3Dsim 调试文档

一、 文档说明

3Dsim 是一款针对 3D 堆叠闪存的 SSD 仿真软件，其主要功能包括：

1. 支持 3D 闪存的高级命令，例如：Mutli plane、Half page read、One shot read、One shot program、Erase suspend/resume..；
2. 多级别并行，如 channel/chip/die/plane 之间的并行；
3. 清晰的垂直结构，5 层结构，包括 interface, buffer, FTL, FCL, flash。

本文档用于记录 3Dsim 仿真器调试阶段在修改源码时遇到的错误，错误原因及解决方法。

二、 调试部分

1. 3Dsim 缓存及分配方式调试记录

1.1 代码修改

缓存部分修改和测试，主要完成的修改为将函数 distribute2_command_buffer()和函数 allocate_location()中对于 channel、chip、die 和 aim_die 的计算部分封装为一个函数，并将返回结果统一到一个结构体内，具体为：

```
35    struct allocation_info * allocation_method(struct ssd_info *ssd,unsigned int lpn);
```

图 1 allocation_method（）函数

函数名：allocation_method（）

参　数：struct ssd_info *ssd

　　　　unsigned int lpn

功　能：计算 channel/chip/die/aim_die 等参数

返回类型：struct allocation_info *

allocation_info 结构体是函数的返回类型，具体如下：

```
struct allocation_info                             //记录分配信息
{
    unsigned int channel;
    unsigned int chip;
    unsigned int die;
    unsigned int aim_die;
    unsigned int plane;
    unsigned int mount_flag;
    unsigned int block;
    unsigned int page;
    struct buffer_info * aim_command_buffer;
};
```

图 2 allocation_info 结构体

修改后的方案有以下优点：

1) 将源代码中物理分配相关参数计算的代码量减少一半，且完全封装成一个函数，直接调用即可，简化了 distribute2_command_buffer（）函数和 allocate_location（）函数中的逻辑，使得整体代码更加整洁。

2) 原代码逻辑中 allocate_location（）函数中计算 channel、chip 和 die 等参数结果时需要使用到 distribute2_command_buffer（）函数中计算的结果（aim_die），使得多个函数需要传递参数 aim_die，本次修改了消除了对于 aim_die 的依赖，不在需要层层传递 aim_die，简化了代码逻辑，具体修改见图 3。

```
struct ssd_info *no_buffer_distribute(struct ssd_info *);
struct ssd_info * getout2buffer(struct ssd_info *ssd, struct sub_request *sub, struct request *req);
struct ssd_info * check_w_buff(struct ssd_info *ssd, unsigned int lpn, int state, struct sub_request *sub, struct re
struct ssd_info * insert2buffer(struct ssd_info *ssd, unsigned int lpn, int state, struct sub_request *sub, struct re
struct sub_request * creat_sub_request(struct ssd_info * ssd, unsigned int lpn, int size, unsigned int state, struct
struct ssd_info * insert2_command_buffer(struct ssd_info * ssd, struct buffer_info * command_buffer, unsigned int lp
struct ssd_info * distribute2_command_buffer(struct ssd_info * ssd, unsigned int lpn, int size_count, unsigned int s
struct allocation_info * allocation_method(struct ssd_info *ssd,unsigned int lpn);


unsigned int size(unsigned int);
__int64 calculate_distance(struct ssd_info * ssd, struct buffer_info * die_buffer, unsigned int lpn);
Status allocate_location(struct ssd_info * ssd, struct sub_request *sub_req);
```

图 3 消除 unsigned int die_number 参数的函数

1.2 trace 测试结果

测试 trace 共 19 个（16GB），所有结果和修改前相同，测试使用 trace 如图 4 所示

```
//trace 路径名
char *trace_file[19] =
{
    "exchange.ascii", "fiu_web.ascii", "hm0.ascii", "hm1.ascii", "proj0.ascii", "proj3.ascii", "rsrch0.ascii", "src0.ascii", "src1.ascii",
    "ts0.ascii", "usr0.ascii", "vps.ascii", "w1.ascii", "w2.ascii", "wdev0.ascii","mds1.ascii","stg1.ascii","usr2.ascii","web2.ascii"
};
```

图 4 测试 trace

部分 trace 结果对比如下：

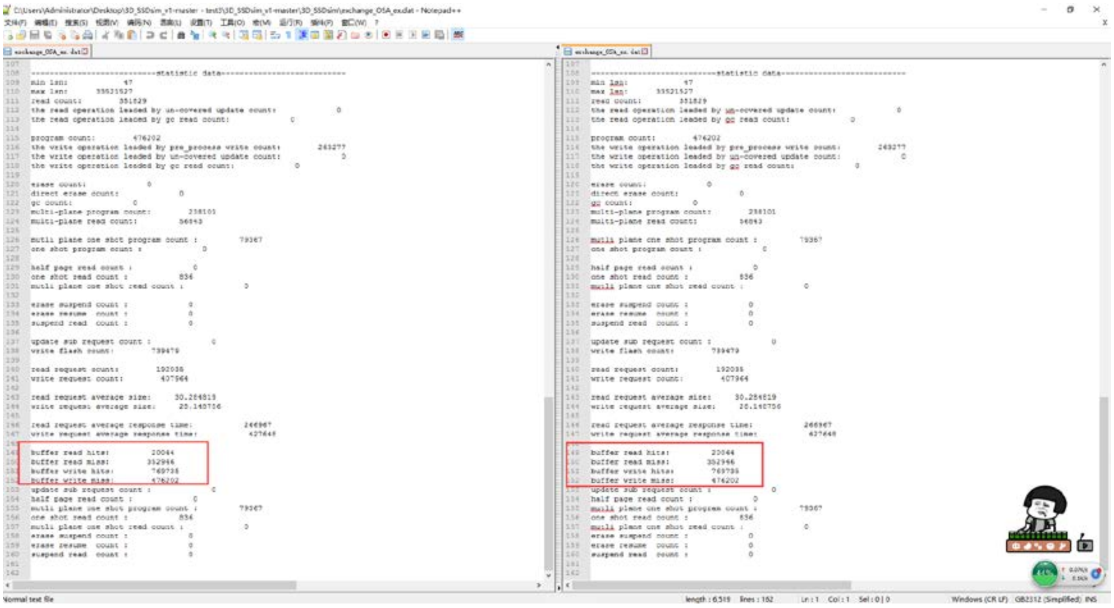

图 1 trace：exchange.ascii，修改后和修改前结果相同

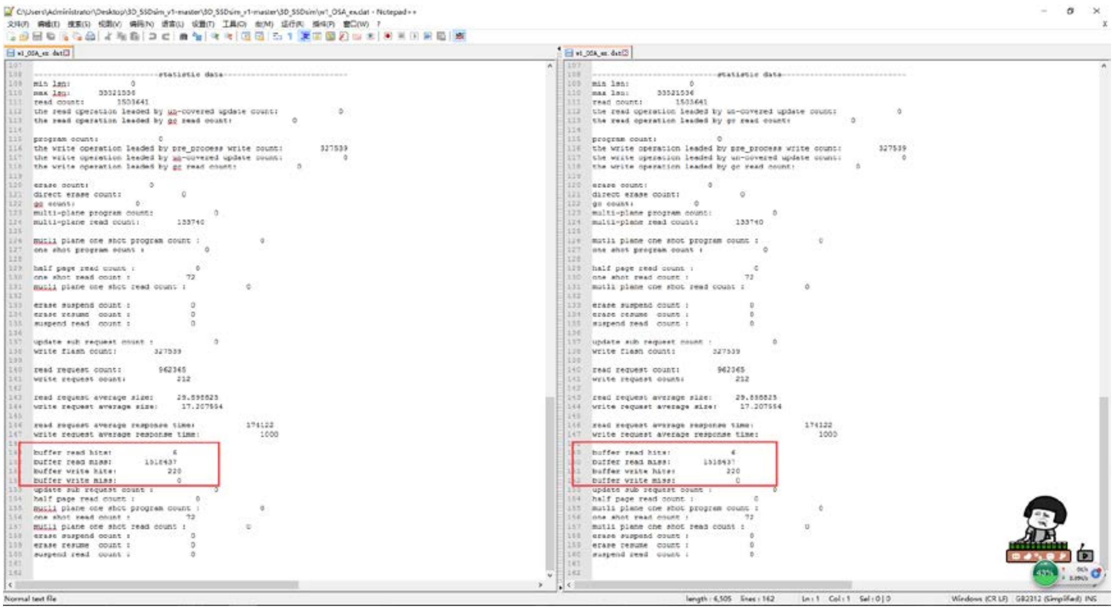

图 2 trace：w1.ascii，修改后和修改前结果相同

2. 3Dsim 预处理部分调试记录

2.1 warm_flash 参数

修改：参数中添加 warm_flash

位置：Initialize.h

源码位置截图：

```
float aged_ratio;
int queue_length;                   //Request the length of the queue
int warm_flash;
int update_reqeust_max;             //request the length of sub request(partial page)
int flash_mode;                     //0--slc mode,1--tlc mode
```

修改：读参数文件时读入 warm_flash

位置：Initialize.c

函数：load_parameters（）

源码位置截图：

```
    sscanf(buf + next_eql,"%d",&p->queue_length);          //Request the queue depth
}else if((res_eql=strcmp(buf,"warm flash")) == 0){
    sscanf(buf + next_eql, "%d", &p->warm_flash);
}else if((res_eql=strncmp(buf,"chip number",11)) ==0)
{
```

2.2 warm_flash

修改：ssd_info 结构中增加 warm_flash_cmplt 作为 warm_flash 控制

位置：initialize.h

源码位置截图：

```
struct ssd_info{
    //Global variable

    //unsigned int gc_num[6];

    int make_age_free_page;
    int buffer_full_flag;
    int trace_over_flag;
    __int64 request_lz_count;
    unsigned int update_sub_request;
    unsigned int page_count;
    int test_count;
    unsigned int die_token;
    unsigned int plane_count;
    int warm_flash_cmplt;
```

修改：将 warm_flash 加入处理过程中

位置：ssd.c

函数： tracefile_sim（）

源码位置截图：

```c
	pre_process_page(ssd);

	if(ssd->parameter->warm_flash == 1)
		warm_flash(ssd);

	/*
	make_aged(ssd);

	pre_process_page(ssd);

	if (ssd->parameter->aged == 1)
		pre_process_write(ssd);
	*/
	//After the preprocessing is complete, the page offset address of each plane should be guaranteed to be consistent
	for (i=0;i<ssd->parameter->channel_number;i++)
```

修改： 增加 warm_flash 函数

位置： ssd.c

函数： warm_flah（）

源码位置截图：

```c
struct ssd_info *warm_flash(struct ssd_info *ssd)
{
	int flag = 1;
	double output_step = 0;
	unsigned int a = 0, b = 0;
	errno_t err;
	unsigned int i, j, k, m, p;
```

2.3 flush_all

修改： 在 buffer 中创建 flush_all（）函数，在 warm_flash（）中被调用

位置： buffer.c

函数： flush_all（）

源码位置截图：

```c
struct ssd_info *flush_all(struct ssd_info *ssd)
{

	struct buffer_group *pt;
	struct sub_request *sub_req = NULL;
	unsigned int sub_req_state = 0, sub_req_size = 0, sub_req_lpn = 0;

	struct request *req = (struct request*)malloc(sizeof(struct request));
	alloc_assert(req, "request");
```

2.4 trace 读取

修改： 修改 warm_flash 过程中读取 tarce 的过程

位置： interface.c

函数： get_request（）

源码位置截图：

```c
	ope = 0;

	while (TRUE)
	{
		if (ssd->warm_flash_cmplt == 1)
		{
			filepoint = ftell(ssd->tracefile);
			fgets(buffer, 200, ssd->tracefile);
			sscanf(buffer, "%I64u %d %d %d %d", &time_t, &device, &lsn, &size, &ope);
		}
		else
		{
			while (ope != 1)
			{
				if (feof(ssd->tracefile))
					break;
				filepoint = ftell(ssd->tracefile);
				fgets(buffer, 200, ssd->tracefile);
				sscanf(buffer, "%I64u %d %d %d %d", &time_t, &device, &lsn, &size, &ope);
			}
			ope = 0;
		}
		if (ssd->parameter->dram_capacity == 0)
			break;
		if (size < (ssd->parameter->dram_capacity / SECTOR))
			break;

		if (feof(ssd->tracefile))      //if the end of trace
			break;

	}
```

2.5 command_buffer

修改： 改变从 command_buffer 中下发请求的方式

位置： buffer.c

函数： insert2_command_buffer（）

源码位置截图：

```c
		//如果缓存已满，此时发生flush操作，将缓存的内存一次性flush到闪存上
		if (command_buffer->command_buff_page >= command_buffer->max_command_buff_page)
		{
			if (ssd->warm_flash_cmplt == 0)
				tmp = command_buffer->command_buff_page;
			else
				tmp = command_buffer->max_command_buff_page;
			//printf("begin to flush command_buffer\n");
			for (i = 0; i < tmp; i++)
			{
				sub_req = NULL;
				sub_req_state = command_buffer->buffer_tail->stored;
				sub_req_size = size(command_buffer->buffer_tail->stored);
```

2.6 initialize_statistic

修改：增加初始化的项目

位置：initialize.c

函数：initialize_statistic ()

源码位置截图：

```
void initialize_statistic(struct ssd_info * ssd)
{
    //Initialize parameters
    ssd->read_count = 0;
    ssd->update_read_count = 0;
    ssd->gc_read_count = 0;
    ssd->program_count = 0;
    ssd->pre_all_write = 0;
    ssd->update_write_count = 0;
    ssd->gc_write_count = 0;
    ssd->erase_count = 0;
    ssd->direct_erase_count = 0;
    ssd->m_plane_read_count = 0;
    ssd->read_request_count = 0;
    ssd->write_flash_count = 0;
    ssd->write_request_count = 0;
    ssd->read_request_count = 0;
    ssd->ave_read_size = 0.0;
    ssd->ave_write_size = 0.0;

    ssd->gc_count = 0;
    ssd->mplane_erase_count = 0;

    //Initializes the global variable for ssd_info
    ssd->make_age_free_page = 0;
    ssd->buffer_full_flag = 0;
    ssd->request_lz_count = 0;
```

2.7 service_advance_command

修改：当子请求不满足 3 个时，使用 get_ppn_for_normal_command 处理

位置：fcl.c

函数：service_advance_command ()

源码位置截图：

```c
//根据不同的高级命令去凑足请求的个数
Status service_advance_command(struct ssd_info *ssd, unsigned int channel, unsigned int c
{
    unsigned int i = 0;
    unsigned int max_sub_num;
    struct sub_request *sub = NULL, *p = NULL;

    max_sub_num = (ssd->parameter->die_chip)*(ssd->parameter->plane_die)*PAGE_INDEX;

    //当最后读完了，只剩下了单个请求，这个时候可以使用普通的one page program去写完
    if (ssd->trace_over_flag == 1 && ssd->request_work == NULL)
    {
        for (i = 0; i < subs_count;i++)
            get_ppn_for_normal_command(ssd, channel, chip, subs[i]);

        return SUCCESS;
    }

    if (subs_count >= aim_subs_count)
    {
        for (i = aim_subs_count; i < subs_count; i++)
            subs[i] = NULL;

        subs_count = aim_subs_count;
        get_ppn_for_advanced_commands(ssd, channel, chip, subs, subs_count, command);
    }
    else if (subs_count > 0)
    {
        for (i = 0; i < subs_count; i++)
            get_ppn_for_normal_command(ssd, channel, chip, subs[i]);

        return SUCCESS;
        /*
```

3. 3Dsim GC 处理部分调试记录

3.1 修改原因

原 3Dsim 代码未实现被动 GC 的硬阈值与软阈值，所以本次修改新增了 GC 的硬阈值与软阈值。

3.2 修改记录

1) 修改 main（）函数，使程序可以选择批处理 trace 文件或单个 trace 文件，以及选择 512M 的配置文件或是 16G 的 trace 文件。代码截图如下：

```c
void main()
{
    unsigned int i = 0, j = 0;
    unsigned int flag1,a;
    struct ssd_info *ssd;

    //for (j = 0; j < 1; j++)
    //{
    a = 0;//0:512M 配置; 1:16G配置
    flag1 = 1;//0:处理单个trace;1:批处理trace
    j = 1;
    if (flag1 == 1)
```

2) 修改被动 GC，在 get_ppn()函数和 find_level_page()函数中设置硬阈值与软阈值。代码截图如下：

修改 get_ppn（）函数：

```c
ssd->channel_head[channel].gc_soft = 0;
ssd->channel_head[channel].gc_hard = 0;

if (ssd->parameter->active_write == 0)
{                                                    /*If the number of free_page in plane is less than the threshold set by gc_hard_thresho
    if (ssd->channel_head[channel].chip_head[chip].die_head[die].plane_head[plane].free_page<(ssd->parameter->page_block*ssd->parameter->block_plane*ssd->parameter->gc
    {
        ssd->channel_head[channel].gc_soft = 1;
        if (ssd->channel_head[channel].chip_head[chip].die_head[die].plane_head[i].free_page < (ssd->parameter->page_block*ssd->parameter->block_plane*ssd->parameter->
        {
            ssd->channel_head[channel].gc_hard = 1;
        }
```

修改 find_level_page（）函数

```c
gc_add = 0;
ssd->channel_head[channel].gc_soft = 0;
ssd->channel_head[channel].gc_hard = 0;
for (i = 0; i < ssd->parameter->plane_die; i++)
{
    if (ssd->channel_head[channel].chip_head[chip].die_head[die].plane_head[i].free_page <= (ssd->parameter->page_block*ssd->parameter->block_plane*ssd->parameter->gc_
    {
        ssd->channel_head[channel].gc_soft = 1;
        gc_add = 1;
        if (ssd->channel_head[channel].chip_head[chip].die_head[die].plane_head[i].free_page <= (ssd->parameter->page_block*ssd->parameter->block_plane*ssd->parameter-
        {
            ssd->channel_head[channel].gc_hard = 1;
        }
    }
}
```

3) 添加执行被动 GC 的函数。代码截图如下

```c
if (ssd->gc_request > 0)
{
    if (ssd->channel_head[i].gc_soft == 1 && ssd->channel_head[i].gc_hard == 1 && ssd->channel_head[i].gc_command != NULL)
    {
        flag_gc = gc(ssd, i, 0);
        if (flag_gc == 1)
        {
            //ssd->channel_head[i].gc_soft = 0;
            //ssd->channel_head[i].gc_hard = 0;
            continue;
        }
    }
}
```

4） 修改 gc_for_channel（）函数，代码截图如下：

```
if (gc_node->priority == GC_UNINTERRUPT /*&& hard == 1 && soft == 1*/)
{
    flag_direct_erase = gc_direct_erase(ssd, channel, chip, die);
    if (flag_direct_erase != SUCCESS)
    {
        flag_gc = greedy_gc(ssd, channel, chip, die);
        if (flag_gc == SUCCESS)
        {
            delete_gc_node(ssd, channel, gc_node);
        }
        else
        {
            return FAILURE;
        }

    }
    else
    {
        delete_gc_node(ssd, channel, gc_node);
    }
    return SUCCESS;
}
```

3.3 调试记录

1） 问题：配置文件中软硬阈值设置较高，程序运行时间较长。

解决方法：同意配置软阈值=0.2，硬阈值=0.1；

```
gc hard threshold=0.1;
gc soft threshold=0.2;
```

2） 问题：源代码 gc_for_channel()函数逻辑不正确，返回值有误；

解决办法：修改 gc_for_channle()函数的返回值，新增 return FAILURE。

```
if (gc_node->priority == GC_UNINTERRUPT /*&& hard == 1 && soft == 1*/)
{
    flag_direct_erase = gc_direct_erase(ssd, channel, chip, die);
    if (flag_direct_erase != SUCCESS)
    {
        flag_gc = greedy_gc(ssd, channel, chip, die);
        if (flag_gc == SUCCESS)
        {
            delete_gc_node(ssd, channel, gc_node);
        }
        else
        {
            return FAILURE;
        }

    }
    else
    {
        delete_gc_node(ssd, channel, gc_node);
    }
    return SUCCESS;
}
```

3） 问题：程序运行 gc 的位置刚开始是在写请求做完后进行判断，后来测试发现这样做程序的逻辑不对，可能不会产生 gc，因为时间线上可能还有其他请求没做完。

解决方法：将 gc 放在读写请求之前，在读写之前判断 gc 是否达到硬阈值。

```
if (ssd->gc_request > 0)
{
    if (ssd->channel_head[i].gc_soft == 1 && ssd->channel_head[i].gc_hard == 1 && ssd->channel_head[i].gc_command != NULL)
    {
        flag_gc = gc(ssd, i, 0);
        if (flag_gc == 1)
        {
            //ssd->channel_head[i].gc_soft = 0;
            //ssd->channel_head[i].gc_hard = 0;
            continue;
        }
    }
}
```

4） 问题：全是读请求或写请求时，trace 输出有问题。

解决方法：计算读写平均请求时间时，判断是否是全读或全写，若是，则跳过输出。

```
if(ssd->read_request_count != 0)
    fprintf(ssd->outputfile,"read request average response time: %16I64u\n",ssd->read_avg/ssd->read_request_count);
if(ssd->write_request_count != 0)
    fprintf(ssd->outputfile,"write request average response time: %16I64u\n",ssd->write_avg/ssd->write_request_count);
```

5） 问题：之前判断被动 gc 退出的条件是空闲页的数量大于 10%，但由于 gc 的粒度是块，所以做一次 gc 就能保证空闲页的数量，但之前还对 gc 次数进行

判断，有点冗余。

解决方法：直接做 gc，不进行判断。

```
if (ssd->gc_request > 0)
{
    if (ssd->channel_head[i].gc_soft == 1 && ssd->channel_head[i].gc_hard == 1 && ssd->channel_head[i].gc_command != NULL)
    {
        flag_gc = gc(ssd, i, 0);
        if (flag_gc == 1)
        {
            //ssd->channel_head[i].gc_soft = 0;
            //ssd->channel_head[i].gc_hard = 0;
            continue;
        }
    }
}
```

6）问题：在之前的 get_page_level（）函数判断硬阈值与软阈值是所有 plane 都小于这个预先设定的值，这样可能会出现有的 plane 空闲页不足。

解决方法：若有一个 plane 小于硬阈值或软阈值，就进行相应处理。

```
gc_add = 0;
ssd->channel_head[channel].gc_soft = 0;
ssd->channel_head[channel].gc_hard = 0;
for (i = 0; i < ssd->parameter->plane_die; i++)
{
    if (ssd->channel_head[channel].chip_head[chip].die_head[die].plane_head[i].free_page <= (ssd->parameter->page_block*ssd->parameter->block_plane*ssd->parameter->gc_
    {
        ssd->channel_head[channel].gc_soft = 1;
        gc_add = 1;
        if (ssd->channel_head[channel].chip_head[chip].die_head[die].plane_head[i].free_page <= (ssd->parameter->page_block*ssd->parameter->block_plane*ssd->parameter-
        {
            ssd->channel_head[channel].gc_hard = 1;
        }
    }
}
```

7）问题：Financial1 测试不能通过，当硬阈值与软阈值设置较小时。

解决办法：适当增加硬阈值与软阈值的值，但不要太大，避免程序运行时间过长。

8）问题：Financial2 测试时间异常大。

解决办法：trace 中的小写请求特别多，导致 gc 次数过多，这种异常属于正常情况。

9）问题：512M trace 的 4K 和 16K 顺序写 trace 跑不通。

解决办法：原因应该是 trace 生成的小写请求特别多，以至于找不到空闲页因而报错，所以应适当增加硬阈值和软阈值来请求更多的 gc。

3.4 调试结果

1） 512M

配置文件：

```
#parameter file
dram capacity = 524288;     #the unit is B
chip number[0] = 2;
chip number[1] = 2;
chip number[2] = 0;
chip number[3] = 0;
chip number[4] = 0;
chip number[5] = 0;
chip number[6] = 0;
chip number[7] = 0;
chip number[8] = 0;
chip number[9] = 0;
chip number[10] = 0;
chip number[11] = 0;
chip number[12] = 0;
chip number[13] = 0;
chip number[14] = 0;
chip number[15] = 0;
chip number[16] = 0;
chip number[17] = 0;
chip number[18] = 0;
channel number = 2;   #the number of channel
chip number = 4;
die number = 1;
plane number = 2;
block number = 32;
page number = 64;
subpage page = 4;                     #can not beyond 32
page capacity = 16384;                #16kb
subpage capacity = 4096;
t_PROG = 1100000;            #the unit is ns
t_DBSY = 500;
t_BERS = 10000000;
t_PROGO = 1100000;           #one shot program time
t_ERSL = 500000;            #the trans time of suspend/resume operation
t_R = 90000;
t_WC = 5;
t_RC = 5;
```

```
t_CLS = 12;
t_CLH = 5;
t_CS = 20;
t_CH = 5;
t_WP = 12;
t_ALS = 12;
t_ALH = 5;
t_DS = 12;
t_DH = 5;
t_WH = 10;
t_ADL = 70;
t_AR = 10;
t_CLR = 10;
t_RR = 20;
t_RP = 12;
t_WB = 100;
t_REA = 30;
t_CEA = 45;
t_RHZ = 100;
t_CHZ = 30;
t_RHOH = 15;
t_RLOH = 5;
t_COH = 15;
t_REH = 10;
t_IR = 0;
t_RHW = 100;
t_WHR = 60;
t_RST = 5000;
erase limit=100000;            #记录block最大次数擦写次数
overprovide=0.20;              #op空间大小
requset queue depth=8;         #请求队列深度
scheduling algorithm=1;        #记录使用哪种调度算法，1:FCFS
buffer management=0;           #缓存策略：0：sub_page拼凑
address mapping=1;             #映射表策略：1：page；2：4kb_map(目前只支持page-level)
wear leveling=1;               #WL算法
gc=1;                          #gc策略：1：superblock
gc hard threshold=0.1;         #gc硬阈值大小，当plane内无效页个数超过此阈值时，触发gc操作
allocation=2;                  #分配方式，0：动态分配，1：静态分配，2：基于负载感知分配
static_allocation=2;           #tlc mode 静态分配方式类型，0：plane>superpage>channel>chip>die, 1:superpage>plane>channel>chip>die, 2:channel>chip>plane>superpage>die, 3:channel>chip>superpage>plane>die
```

dynamic_allocation=2;   #slc mode 动态分配方式优先级，0：channel>chip>die>plane,1 ：plane>channel>chip>die,2:stripe_poll,3:stripe_distance,4:poll+distance
advanced command=13;   #高级命令支持，用二进制表示，无(00000)、mutli plane(00001), half-page-read(00010), one shot program(00100),one shot read(01000),erase suspend/resume(10000)
greed MPW command=1;   #multi-plane 贪心算法，0：非贪心，1：贪心
aged=0;   #旧化处理，0：non-aged，1：aged
aged ratio=0.5;   #旧化率
flash mode=1;   #flash 支持模式，0：slc,1：tlc

Trace—Public—Financial2—4KB 512M

测试结果见 Excel 表格。

Trace—User—4KB 512M

测试结果见 Excel 表格

Trace—User—16KB 512M

测试结果见 Excel 表格

Trace—Public—16KB 512M

测试结果见 Excel 表格。

2) 16G

配置文件：

```
-----------------------parameter file----------------------
#parameter file
dram capacity = 16777216;     #the unit is B
chip number[0] = 2;
chip number[1] = 2;
chip number[2] = 0;
chip number[3] = 0;
chip number[4] = 0;
chip number[5] = 0;
chip number[6] = 0;
chip number[7] = 0;
chip number[8] = 0;
chip number[9] = 0;
chip number[10] = 0;
chip number[11] = 0;
chip number[12] = 0;
chip number[13] = 0;
```

```
chip number[14] = 0;
chip number[15] = 0;
chip number[16] = 0;
chip number[17] = 0;
chip number[18] = 0;
channel number = 2;   #the number of channel
chip number = 4;
die number = 1;
plane number = 2;
block number = 2048;
page number = 64;
subpage page = 4;                       #can not beyond 32
page capacity = 16384;                  #16kb
subpage capacity = 4096;
t_PROG = 1100000;             #the unit is ns
t_DBSY = 500;
t_BERS = 10000000;
t_PROGO = 1100000;            #one shot program time
t_ERSL = 500000;             #the trans time of suspend/resume operation
t_R = 90000;
t_WC = 5;
t_RC = 5;
t_CLS = 12;
t_CLH = 5;
t_CS = 20;
t_CH = 5;
t_WP = 12;
t_ALS = 12;
t_ALH = 5;
t_DS = 12;
t_DH = 5;
t_WH = 10;
t_ADL = 70;
t_AR = 10;
t_CLR = 10;
t_RR = 20;
t_RP = 12;
t_WB = 100;
t_REA = 30;
t_CEA = 45;
t_RHZ = 100;
t_CHZ = 30;
t_RHOH = 15;
t_RLOH = 5;
```

t_COH = 15;
t_REH = 10;
t_IR = 0;
t_RHW = 100;
t_WHR = 60;
t_RST = 5000;
erase limit=100000; #记录block最大次数擦写次数
overprovide=0.20; #op空间大小
requset queue depth=8; #请求队列深度
scheduling algorithm=1; #记录使用哪种调度算法，1:FCFS
buffer management=0; #缓存策略：0：sub_page拼凑
address mapping=1; #映射表策略：1：page；2：4kb_map(目前只支持page-level)
wear leveling=1; #WL算法
gc=1; #gc策略：1：superblock
gc hard threshold=0.1;
allocation=2; #分配方式，0：动态分配，1：静态分配，2：基于负载感知分配
static_allocation=2; #tlc mode 静态分配方式类型，0：plane>superpage>channel>chip>die, 1:superpage>plane>channel>chip>die, 2:channel>chip>plane>superpage>die, 3:channel>chip>superpage>plane>die
dynamic_allocation=2; #slc mode 动态分配方式优先级，0：channel>chip>die>plane,1：plane>channel>chip>die,2:stripe_poll,3:stripe_distance,4:poll+distance
advanced command=13; #高级命令支持，用二进制表示，无(00000)、mutli plane(00001), half-page-read(00010), one shot program(00100),one shot read(01000),erase suspend/resume(10000)
greed MPW command=1; #multi-plane贪心算法，0：非贪心，1：贪心
aged=0; #旧化处理，0：non-aged，1：aged
aged ratio=0.5; #旧化率
flash mode=1; #flash支持模式，0：slc,1：tlc

Trace—User—16KB 16G

测试结果见Excel表格

Trace—Public—4KB 16G

测试结果见Excel表格。

Trace—Public—16KB 16G

测试结果见Excel表格。