

# Rockchip DSMC 开发文档

---

文件标识: RK-KF-YF-C05

发布版本: V1.0.0

日期: 2024-06-14

文件密级: ☐绝密 ☐秘密 ☐内部资料 ☒公开

## 免责声明

本文档按“现状”提供, 瑞芯微电子股份有限公司(“本公司”, 下同)不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因, 本文档将可能在未经任何通知的情况下, 不定期进行更新或修改。

## 商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标, 归本公司所有。

本文档可能提及的其他所有注册商标或商标, 由其各自拥有者所有。

版权所有 © 2024 瑞芯微电子股份有限公司

超越合理使用范畴, 非经本公司书面许可, 任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部, 并不得以任何形式传播。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址: 福建省福州市铜盘路软件园A区18号

网址: [www.rock-chips.com](http://www.rock-chips.com)

客户服务电话: +86-4007-700-590

客户服务传真: +86-591-83951833

客户服务邮箱: [fae@rock-chips.com](mailto:fae@rock-chips.com)

前言

概述

本文为ROCKHIP DSMC模块的kernel开发提供说明和使用方法。

产品版本

| 芯片名称   | 内核版本        |
|--------|-------------|
| RK3576 | kernel 6.10 |

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

| 版本号    | 作者  | 修改日期       | 修改说明 |
|--------|-----|------------|------|
| V1.0.0 | 何智欢 | 2024-06-14 | 初始版本 |

## 目录

### Rockchip DSMC 开发文档

1. 名称解释
2. 概述
3. DSMC驱动
  - 3.1 驱动文件
  - 3.2 DTS节点配置
  - 3.3 内核配置
4. 内核态对DSMC从设备内存的访问
  - 4.1 调用驱动接口
  - 4.2 直接访问
5. 用户态对DSMC的访问
  - 5.1 通过特定节点访问
  - 5.2 直接访问
6. DSMC slave内存空间分配
  - 6.1 PSRAM
  - 6.2 Local bus
7. DSMC Local bus host与slave的数据交互
  - 7.1 FIFO
  - 7.2 Register

## 1. 名称解释

- DSMC: Double Data Rate Serial Memory Controller, 双倍速率串行存储器控制器
- PSRAM: Pseudo static random access memory, 伪静态随机存储器
- DPRAM: Dual Port Random Access Memory, 双向随机存取存储器

## 2. 概述

Double Data Rate Serial Memory Controller (DSMC), 双倍速率串行存储器控制器, 通过命令、地址、数据线分时复用, 数据上下沿传输, 具有少引脚数、高带宽的特点。数据线位宽支持x8、x16, 最多支持4个chip select。传输协议支持Hyperbus PSRAM、Xccela PSRAM和 Local bus。若使用Local bus协议, 从设备需使用RK开发的slave模型, 或者传输协议相同。若使用HYPERBUS PSRAM、XCCELA PSRAM协议, 从设备支持winbond、AP memory、Cypress、ISSI等厂家生产的PSRAM颗粒。

## 3. DSMC驱动

### 3.1 驱动文件

DSMC驱动文件位置:

```
drivers/memory/rockchip/dsmc-host.c          /* 主要驱动程序 */
drivers/memory/rockchip/dsmc-controller.c     /* DSMC控制器行为配置 */
drivers/memory/rockchip/dsmc-lb-device.c      /* DSMC Local bus设备 */
```

### 3.2 DTS节点配置

```
dsmc: dsmc@2a280000 {
    ...
    clock-frequency = <100000000>;          /* DSMC接口频率设置 */
    ...
    /* 从设备属性 */
    slave {
        rockchip,dqs-dll = <0x20 0x20      /* 从设备cs0的DQS0、DQS1 DLL延迟参数
    */
                                0x20 0x20      /* 从设备cs1的DQS0、DQS1 DLL延迟参数
    */
                                0x20 0x20      /* 从设备cs2的DQS0、DQS1 DLL延迟参数
    */
                                0x20 0x20>;      /* 从设备cs3的DQS0、DQS1 DLL延迟参数
    */
    }
```

```

/*
 * rockchip, ranges: DSMC访问从设备内存的基地址，大小；
 * 若不同CS的内存空间大小不同，那么需要配置最大的。
 * rockchip, ranges = <0x0 0x10000000 0x0 0x2000000> 含义：若外设是
PSRAM,
 * 那么每个CS都分配0x2000000大小的内存空间；
 * 若外设是Local Bus，那么每个region都分配0x2000000大小的内存空间。
 */
rockchip, ranges = <0x0 0x10000000 0x0 0x2000000>;
rockchip, slave-dev = <&dsmc_slave>;
};

};

dsmc_slave: dsmc_slave {
    compatible = "rockchip, dsmc-slave";
    rockchip, clk-mode = <0>; /* clk 模式，仅限Local bus */
    status = "disabled";
    /* 从设备是PSRAM (Hyperbus Psram或Xccela Psram) 时，开启对应从设备cs的节点 */
    psram {
        psram0 {
            status = "disabled"; /* 若从设备cs0为PSRAM，则改为"okay" */
        };
        psram1 {
            status = "disabled"; /* 若从设备cs1为PSRAM，则改为"okay" */
        };
        psram2 {
            status = "disabled"; /* 若从设备cs2为PSRAM，则改为"okay" */
        };
        psram3 {
            status = "disabled"; /* 若从设备cs3为PSRAM，则改为"okay" */
        };
    };

    /* 从设备是Local bus设备时，开启、配置对应节点 */
    lb-slave {
        dsmc_lb_slave0: lb-slave0 {
            status = "disabled"; /* 若从设备cs0为Local bus设备，则改为"okay" */
        };

        dsmc_p0_region: region {
            dsmc_p0_region0: region0 { /* 此从设备region0的属性 */
                rockchip, attribute = "Merged FIFO"; /* region0 为从设备可
merge FIFO */
                rockchip, ca-addr-width = <0>; /* CA传输格式，0: 为32bit,
1: 为16bit */
                rockchip, dummy-clk-num = <1>;
                rockchip, cs0-be-ctrlled = <0>; /* 从设备cs0被从设备cs1、2、
3控制 */
                rockchip, cs0-ctrl = <0>; /* 从设备cs0控制从设备cs1、2、3 */
                status = "disabled";
            };
            dsmc_p0_region1: region1 { /* 此从设备region1的属性 */
                rockchip, attribute = "No-Merge FIFO"; /* region1 为从设备可
merge FIFO */
                rockchip, ca-addr-width = <0>;
            };
        };
    };
};

```

```

        rockchip,dummy-clk-num = <1>;
        rockchip,cs0-be-ctrl = <0>;
        rockchip,cs0-ctrl = <0>;
        status = "disabled";
    };
dsmc_p0_region2: region2 { /* 此从设备region2的属性 */
    rockchip,attribute = "DPRA"; /* region2 为从设备DPRAM
*/

    rockchip,ca-addr-width = <0>;
    rockchip,dummy-clk-num = <1>;
    rockchip,cs0-be-ctrl = <0>;
    rockchip,cs0-ctrl = <0>;
    status = "disabled";
};
dsmc_p0_region3: region3 { /* 此从设备region3的属性 */
    rockchip,attribute = "Register"; /* region3 为从设备寄存
器 */

    rockchip,ca-addr-width = <0>;
    rockchip,dummy-clk-num = <1>;
    rockchip,cs0-be-ctrl = <0>;
    rockchip,cs0-ctrl = <0>;
    status = "disabled";
};
};
dsmc_lb_slave1: lb-slave1 {
    status = "disabled"; /* 若从设备cs1为Local bus设备，则改为"okay"
*/

dsmc_p1_region: region {
    dsmc_p1_region0: region0 {
        rockchip,attribute = "Merged FIFO";
        rockchip,ca-addr-width = <0>;
        rockchip,dummy-clk-num = <1>;
        rockchip,cs0-be-ctrl = <0>;
        rockchip,cs0-ctrl = <0>;
        status = "disabled";
    };
    dsmc_p1_region1: region1 {
        rockchip,attribute = "No-Merge FIFO";
        rockchip,ca-addr-width = <0>;
        rockchip,dummy-clk-num = <1>;
        rockchip,cs0-be-ctrl = <0>;
        rockchip,cs0-ctrl = <0>;
        status = "disabled";
    };
    dsmc_p1_region2: region2 {
        rockchip,attribute = "DPRA";
        rockchip,ca-addr-width = <0>;
        rockchip,dummy-clk-num = <1>;
        rockchip,cs0-be-ctrl = <0>;
        rockchip,cs0-ctrl = <0>;
        status = "disabled";
    };
    dsmc_p1_region3: region3 {
        rockchip,attribute = "Register";

```

```

        rockchip,ca-addr-width = <0>;
        rockchip,dummy-clk-num = <1>;
        rockchip,cs0-be-ctrlled = <0>;
        rockchip,cs0-ctrl = <0>;
        status = "disabled";
    };
};
};
dsmc_lb_slave2: lb-slave2 {
    status = "disabled";    /* 若从设备cs2为Local bus设备，则改为"okay"
*/

    dsmc_p2_region: region {
        dsmc_p2_region0: region0 {
            rockchip,attribute = "Merged FIFO";
            rockchip,ca-addr-width = <0>;
            rockchip,dummy-clk-num = <1>;
            rockchip,cs0-be-ctrlled = <0>;
            rockchip,cs0-ctrl = <0>;
            status = "disabled";
        };
        dsmc_p2_region1: region1 {
            rockchip,attribute = "No-Merge FIFO";
            rockchip,ca-addr-width = <0>;
            rockchip,dummy-clk-num = <1>;
            rockchip,cs0-be-ctrlled = <0>;
            rockchip,cs0-ctrl = <0>;
            status = "disabled";
        };
        dsmc_p2_region2: region2 {
            rockchip,attribute = "DPRA";
            rockchip,ca-addr-width = <0>;
            rockchip,dummy-clk-num = <1>;
            rockchip,cs0-be-ctrlled = <0>;
            rockchip,cs0-ctrl = <0>;
            status = "disabled";
        };
        dsmc_p2_region3: region3 {
            rockchip,attribute = "Register";
            rockchip,ca-addr-width = <0>;
            rockchip,dummy-clk-num = <1>;
            rockchip,cs0-be-ctrlled = <0>;
            rockchip,cs0-ctrl = <0>;
            status = "disabled";
        };
    };
};
dsmc_lb_slave3: lb-slave3 {
    status = "disabled";    /* 若从设备cs3为Local bus设备，则改为"okay"
*/

    dsmc_p3_region: region {
        dsmc_p3_region0: region0 {
            rockchip,attribute = "Merged FIFO";
            rockchip,ca-addr-width = <0>;
            rockchip,dummy-clk-num = <1>;
            rockchip,cs0-be-ctrlled = <0>;

```

```

        rockchip,cs0-ctrl = <0>;
        status = "disabled";
    };

    dsmc_p3_region1: region1 {
        rockchip,attribute = "No-Merge FIFO";
        rockchip,ca-addr-width = <0>;
        rockchip,dummy-clk-num = <1>;
        rockchip,cs0-be-ctrled = <0>;
        rockchip,cs0-ctrl = <0>;
        status = "disabled";
    };

    dsmc_p3_region2: region2 {
        rockchip,attribute = "DPRA";
        rockchip,ca-addr-width = <0>;
        rockchip,dummy-clk-num = <1>;
        rockchip,cs0-be-ctrled = <0>;
        rockchip,cs0-ctrl = <0>;
        status = "disabled";
    };

    dsmc_p3_region3: region3 {
        rockchip,attribute = "Register";
        rockchip,ca-addr-width = <0>;
        rockchip,dummy-clk-num = <1>;
        rockchip,cs0-be-ctrled = <0>;
        rockchip,cs0-ctrl = <0>;
        status = "disabled";
    };

};

};

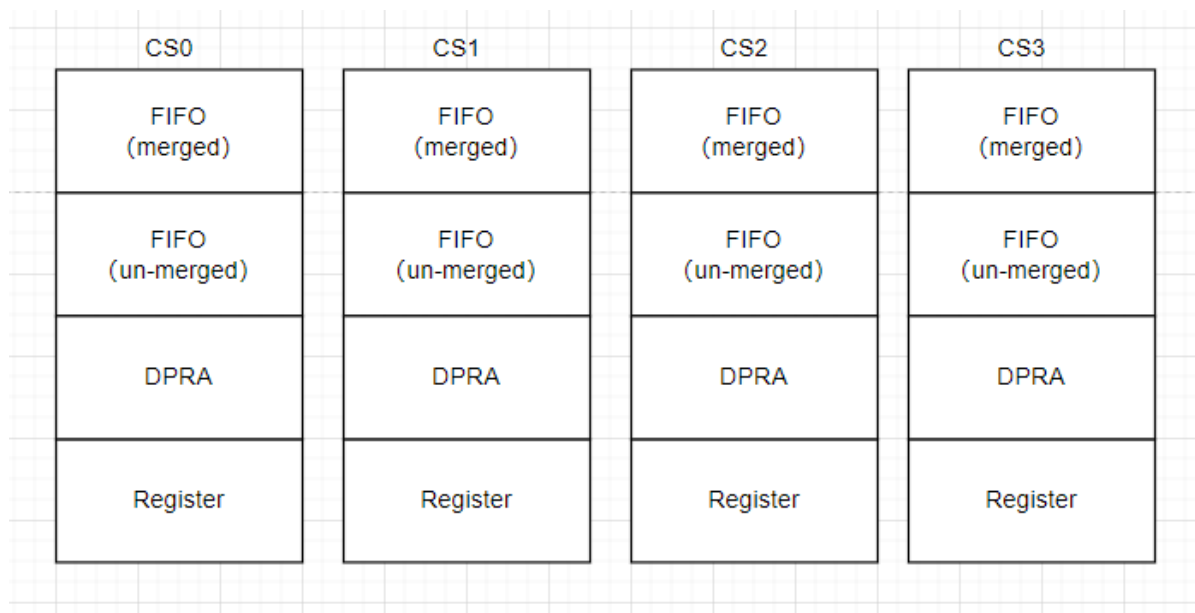
};

```

用户需根据实际从设备的类型，开启对应的节点。如PSRAM设备，需根据PCB外接CS的序号，开启对应psramx节点。具体是Hyperbus还是Xccela Psram，由驱动自动识别。当从设备是RK设计 DSMC slave时，根据PCB外接CS的序号，开启对应 lb\_slavex 节点。另外还需根据从设备的属性修改对应region的配置。其中 clk-mode 控制支持的三种clk行为：clk-mode = 0 即 CS 拉高期间无时钟，拉低期间有时钟；clk-mode = 1 即无论 CS 怎么变化，时钟一直有， slave 可将其作为参考时钟，但是此模式下无法跑高频且各类 AC timing 可调参数不可用；clk-mode = 2 即 CS拉高和拉低期间都有时钟，但是CS跳变沿前后会关闭几个时钟。

作为 Local bus 设备时，设备空间示意图如下：

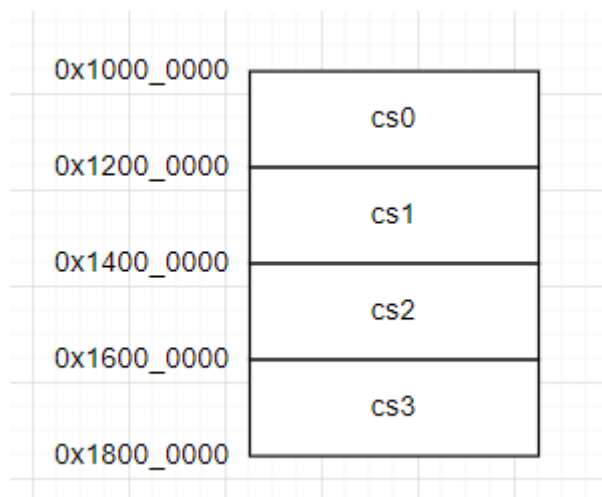




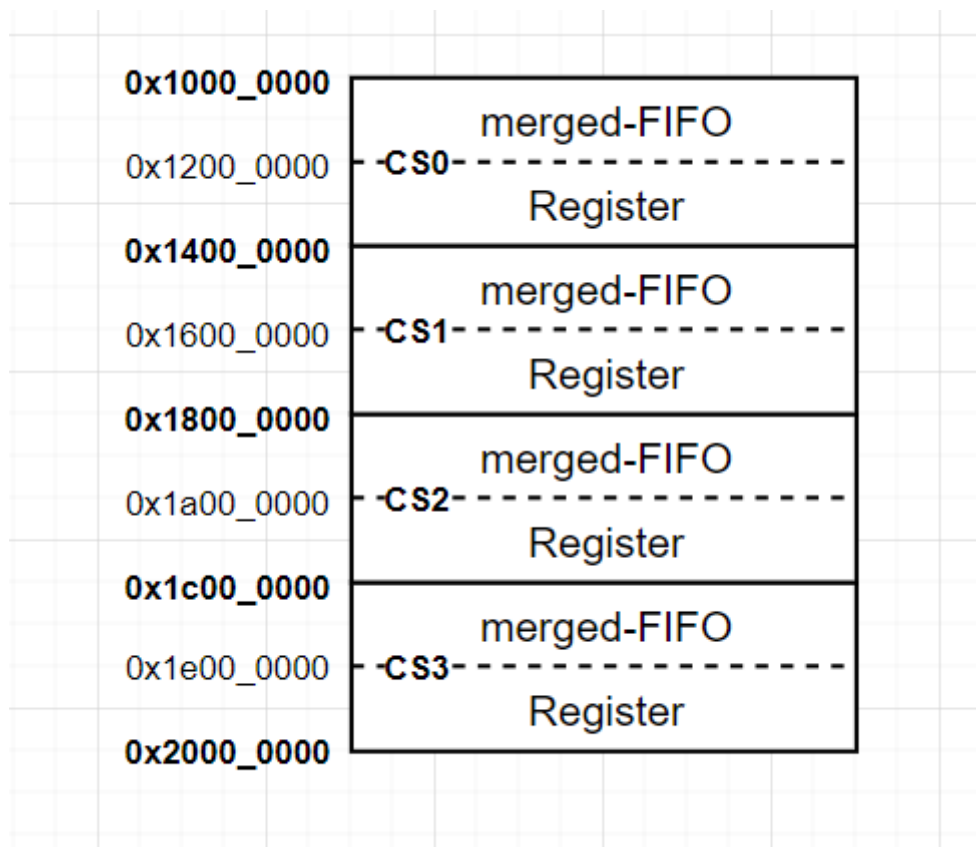
每个从设备片选CS的访问空间都可以分成1、2、4个region（均分），只需要在DTS开启对应属性region的status。

对于 `rockchip,ranges = <0x0 0x10000000 0x0 0x2000000>`; 属性，配置的是从设备内存空间的起始地址和大小，不同外设类型有不同含义。

若外设是PSRAM，`rockchip,ranges` 配置的是最大CS空间的大小。每个CS的内存空间划分如下：



若外设是 Local Bus，`rockchip,ranges` 配置的是最大region空间的大小。在只开启Register和merged-FIFO 2个region的情况下，每个CS的region空间划分如下：



### 3.3 内核配置

```

Symbol: ROCKCHIP_DSMC [=y]

|
| Type : bool
|
| Prompt: Rockchip DSMC (Double Data Rate Serial Memory Controller) driver
|
| Depends on: MEMORY [=y]
|
| Location:
|
| -> Device Drivers
|
| -> Memory Controller drivers (MEMORY [=y])
|
| -> Rockchip DSMC (Double Data Rate Serial Memory Controller) driver
(ROCKCHIP_DSMC [=y]) |

```

## 4. 内核态对DSMC从设备内存的访问

## 4.1 调用驱动接口

在DSMC驱动里 `drivers/memory/rockchip/dsmc-host.c` 实现了如下几种访问接口：

```
struct rockchip_dsmc_device *rockchip_dsmc_find_dev(void);

static struct dsmc_ops rockchip_dsmc_ops = {
    .read = dsmc_read,
    .write = dsmc_write,
    .copy_from = dsmc_copy_from,
    .copy_from_state = dsmc_copy_from_state,
    .copy_to = dsmc_copy_to,
    .copy_to_state = dsmc_copy_to_state,
};
```

通过调用函数 `rockchip_dsmc_find_device_by_compat()` 查找 dsmc 设备，并获取这个设备的私有参数。并通过 `dsmc_dev.ops` 实现 CPU，DMA 对 DSMC 从设备的访问。其中 `ops->read`、`ops->write` 为 CPU 读写 DSMC 从设备的内存空间。`ops->copy_from` 用于 DMA 读取从设备内存，并写入 host 端内存。`ops->copy_to` 用于 DMA 从 host 端内存写入从设备内存。

```
static void test(void)
{
    u32 cs;
    struct rockchip_dsmc_device *dsmc_dev;

    dsmc_dev = rockchip_dsmc_find_device_by_compat(rockchip_dsmc_get_compat(0));
    if (dsmc_dev == NULL) {
        printk("error: can not find dsmc device\n");
        return 0;
    }
    for (cs = 0; cs < DSMC_MAX_SLAVE_NUM; cs++) {
        if (dsmc_dev->dsmc.cfg.cs_cfg[i].device_type == DSMC_UNKNOWN_DEVICE)
            continue;
        dsmc_dev->ops->write(dsmc_dev, cs, 0, test_addr, test_data);
        /* TODO */
    }
}
```

## 4.2 直接访问

CPU 或 master 可直接访问 DSMC slave memory 空间（支持 Byte，half-word，word 的随机地址访问；支持 cacheable、uncacheable、write combine 映射方式）。

## 5. 用户态对 DSMC 的访问

---

## 5.1 通过特定节点访问

在Local bus使能的情况下，DSMC驱动会在 `/dev/dsmc/` 创建4个从设备片选CS，每个CS下会创建4个memory region，各属性分别对应DTS的描述。若需要访问 merged-FIFO，对应为region0，需操作对应片选CS下的对应region节点。具体如下：

1. 使用 `open()` 接口 打开对应region设备节点，获取文件描述符；
2. 使用 `mmap()` 系统调用将上述region设备内存映射到进程地址空间，获取映射地址；
3. 读写设备内存；
4. 使用 `munmap()` 解除内存映射；
5. 使用 `close()` 关闭文件描述符。

代码示例如下：

```
device_name = "/dev/dsmc/cs0/region0"
wantbytes = 0x200000;
memfd = open(device_name, O_RDWR | O_SYNC);
if (memfd == -1) {
    fprintf(stderr, "failed to open %s for physical memory: %s\n",
        device_name, strerror(errno));
    exit(EXIT_FAILURE);
}
/* 使用 O_SYNC 标志打开文件和 MAP_LOCKED 标志进行 mmap 操作的memory空间是 uncached 的 */
buf = (void volatile *) mmap(0, wantbytes, PROT_READ | PROT_WRITE,
    MAP_SHARED | MAP_LOCKED, memfd,
    0x0);

if (buf == MAP_FAILED) {
    fprintf(stderr, "failed to mmap %s for physical memory: %s\n",
        device_name, strerror(errno));
    exit(EXIT_FAILURE);
}
bufsize = wantbytes;
halflen = bufsize / 2;
count = halflen / sizeof(u32);
bufa = (u32v *) buf;
bufb = (u32v *) ((size_t) buf + halflen);

test_dsmc(bufa, bufb, count); /* 读写DSMC slave内存 */

if (munmap((void*)buf, wantbytes) == -1) {
    perror("munmap");
    close(memfd);
    exit(EXIT_FAILURE);
}

close(memfd);
```

## 5.2 直接访问

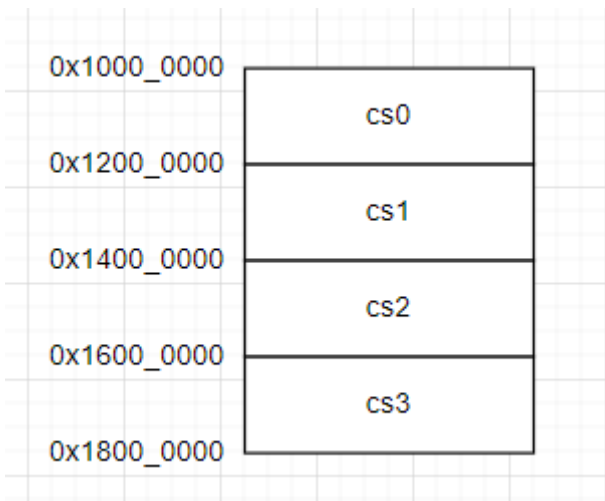
CPU可以通过 dev/mem 映射的DSMC slave memory空间直接进行访问，如RK3576上使用 io 命令进行 DSMC slave memory 的 region0 访问：io -4 0x10000000。

# 6. DSMC slave内存空间分配

DSMC slave的空间分配由 rockchip,ranges 属性控制，配置的是从设备内存空间的起始地址和大小。

## 6.1 PSRAM

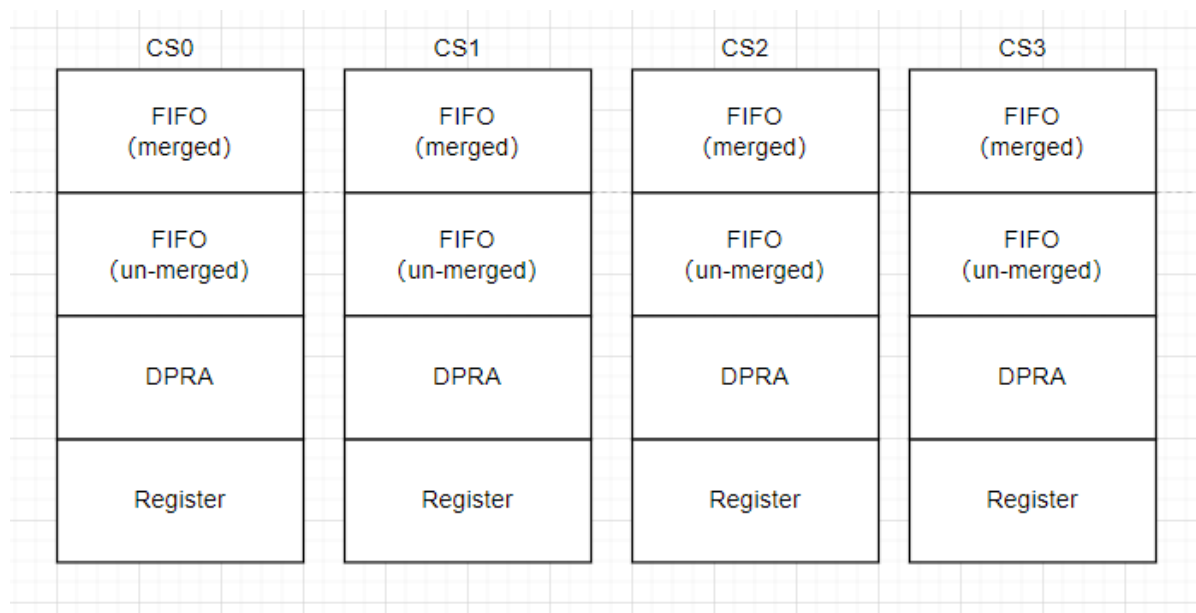
例如配置 rockchip,ranges = <0x0 0x10000000 0x0 0x2000000>;，若外设是PSRAM，rockchip,ranges 配置的是最大CS空间的大小。对DSMC控制器来说，每个片选CS的容量是相同的。若实际不同片选CS贴不同容量PSRAM，用户访问时应当注意边界。每个CS的内存空间划分如下：



Note: DSMC理论上支持不同片选CS是不同厂家PSRAM，支持容量不同，但不同片选CS的位宽（x8 or x16）必须相同。

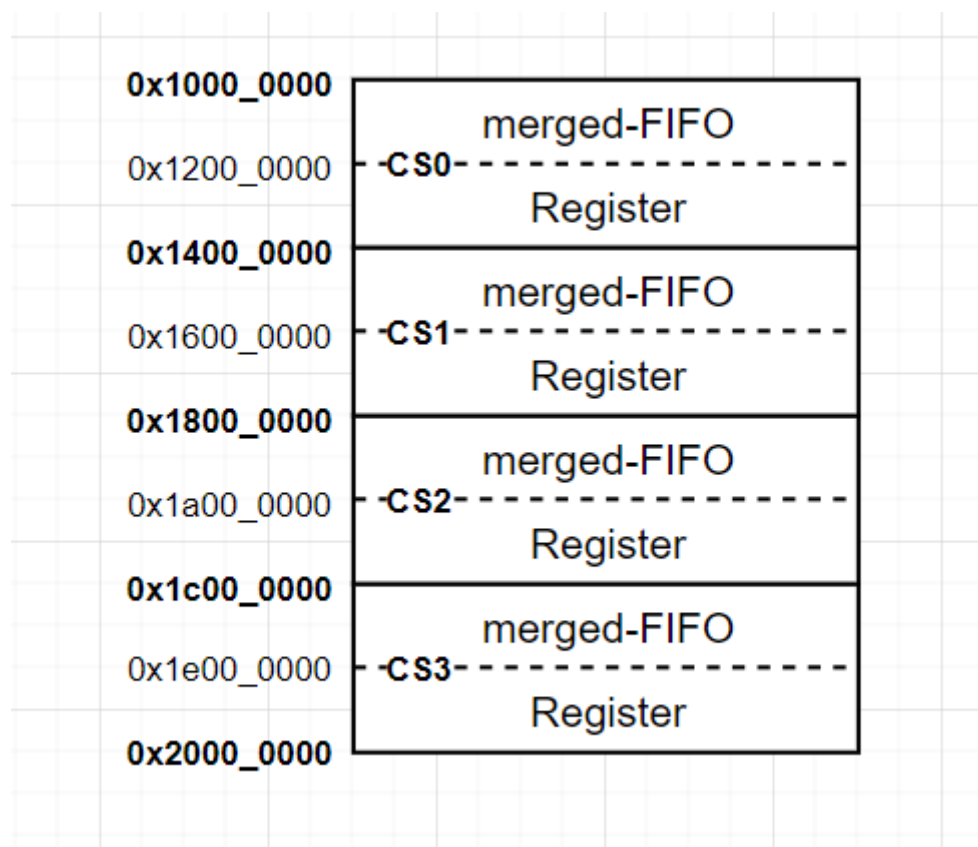
## 6.2 Local bus

若外设是 Local Bus，对于DSMC控制器来说，每个片选CS的容量也是相同的。每个片选CS都可以均分成 1、2、4个region，每个region的属性可以是DPRAM、Register、merged FIFO和un-merged FIFO。若实际每个CS的region容量不同，用户访问时应当注意边界。



对于Local Bus `rockchip, ranges` 配置的是最大region空间的大小。

若一个片选CS开启2个region，每个region大小由DTS（DSMC节点slave设备的 `rockchip, ranges = 0x0 0x10000000 0x0 0x2000000`）决定。各个CS的各region的内存空间分配如下：



## 7. DSMC Local bus host与slave的数据交互

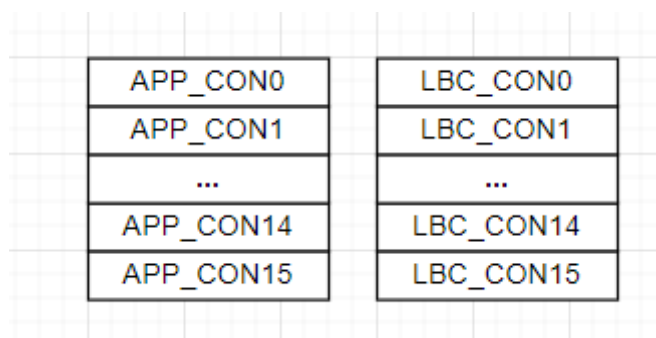
## 7.1 FIFO

当DSMC使用Local bus 协议，且外接RK slave时，slave端有一段FIFO，当访问的region属性是merged-FIFO或者un-merged-FIFO时，host端传输的数据经过FIFO后，slave端会再次写入slave端的内存，如DDR，SRAM等。这段FIFO不可见，对于DSMC host端来说，slave端的DDR，SRAM内存空间即是slave端的内存空间。通过DSMC写入的数据，最终都将被写入slave端的内存，在使用时应注意slave端内存空间的管理和数据一致性。

## 7.2 Register

当DSMC使用Local bus 协议，且外接RK slave时，有一段SLAVE\_CSR Register，当访问的region属性是Register时，即是访问这段SLAVE\_CSR Register。这段是可以用于host与slave的信息快速传递。

可用于信息交互的寄存器：



其中 APP\_CONx 寄存器，slave有读写权限，host端只有读权限；LBC\_CONx寄存器 slave只有读权限，host端有读写权限。

- 当host写入LBC\_CONx寄存器，会触发host2slave中断给slave端CPU，用于处理host端传入的数据。
- 当slave写入APP\_CONx寄存器后，通过IO引脚 INT 触发slave2host中断，传入host 端，host端CPU也可以响应中断，获取slave传递过来的数据。DSMC 接收 INT 信号后，也可以发起DMA 硬件请求，DMA开始搬移数据（DMA需提前配置好）。

### 典型场景1

host端将信息写入LBC\_CONx寄存器，将触发host2slave中断，slave端从LBC\_CONx寄存器读取信息；slave端将信息写入 APP\_CONx 寄存器，将触发slave2host中断，host端从APP\_CONx寄存器读取信息。

### 典型场景2：

host端DSMC、DMA配置好后，host端通过写LBC\_CONx寄存器通知slave端，然后slave端通过写APP\_CONx，从INT引脚返回host一个有效信号，DSMC host接收后，发起DMA 硬件请求，开启DMA搬移。

Note: APP\_CON15、LBC\_CON15 已被使用。DSMC 使用DMA硬件请求的驱动已实现，当host端DMA配置完成，host端将LBC\_CON15 原值+1 写入LBC\_CON15 寄存器，slave接收到数据后，将APP\_CON15 写1 触发slave2host中断，DSMC host接收后自动发起一定数量的DMA 硬件请求，触发DMA搬移。