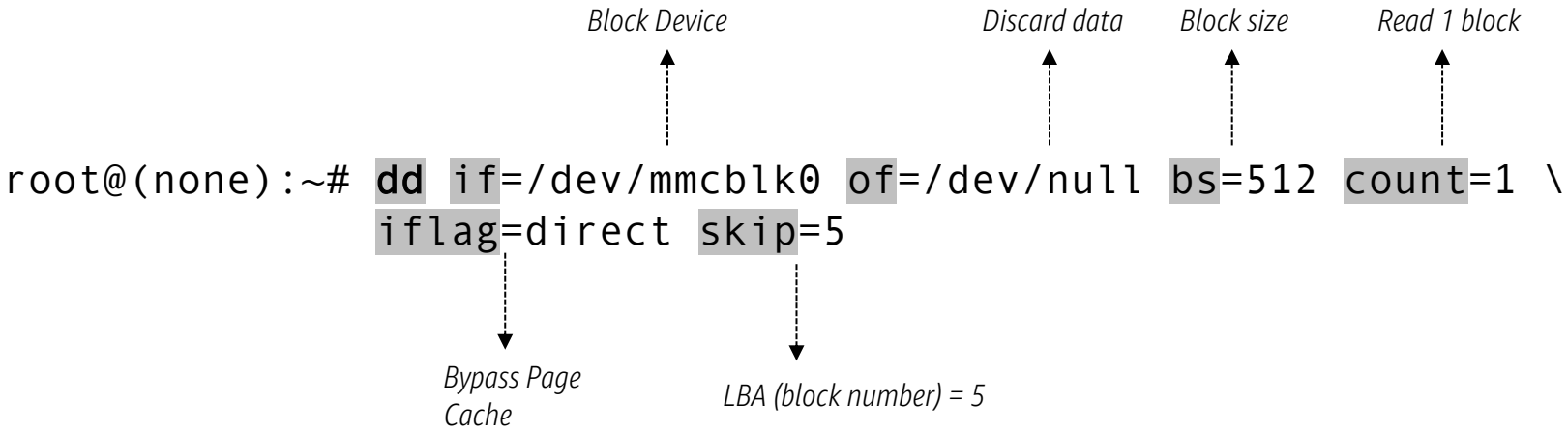


Tapping I/O requests using golden breakpoints

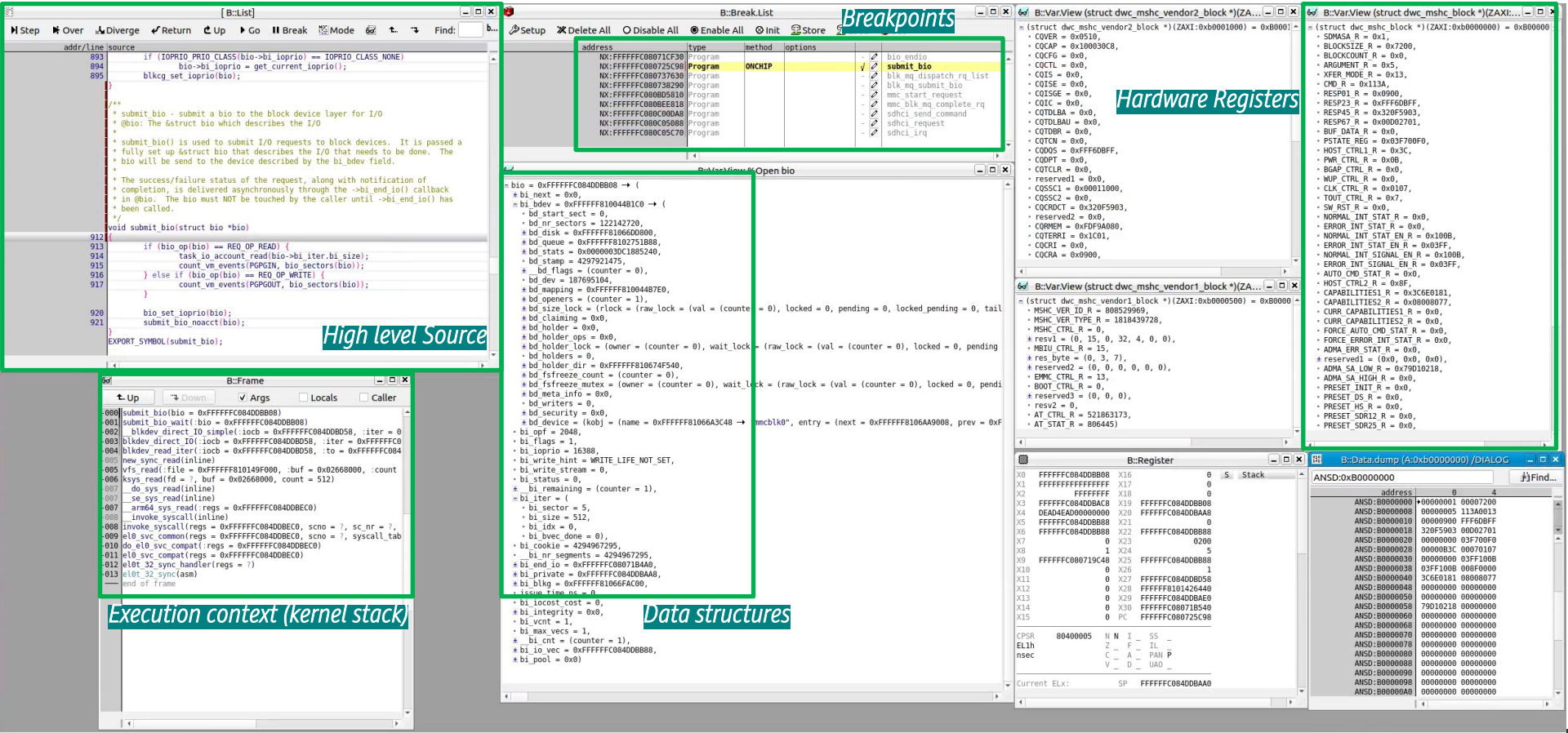
Breakpoints at a glance

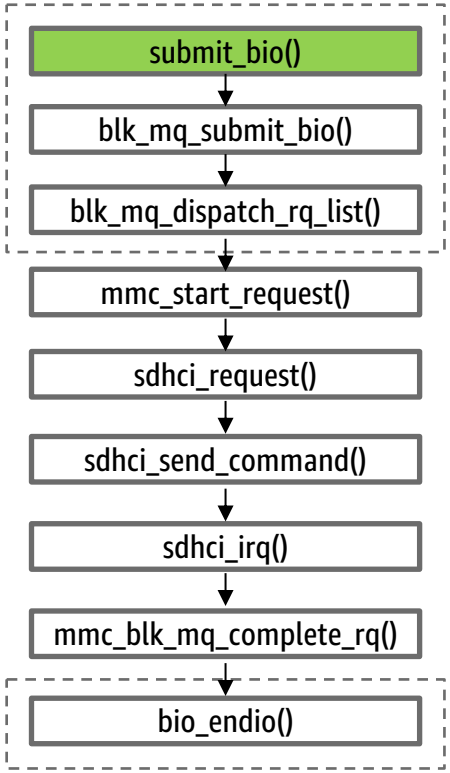
1	submit_bio()	fs/user entry into block layer, submit request
2	blk_mq_submit_bio()	BIO → blk_mq request, insert in hw dispatch queue
3	blk_mq_dispatch_rq_list()	Dispatch to drivers hw queue
4	mmc_start_request()	Entry of request in mmc driver core layer
5	sdhci_request()	I/O request reaches to host controller
6	sdhci_send_command()	Low level MMC command sent
7	sdhci_irq()	mmc driver threaded interrupt handler
8	mmc_blk_mq_complete_rq()	I/O completion notification to block layer
9	bio_endio()	End I/O and notify fs/caller

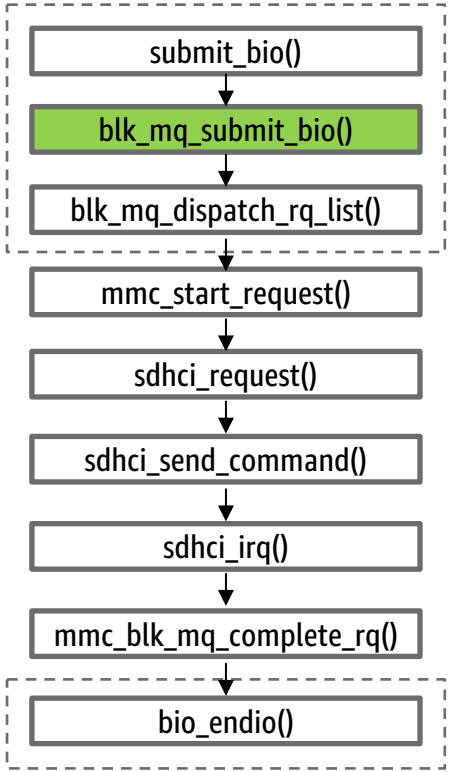
Read Process



Debugger overview (T32 JTAG)

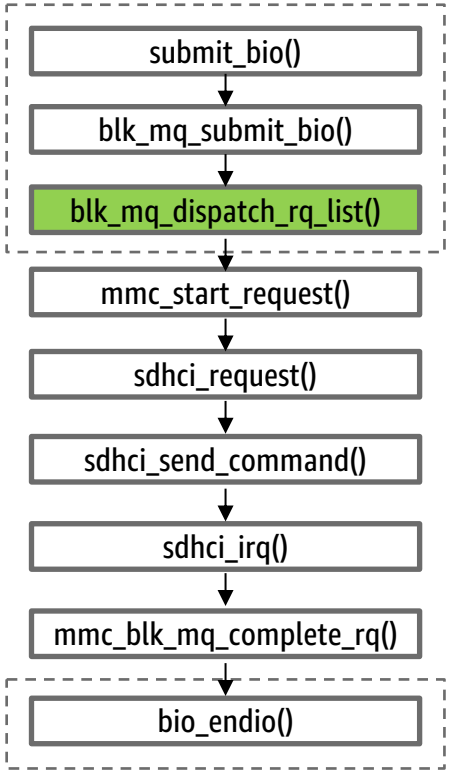

[B::List]
Step
Over
Diverge
Return
Up
Go
Break
Mode
Find:
addr/line source
893 if (IOPRIO_PRIO_CLASS(bio->bi_ioprio) == IOPRIO_CLASS_NONE)
894 bio->bi_ioprio = get_current_ioprio();
895 blkcg_set_ioprio(bio);
}
/**
* submit_bio - submit a bio to the block device layer for I/O
* @bio: The &struct bio which describes the I/O
*
* submit_bio() is used to submit I/O requests to block devices. It is passed a
* fully set up &struct bio that describes the I/O that needs to be done. The
* bio will be send to the device described by the bi_bdev field.
*
* The success/failure status of the request, along with notification of
* completion, is delivered asynchronously through the ->bi_end_io() callback
* in @bio. The bio must NOT be touched by the caller until ->bi_end_io() has
* been called.
*/
void submit_bio(struct bio *bio)
{
912
913 if (bio_op(bio) == REQ_OP_READ) {
914 task_io_account_read(bio->bi_iter.bi_size);
915 count_vm_events(PGPGIN, bio_sectors(bio));
916 } else if (bio_op(bio) == REQ_OP_WRITE) {
917 count_vm_events(PGPGOUT, bio_sectors(bio));
}
920 bio_set_ioprio(bio);
921 submit_bio_noacct(bio);
}
EXPORT_SYMBOL(submit_bio);
High level Source
B::Break.List
Breakpoints
Setup
Delete All
Disable All
Enable All
Init
Store
address type method options
NX:FFFFFFC08071CF30 Program bio_endio
NX:FFFFFFC080725C98 Program ONCHIP submit_bio
NX:FFFFFFC080737630 Program blk_mq_dispatch_rq_list
NX:FFFFFFC080738290 Program blk_mq_submit_bio
NX:FFFFFFC080BD5810 Program mmc_start_request
NX:FFFFFFC080BEE818 Program mmc_blk_mq_complete_rq
NX:FFFFFFC080C00DA8 Program sdhci_send_command
NX:FFFFFFC080C05088 Program sdhci_request
NX:FFFFFFC080C05C70 Program sdhci_irq
B::Var:View (struct dwc_mshc_vendor2_block *)(ZA...
(struct dwc_mshc_vendor2_block *)(ZAXI:0xb0001000) = 0xB000
CQVER = 0x0510,
CQCAP = 0x100030C8,
CQCFG = 0x0,
CQCTL = 0x0,
CQIS = 0x0,
CQISE = 0x0,
CQISGE = 0x0,
CQIC = 0x0,
CQTDLBA = 0x0,
CQTDLBAU = 0x0,
CQTDBR = 0x0,
CQTCN = 0x0,
CQDQS = 0xFFF6DBFF,
CQDPT = 0x0,
CQTCLR = 0x0,
reserved1 = 0x0,
CQSSC1 = 0x00011000,
CQSSC2 = 0x0,
CQCRDCT = 0x320F5903,
reserved2 = 0x0,
CQRMEM = 0xFDF9A080,
CQTERRI = 0x1C01,
CQCRI = 0x0,
CQCRA = 0x0900,
Hardware Registers
B::Var:View (struct dwc_mshc_block *)(ZAXI:...
(struct dwc_mshc_block *)(ZAXI:0xb0000000) = 0xB0000
SDMASA_R = 0x1,
BLOCKSIZE_R = 0x7200,
BLOCKCOUNT_R = 0x0,
ARGUMENT_R = 0x5,
XFER_MODE_R = 0x13,
CMD_R = 0x113A,
RESP01_R = 0x0900,
RESP23_R = 0xFFF6DBFF,
RESP45_R = 0x320F5903,
RESP67_R = 0x00D02701,
BUF_DATA_R = 0x0,
PSTATE_REG = 0x03F700F0,
HOST_CTRL1_R = 0x3C,
PWR_CTRL_R = 0x0B,
BGAP_CTRL_R = 0x0,
WUP_CTRL_R = 0x0,
CLK_CTRL_R = 0x0107,
TOUT_CTRL_R = 0x7,
SW_RST_R = 0x0,
NORMAL_INT_STAT_R = 0x0,
ERROR_INT_STAT_R = 0x0,
NORMAL_INT_STAT_EN_R = 0x100B,
ERROR_INT_STAT_EN_R = 0x03FF,
NORMAL_INT_SIGNAL_EN_R = 0x100B,
ERROR_INT_SIGNAL_EN_R = 0x03FF,
AUTO_CMD_STAT_R = 0x0,
HOST_CTRL2_R = 0x0F,
CAPABILITIES1_R = 0x3C6E0181,
CAPABILITIES2_R = 0x08008077,
CURR_CAPABILITIES1_R = 0x0,
CURR_CAPABILITIES2_R = 0x0,
FORCE_AUTO_CMD_STAT_R = 0x0,
FORCE_ERROR_INT_STAT_R = 0x0,
ADMA_ERR_STAT_R = 0x0,
reserved1 = (0x0, 0x0, 0x0),
ADMA_SA_LOW_R = 0x79D10218,
ADMA_SA_HIGH_R = 0x0,
PRESET_INIT_R = 0x0,
PRESET_DS_R = 0x0,
PRESET_HS_R = 0x0,
PRESET_SDR12_R = 0x0,
PRESET_SDR25_R = 0x0,
B::Var:View (struct dwc_mshc_vendor1_block *)(ZA...
(struct dwc_mshc_vendor1_block *)(ZAXI:0xb0000500) = 0xB0000
MSHC_VER_ID_R = 808529969,
MSHC_VER_TYPE_R = 1818439728,
MSHC_CTRL_R = 0,
resv1 = (0, 15, 0, 32, 4, 0, 0),
MBIU_CTRL_R = 15,
res_byte = (0, 3, 7),
reserved2 = (0, 0, 0, 0, 0, 0),
EMMC_CTRL_R = 13,
BOOT_CTRL_R = 0,
reserved3 = (0, 0, 0),
resv2 = 0,
AT_CTRL_R = 521863173,
AT_STAT_R = 806445)
B::Open bio
bio = 0xFFFFFFC084DDB808 → (
bi_next = 0x0,
bi_bdev = 0xFFFFFF810044B1C0 → (
bd_start_sect = 0,
bd_nr_sectors = 122142720,
bd_disk = 0xFFFFFF81066DD800,
bd_queue = 0xFFFFFF8102751B88,
bd_stats = 0x0000003DC1885240,
bd_stamp = 4297921475,
__bd_flags = (counter = 0),
bd_dev = 187695104,
bd_mapping = 0xFFFFFF810044B7E0,
bd_openers = (counter = 1),
bd_size_lock = (rlock = (raw_lock = (val = (counte = 0), locked = 0, pending = 0, locked_pending = 0, tail
bd_claiming = 0x0,
bd_holder = 0x0,
bd_holder_ops = 0x0,
bd_holder_lock = (owner = (counter = 0), wait_lock = (raw_lock = (val = (counter = 0), locked = 0, pending
bd_holders = 0,
bd_holder_dir = 0xFFFFFF810674F540,
bd_fsfreeze_count = (counter = 0),
bd_fsfreeze_mutex = (owner = (counter = 0), wait_lock = (raw_lock = (val = (counter = 0), locked = 0, pendi
bd_meta_info = 0x0,
bd_writers = 0,
bd_security = 0x0,
bd_device = (kobj = (name = 0xFFFFFF81066A3C48 → "mmcblk0", entry = (next = 0xFFFFFF8106AA9008, prev = 0xF
bi_opf = 2048,
bi_flags = 1,
bi_ioprio = 16388,
bi_write_hint = WRITE_LIFE_NOT_SET,
bi_write_stream = 0,
bi_status = 0,
__bi_remaining = (counter = 1),
bi_iter = (
bi_sector = 5,
bi_size = 512,
bi_idx = 0,
bi_bvec_done = 0),
bi_cookie = 4294967295,
__bi_nr_segments = 4294967295,
bi_end_io = 0xFFFFFFC08071B4A0,
bi_private = 0xFFFFFFC084DDBAA8,
bi_blkg = 0xFFFFFF81066FAC00,
issue_time_ns = 0,
bi_iocost_cost = 0,
bi_integrity = 0x0,
bi_vcnt = 1,
bi_max_vecs = 1,
__bi_cnt = (counter = 1),
bi_io_vec = 0xFFFFFFC084DDBB88,
bi_pool = 0x0)
Data structures
B::Frame
Up
Down
Args
Locals
Caller
000 submit_bio(bio = 0xFFFFFFC084DDBB08)
001 submit_bio_wait(:bio = 0xFFFFFFC084DDBB08)
002 __blkdev_direct_IO_simple(:iocb = 0xFFFFFFC084DDBD58, :iter = 0
003 blkdev_direct_IO(:iocb = 0xFFFFFFC084DDBD58, :iter = 0xFFFFFFC0
004 blkdev_read_iter(:iocb = 0xFFFFFFC084DDBD58, :to = 0xFFFFFFC084
005 new_sync_read(inline)
006 vfs_read(:file = 0xFFFFFF810149F000, :buf = 0x02668000, :count
007 ksys_read(fd = ?, buf = 0x02668000, count = 512)
007 __do_sys_read(inline)
007 __se_sys_read(inline)
007 __arm64_sys_read(:regs = 0xFFFFFFC084DDBEC0)
008 __invoke_syscall(inline)
008 invoke_syscall(regs = 0xFFFFFFC084DDBEC0, scno = ?, sc_nr = ?,
009 el0_svc_common(regs = 0xFFFFFFC084DDBEC0, scno = ?, syscall_tab
010 do_el0_svc_compat(:regs = 0xFFFFFFC084DDBEC0)
011 el0_svc_compat(regs = 0xFFFFFFC084DDBEC0)
012 el0t_32_sync_handler(regs = ?)
013 el0t_32_sync(asm)
end of frame
Execution context (kernel stack)
B::Register
X0 FFFFFFC084DDBB08 X16 0 S Stack
X1 FFFFFFFFFFFFFFFF X17 0
X2 FFFFFFF X18 0
X3 FFFFFFC084DDBAC8 X19 FFFFFFC084DDBB08
X4 DEAD4EAD00000000 X20 FFFFFFC084DDBAA8
X5 FFFFFFC084DDBB88 X21 0
X6 FFFFFFC084DDBB88 X22 FFFFFFC084DDBB88
X7 0 X23 0200
X8 1 X24 5
X9 FFFFFFC080719C48 X25 FFFFFFC084DDBB88
X10 0 X26 1
X11 0 X27 FFFFFFC084DDBD58
X12 0 X28 FFFFFF8101426440
X13 0 X29 FFFFFFC084DDBAE0
X14 0 X30 FFFFFFC08071B540
X15 0 PC FFFFFFC080725C98
CPSR 80400005 N N I _ SS _
EL1h Z _ F _ IL _
nsec C _ A _ PAN P
V _ D _ UAO _
Current ELx: SP FFFFFFC084DDBAA0
B::Data.dump (A:0xb0000000) /DIALOG
ANSD:0xB0000000
Find...
address 0 4
ANSD:B0000000 +00000001 00007200
ANSD:B0000008 00000005 113A0013
ANSD:B0000010 00000900 FFF6DBFF
ANSD:B0000018 320F5903 00D02701
ANSD:B0000020 00000000 03F700F0
ANSD:B0000028 00000B3C 00070107
ANSD:B0000030 00000000 03FF100B
ANSD:B0000038 03FF100B 000F0000
ANSD:B0000040 3C6E0181 08008077
ANSD:B0000048 00000000 00000000
ANSD:B0000050 00000000 00000000
ANSD:B0000058 79D10218 00000000
ANSD:B0000060 00000000 00000000
ANSD:B0000068 00000000 00000000
ANSD:B0000070 00000000 00000000
ANSD:B0000078 00000000 00000000
ANSD:B0000080 00000000 00000000
ANSD:B0000088 00000000 00000000
ANSD:B0000090 00000000 00000000
ANSD:B0000098 00000000 00000000
ANSD:B00000A0 00000000 00000000


submit_bio()
blk_mq_submit_bio()
blk_mq_dispatch_rq_list()
mmc_start_request()
sdhci_request()
sdhci_send_command()
sdhci_irq()
mmc_blk_mq_complete_rq()
bio_endio()




```
void blk_mq_submit_bio(struct bio *bio)
{
        struct request_queue *q = bdev_get_queue(bio->bi_bdev);
        struct blk_plug *plug = current->plug;
        const int is_sync = op_is_sync(bio->bi_opf);
        struct blk_mq_hw_ctx *hctx;
        unsigned int nr_segs;
        struct request *rq;
        blk_status_t ret;
```

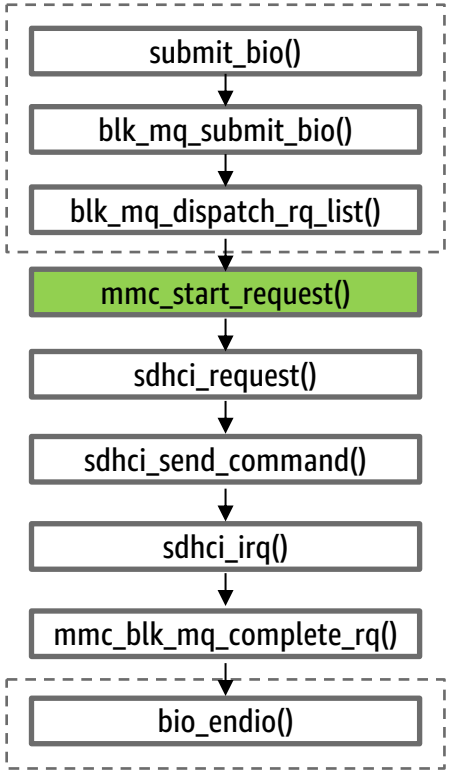
```
- rq = 0xFFFFFF8106A13000 → (
  + q = 0xFFFFFF8102751B88,
  + mq_ctx = 0xFFFFFFFEBF740140,
  + mq_hctx = 0xFFFFFF81066FA600,
  ▪ cmd_flags = 2048,
  ▪ rq_flags = 304,
  ▪ tag = -1,
  ▪ internal_tag = 101,
  ▪ timeout = 0,
  ▪ __data_len = 512,
  ▪ __sector = 5,
  - bio = 0xFFFFFFC084DDBB08 → (
    + bi_next = 0x0,
    + bi_bdev = 0xFFFFFF810044B1C0,
    ▪ bi_opf = 2048,
    ▪ bi_flags = 65,
    ▪ bi_ioprio = 16388,
    ▪ bi_write_hint = WRITE_LIFE_NOT_SET,
    ▪ bi_write_stream = 0,
    ▪ bi_status = 0,
    + __bi_remaining = (counter = 1),
    + bi_iter = (bi_sector = 5, bi_size = 512,
    ▪ bi_cookie = 4294967295,
    ▪ __bi_nr_segments = 4294967295,
    + bi_end_io = 0xFFFFFFC08071B4A0,
    + bi_private = 0xFFFFFFC084DDBAA8,
    + bi_blkg = 0xFFFFFF81066FAC00,
    ▪ issue_time_ns = 0,
    ▪ bi_iocost_cost = 0,
    + bi_integrity = 0x0,
    ▪ bi_vcnt = 1,
    ▪ bi_max_vecs = 1,
    + __bi_cnt = (counter = 1),
    + bi_io_vec = 0xFFFFFFC084DDBB88,
    + bi_pool = 0x0),
```




```
bool blk_mq_dispatch_rq_list(struct blk_mq_hw_ctx *hctx, struct
                             bool get_budget)
{
        enum prep_dispatch prep;
        struct request_queue *q = hctx->queue;
        struct request *rq;
        int queued;
        blk_status_t ret = BLK_STS_OK;
        bool needs_resource = false;

        struct blk_mq_queue_data bd;

        rq = list_first_entry(list, struct request, que

        WARN_ON_ONCE(hctx != rq->mq_hctx);
        prep = blk_mq_prep_dispatch_rq(rq, get_budget);
        if (prep != PREP_DISPATCH_OK)
                break;

        list_del_init(&rq->queuelist);

        bd.rq = rq;
        bd.last = list_empty(list);

        ret = q->mq_ops->queue_rq(hctx, &bd);
```

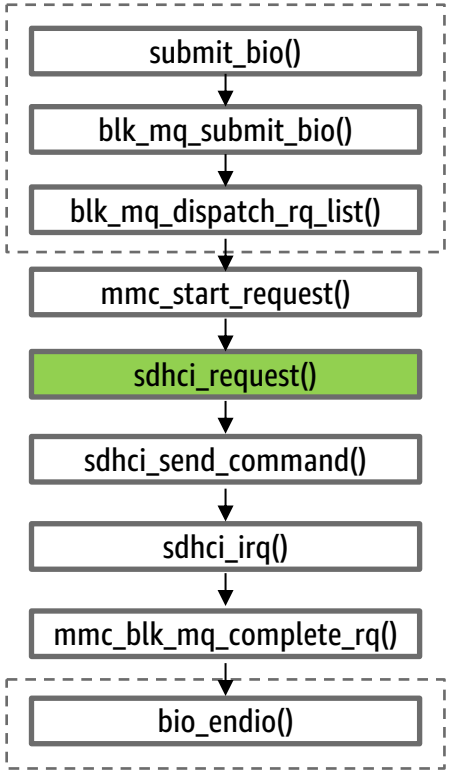
```
B::Var.View %Open bd
bd = (
  + rq = 0xFFFFFF8106A13000,
  - last = FALSE)
```

```
B::Var.View %Open hctx
hctx = 0xFFFFFF81066FA600 → (
  + lock = (rlock = (raw_lock = (val = (counter = 0), locked = 0, pe
  + dispatch = (next = 0xFFFFFF81066FA618, prev = 0xFFFFFF81066FA618
  - state = 0,
  + run_work = (work = (data = (counter = 6291456), entry = (next =
  + cpumask = ((bits = (63))),
  - next_cpu = 0,
  - next_cpu_batch = 8,
  - flags = 16,
  + sched_data = 0x0,
  + queue = 0xFFFFFF8102751B88,
  + fq = 0xFFFFFF8106536800,
  + driver_data = 0x0,
  + ctx_map = (depth = 6, shift = 3, map_nr = 1, round_robin = FALSE
  + dispatch_from = 0x0,
  - dispatch_busy = 0,
```




```
int mmc_start_request(struct mmc_host *host, stru
{
        int err;

        if (mrq->cmd->has_ext_addr)
                mmc_send_ext_addr(host, mrq->cmd-

        mmc_mrq_pr_debug(host, mrq, false);

        WARN_ON(!host->claimed);

        err = mmc_mrq_prep(host, mrq);
        if (err)
                return err;

        if (host->uhs2_sd_tran)
                mmc_uhs2_prepare_cmd(host, mrq);

        led_trigger_event(host->led, LED_FULL);
        __mmc_start_request(host, mrq);
```

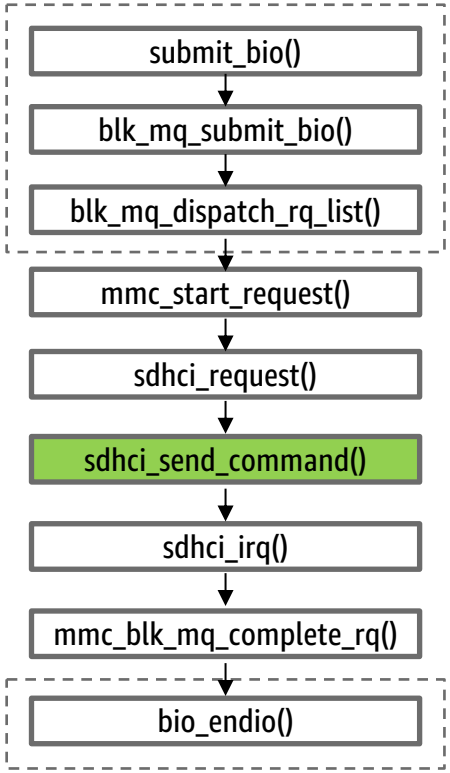
```
- mrq = 0xFFFFFF8106A130F8 → (
  + sbc = 0x0,
  - cmd = 0xFFFFFF8106A13208 → (
      · opcode = 17,
      · arg = 5,
    + resp = (0, 0, 0, 0),
      · flags = 181,
      · retries = 0,
      · error = 0,
      · busy_timeout = 0,
    + data = 0xFFFFFF8106A13298,
    + mrq = 0xFFFFFF8106A130F8,
    + uhs2_cmd = 0x0,
      · has_ext_addr = FALSE,
      · ext_addr = 0),
  - data = 0xFFFFFF8106A13298 → (
      · timeout_ns = 600000000,
      · timeout_clks = 0,
      · blksz = 512,
      · blocks = 1,
      · blk_addr = 5,
      · error = 0,
      · flags = 512,
      · bytes_xfered = 0,
    + stop = 0x0,
    + mrq = 0xFFFFFF8106A130F8,
      · sg_len = 1,
      · sg_count = 1,
    + sg = 0xFFFFFF8106A39000,
      · host_cookie = 1),
```




```
void sdhci_request(struct mmc_host *mmc, struct mmc_request *mrq)
{
        struct sdhci_host *host = mmc_priv(mmc);
        struct mmc_command *cmd;
        unsigned long flags;
        bool present;

        /* Firstly check card presence */
        present = mmc->ops->get_cd(mmc);

        spin_lock_irqsave(&host->lock, flags);

        sdhci_led_activate(host);

        if (sdhci_present_error(host, mrq->cmd, present))
                goto out_finish;

        cmd = sdhci_manual_cmd23(host, mrq) ? mrq->sbc : mrq->cmd;

        if (!sdhci_send_command_retry(host, cmd, flags))
                goto out_finish;
```

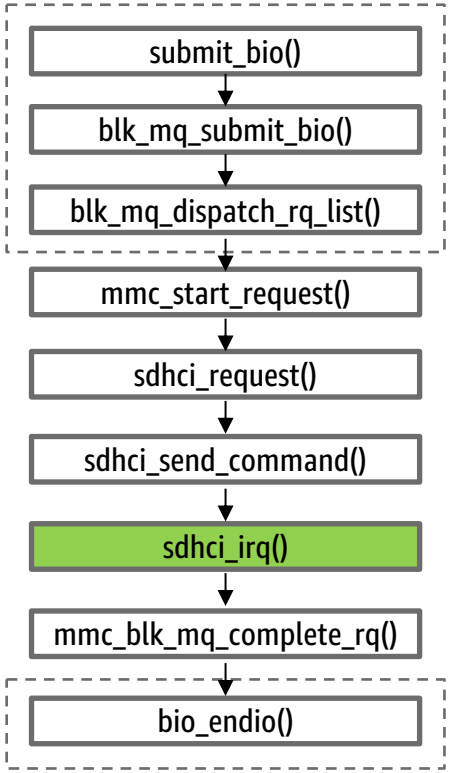
```
- host = 0xFFFFFF81062E9680 → (
  + hw_name = 0xFFFFFF8100E73B10,
  ▪ quirks = 268435456,
  ▪ quirks2 = 16384,
  ▪ irq = 90,
  + ioaddr = 0xFFFFFFC084C94000,
  ▪ mapbase = 0,
  + bounce_buffer = 0x0,
  ▪ bounce_addr = 0,
  ▪ bounce_buffer_size = 0,
  + ops = 0xFFFFFFC0811C5F38,
  + mmc = 0xFFFFFF81062E9000,
  + mmc_host_ops = (post_req = 0xFFFFFFC080BFA7F8
  ▪ dma_mask = 0,
  + led = (name = 0xFFFFFF81062E99A0, brightness
  + led_name = (109, 109, 99, 48, 58, 58, 0, 0, (
  + lock = (rlock = (raw_lock = (val = (counter =
  ▪ flags = 53318,
```

```
- mrq = 0xFFFFFF8106A130F8 → (
  + sbc = 0x0,
  - cmd = 0xFFFFFF8106A13208 → (
    ▪ opcode = 17,
    ▪ arg = 5,
    + resp = (0, 0, 0, 0),
    ▪ flags = 181,
    ▪ retries = 0,
    ▪ error = 0,
    ▪ busy_timeout = 0,
    - data = 0xFFFFFF8106A13298 → (
      ▪ timeout_ns = 600000000,
      ▪ timeout_clks = 0,
      ▪ blksz = 512,
      ▪ blocks = 1,
      ▪ blk_addr = 5,
      ▪ error = 0,
      ▪ flags = 512,
      ▪ bytes_xfered = 0,
      + stop = 0x0,
      + mrq = 0xFFFFFF8106A130F8,
      ▪ sg_len = 1,
      ▪ sg_count = 1,
      + sg = 0xFFFFFF8106A39000,
      ▪ host_cookie = 1),
```




```
static bool sdhci_send_command(struct sdhci_host *host, struct mmc_c
{
        int flags;
        u32 mask;
        unsigned long timeout;

        if (cmd->data) {
                if (host->use_external_dma)
                        sdhci_external_dma_prepare_data(host, cmd);
                else
                        sdhci_prepare_data(host, cmd);
        }

        sdhci_writel(host, cmd->arg, SDHCI_ARGUMENT);

        sdhci_set_transfer_mode(host, cmd);

        timeout = jiffies;
        if (host->data_timeout)
                timeout += nsecs_to_jiffies(host->data_timeout);
        else if (!cmd->data && cmd->busy_timeout > 9000)
                timeout += DIV_ROUND_UP(cmd->busy_timeout, 1000) * HZ + HZ;
        else
                timeout += 10 * HZ;
        sdhci_mod_timer(host, cmd->mrq, timeout);

        if (host->use_external_dma)
                sdhci_external_dma_pre_transfer(host, cmd);

        sdhci_writew(host, SDHCI_MAKE_CMD(cmd->opcode, flags), SDHCI_COMMAND);
```

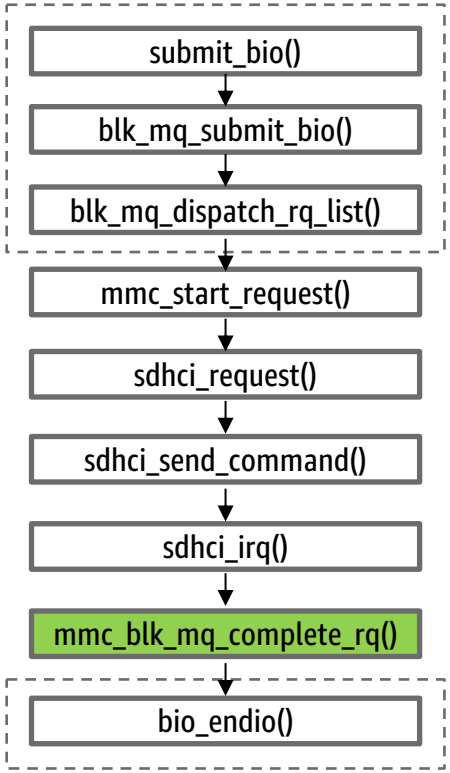
```
(struct dwc_mshc_block *)(ZAXI:0xb00(
  SDMASA_R = 0x1,
  BLOCKSIZE_R = 0x7200,
  BLOCKCOUNT_R = 0x1,
  ARGUMENT_R = 0x5,
  XFER_MODE_R = 0x13,
  CMD_R = 0x113A,
  RESP01_R = 0x0900,
  RESP23_R = 0xFFF6DBFF,
  RESP45_R = 0x320F5903,
  RESP67_R = 0x00D02701,
  BUF_DATA_R = 0x0,
  PSTATE_REG = 0x03F700F0,
  HOST_CTRL1_R = 0x3D,
  PWR_CTRL_R = 0x0B,
  BGAP_CTRL_R = 0x0,
  WUP_CTRL_R = 0x0,
  CLK_CTRL_R = 0x0107,
  TOUT_CTRL_R = 0x7,
  SW_RST_R = 0x0,
  NORMAL_INT_STAT_R = 0x0,
  ERROR_INT_STAT_R = 0x0,
  NORMAL_INT_STAT_EN_R = 0x100B,
  ERROR_INT_STAT_EN_R = 0x03FF,
  NORMAL_INT_SIGNAL_EN_R = 0x100B,
  ERROR_INT_SIGNAL_EN_R = 0x03FF,
  AUTO_CMD_STAT_R = 0x0,
  HOST_CTRL2_R = 0x8F,
  CAPABILITIES1_R = 0x3C6E0181,
  CAPABILITIES2_R = 0x08008077,
  CURR_CAPABILITIES1_R = 0x0,
  CURR_CAPABILITIES2_R = 0x0,
  FORCE_AUTO_CMD_STAT_R = 0x0,
  FORCE_ERROR_INT_STAT_R = 0x0,
  ADMA_ERR_STAT_R = 0x0,
+ reserved1 = (0x0, 0x0, 0x0),
  ADMA_SA_LOW_R = 0x79D10200,
  ADMA_SA_HIGH_R = 0x0,
  PRESET_INIT_R = 0x0,
  PRESET_DS_R = 0x0,
```




```
000 sdhci_irq(irq = 90, dev_id = 0xFFFF
002 handle_irq_event_percpu(inline)
002 handle_irq_event(:desc = 0xFFFFFF81
003 handle_fasteoi_irq(desc = 0xFFFFFF8
004 generic_handle_irq_desc(inline)
004 handle_irq_desc(desc = ?)
```

```
-011 handle_softirqs(:ksirqd = FALSE)
-012 __do_softirq()
-013 ___do_softirq(regs = ?)
-014 call_on_irq_stack()
-015 do_softirq_own_stack()
-016 invoke_softirq(inline)
-016 __irq_exit_rcu()
-017 irq_exit_rcu()
-018 __el1_irq(inline)
-018 el1_interrupt(regs = 0xFFFFFFC084DDB350
-019 el1h_64_irq_handler(regs = ?)
-020 el1h_64_irq(asm)
 ⟶ exception
-021 __daif_local_irq_restore(inline)
-021 arch_local_irq_restore(inline)
-021 __raw_spin_unlock_irqrestore(inline)
-021 _raw_spin_unlock_irqrestore(:lock = 0xF
-022 spin_unlock_irqrestore(inline)
-022 sdhci_request(:mmc = 0xFFFFFF81062E900
-023 __mmc_start_request(:host = 0xFFFFFF810
-024 mmc_start_request(:host = 0xFFFFFF81062
-025 mmc_blk_mq_issue_rw_rq(inline)
```

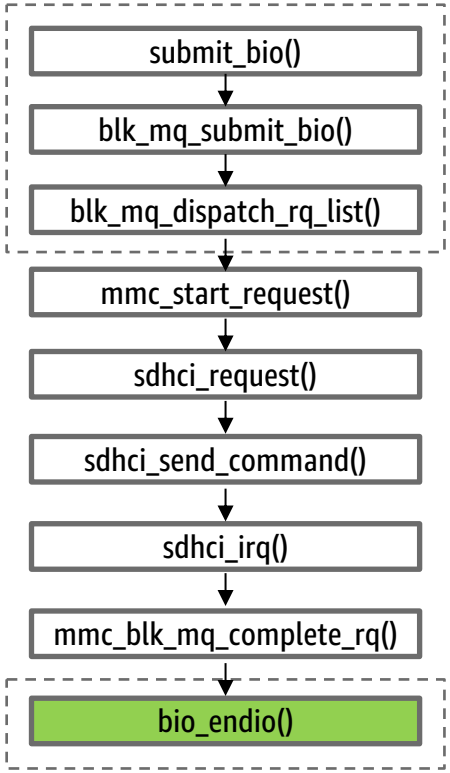
```
(struct dwc_mshc_block *)(ZAXI:0xb00(
  SDMASA_R = 0x1,
  BLOCKSIZE_R = 0x7200,
  BLOCKCOUNT_R = 0x0,
  ARGUMENT_R = 0x5,
  XFER_MODE_R = 0x13,
  CMD_R = 0x113A,
  RESP01_R = 0x0900,
  RESP23_R = 0xFFF6DBFF,
  RESP45_R = 0x320F5903,
  RESP67_R = 0x00D02701,
  BUF_DATA_R = 0x0,
  PSTATE_REG = 0x03F700F0,
  HOST_CTRL1_R = 0x3D,
  PWR_CTRL_R = 0x0B,
  BGAP_CTRL_R = 0x0,
  WUP_CTRL_R = 0x0,
  CLK_CTRL_R = 0x0107,
  TOUT_CTRL_R = 0x7,
  SW_RST_R = 0x0,
  NORMAL_INT_STAT_R = 0x3,
  ERROR_INT_STAT_R = 0x0,
  NORMAL_INT_STAT_EN_R = 0x100B,
  ERROR_INT_STAT_EN_R = 0x03FF,
  NORMAL_INT_SIGNAL_EN_R = 0x100B,
  ERROR_INT_SIGNAL_EN_R = 0x03FF,
  AUTO_CMD_STAT_R = 0x0,
  HOST_CTRL2_R = 0x8F,
  CAPABILITIES1_R = 0x3C6E0181,
  CAPABILITIES2_R = 0x08008077,
  CURR_CAPABILITIES1_R = 0x0,
  CURR_CAPABILITIES2_R = 0x0,
  FORCE_AUTO_CMD_STAT_R = 0x0,
  FORCE_ERROR_INT_STAT_R = 0x0,
  ADMA_ERR_STAT_R = 0x0,
+ reserved1 = (0x0, 0x0, 0x0),
  ADMA_SA_LOW_R = 0x79D10218,
  ADMA_SA_HIGH_R = 0x0,
  PRESET_INIT_R = 0x0,
  PRESET_DS_R = 0x0,
```




```
2078 static void mmc_blk_mq_complete_rq(struct mmc_queue *mq, struct
2079 {
2080         struct mmc_queue_req *mqrq = req_to_mmc_queue_req(req);
2081         unsigned int nr_bytes = mqrq->brq.data.bytes_xfered;
```

```
-000 mmc_blk_mq_complete_rq(mq = 0xF
-001 mmc_blk_mq_complete(:req = 0xFF
-002 blk_mq_complete_request_direct(
-002 mmc_blk_mq_post_req(:mq = 0xFFF
-003 mmc_blk_mq_complete_prev_req(in
-003 mmc_blk_mq_complete_prev_req(in
-003 mmc_blk_mq_complete_work(:work
-004 process_one_work(:worker = 0xFF
-005 process_scheduled_works(inline)
-005 worker_thread(:__worker = 0xFFF
-006 kthread(_create = ?)
-007 ret_from_fork(asm)
 —— end of frame
```

```
- req = 0xFFFFFF8106A13000 → (
  + q = 0xFFFFFF8102751B88,
  + mq_ctx = 0xFFFFFFFEBF740140,
  + mq_hctx = 0xFFFFFF81066FA600,
  • cmd_flags = 2048,
  • rq_flags = 313,
  • tag = 12,
  • internal_tag = 101,
  • timeout = 60000,
  • __data_len = 512,
  • __sector = 5,
  - bio = 0xFFFFFFC084DDBB08 → (
```




```
void bio_endio(struct bio *bio)
{
again:
        if (!bio_remaining_done(bio))
                return;
        if (!bio_integrity_endio(bio))
                return;

        blk_zone_bio_endio(bio);

        rq_qos_done_bio(bio);
```

```
- bio = 0xFFFFFFC084DDBB08 → (
  + bi_next = 0x0,
  + bi_bdev = 0xFFFFFF810044B1C0,
  - bi_opf = 2048,
  - bi_flags = 1,
  - bi_ioprio = 16388,
  - bi_write_hint = WRITE_LIFE_NOT_S
  - bi_write_stream = 0,
  - bi_status = 0,
  + __bi_remaining = (counter = 1),
  - bi_iter = (
    - bi_sector = 5,
      bi_size = 0,
    - bi_idx = 0,
    - bi_bvec_done = 0),
```

```
-000 bio_endio bio = 0xFFFFFFC084DDB
-001 blk_update_request(:req = 0xFFF
-002 mmc_blk_mq_complete_rq(mq = ?,
-003 mmc_blk_mq_complete(:req = 0xFF
-004 blk_mq_complete_request_direct(
-004 mmc_blk_mq_post_req(:mq = 0xFFF
-005 mmc_blk_mq_complete_prev_req(in
-005 mmc_blk_mq_complete_prev_req(in
-005 mmc_blk_mq_complete_work(:work
-006 process_one_work(:worker = 0xFF
-007 process_scheduled_works(inline)
-007 worker_thread(:__worker = 0xFFF
-008 kthread(_create = ?)
```

Dive deep into physical signals and bus protocol
Mapping the code and signals

When to probe signals ?

Random I/O errors

Filesystem mount failures

Tuning Failures

RF interference issues

Random I/O timeouts

Specification Compliance

WRITE errors

Signal Integrity

Suspend Resume failures

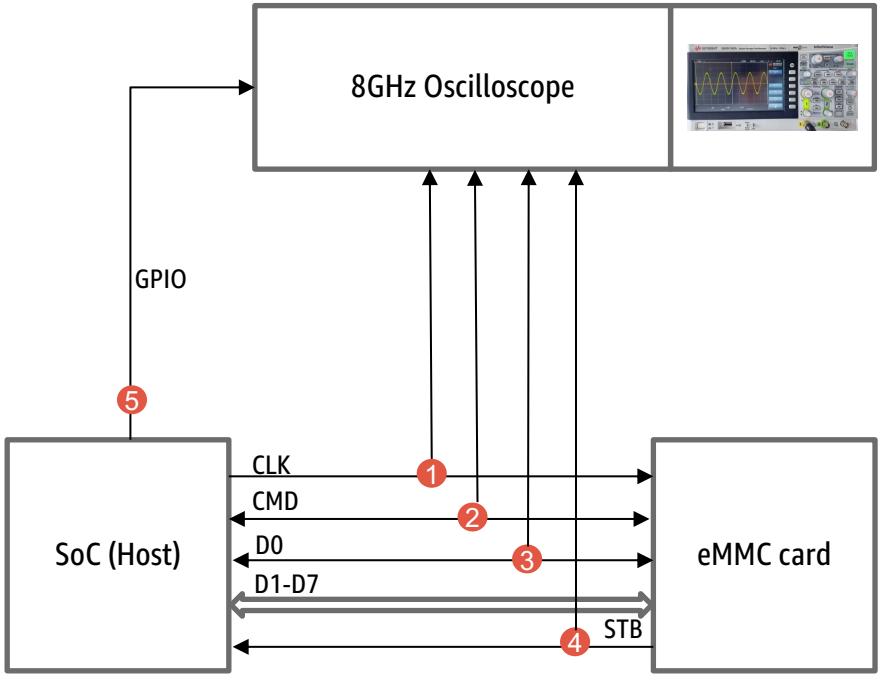
READ errors

Low performance

Random Boot Failures

Hardware Setup

- ✓ Oscilloscope selection
- ✓ Trigger and capture Setup
- ✓ Signal Selection



Mapping code to signal capture

```c
static void mmc_gpio_trigger(struct device *dev)
{
    struct gpio_desc *toggle_gpio;

    /* Get optional GPIO descriptor, default HIGH */
    toggle_gpio = devm_gpiod_get_optional(dev, "toggle",
                                          GPIOD_OUT_HIGH);

    gpiod_set_value(toggle_gpio, 0);  /* Pull Low  */
    gpiod_set_value(toggle_gpio, 1);  /* Pull High */
    udelay(100);                      /* 100µs pulse width */
    gpiod_set_value(toggle_gpio, 0);  /* Pull Low  */
}
```

Device Tree

```
&mmc_controller {
    toggle-gpios = <&gpio0 12 GPIO_ACTIVE_HIGH>;
};
```

Capture start

```c
static bool sdhci_send_command(struct sdhci_host *host,
                               struct mmc_command *cmd)
{
    int flags;

    /* ... */

    if (cmd->flags & MMC_RSP_CRC)
        flags |= SDHCI_CMD_CRC;
    if (cmd->flags & MMC_RSP_OPCODE)
        flags |= SDHCI_CMD_INDEX;
    if (cmd->data || mmc_op_tuning(cmd->opcode))
        flags |= SDHCI_CMD_DATA;

    /* ... */

    if (host->use_external_dma)
        sdhci_external_dma_pre_transfer(host, cmd);

    /* ★ GPIO pulse — oscilloscope trigger before CMD write ★ */
5   mmc_gpio_trigger(host->mmc->parent);

    sdhci_writew(host,  ⟸ MMC transaction start
                 SDHCI_MAKE_CMD(cmd->opcode, flags),
                 SDHCI_COMMAND);
    return true;
}
```

drivers/mmc/host/sdhci.c

Electrical signals

```
root@(none):~# dd if=/dev/mmcblk0 of=/dev/null
bs=512 count=1 iflag=direct skip=5
```

4646	5.161218839	R1b (index = 38)		DEVICE_STATUS	0x00001201	0x53	1	DDR8	211MHz	3.3V	Ncr = 10
4647	47.911617287	CMD6	AC	SWITCH	0x03B30801	0x79	1	DDR8	211MHz	3.3V	Nrc = 8
4648	47.91161756s	Busy Asserted						DDR8	211MHz	3.3V	
4649	47.911617575	R1 (index = 12)		DEVICE_STATUS	0x00001201	0x5F	1	DDR8	200MHz	3.3V	Ncr = 3
4650	47.91184889S	Busy Deasserted						DDR8	200MHz	3.3V	
4651	47.911873783	CMD13	AC	SEND_STATUS	0x00010000	0x29	1	DDR8	200MHz	3.3V	Nrc = 4977
4652	47.911874071	R1 (index = 26)		DEVICE_STATUS	0x00001200	0x3F	1	DDR8	211MHz	3.3V	Ncr = 10
4653	47.911900615	CMD17	ADTC	READ_SINGLE_BLOCK	0x00000005	0x7	1	DDR8	200MHz	3.3V	Nrc = 5263
4654	47.911900919	RESERVED (index = 34)		RESERVED	0x00001200	0x67	1	DDR8	200MHz	3.3V	Ncr = 12
4655	80.57052642:	CMD17	ADTC	READ_SINGLE_BLOCK	0x00000005	0x7	1	DDR8	200MHz	3.3V	Nrc = 8
4656	80.57052670:	RESERVED (index = 34)		RESERVED	0x00001200	0x67	1	DDR8	211MHz	3.3V	Ncr = 8

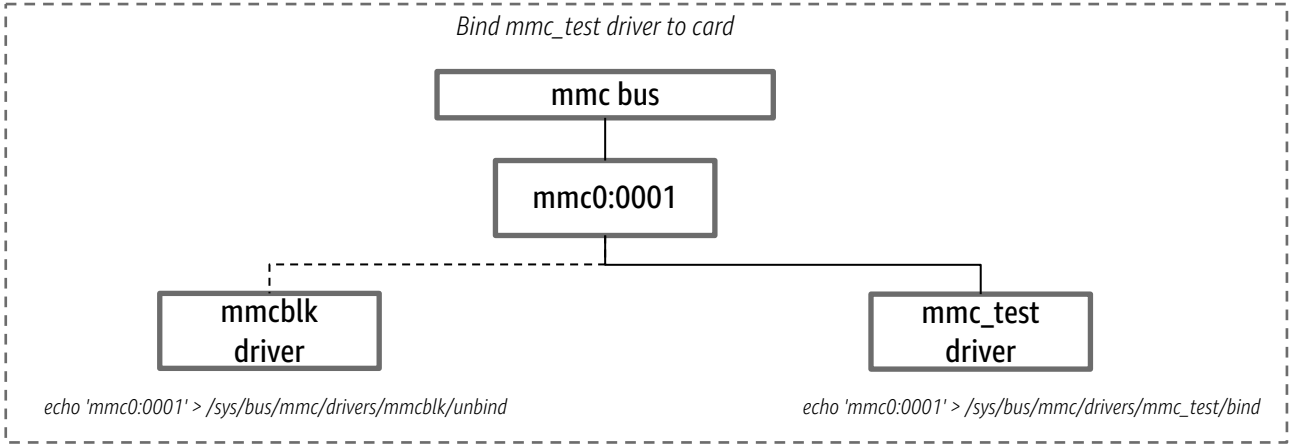
Bus protocol decode

Leverage mmc_test module

Menuconfig:
Device Drivers → MMC/SD/SDIO card support → MMC host test driver

Unmount any active filesystem mounts on mmc

Bind mmc_test driver to card



Run targeted tests:
echo 4 > /sys/kernel/debug/mmc0/mmc0:0001/test

Thank You