地平线
Horizon Robotics

X3J3

地平线系统软件DDR分区与定制修改介绍

V1.1

2020-11

免责声明

本文档信息仅用于帮助系统和软件使用人员使用地平线产品。本文档信息未以明示或暗示方式授权他人基于本文档信息设计或制造任何集成电路。

本文档中的信息如有更改，恕不另行通知。尽管本文档已尽可能确保内容的准确性，本文档中的所有声明、信息和建议均不构成任何明示或暗示的保证、陈述或担保。

本文档中的所有信息均按“原样”提供。地平线不就其产品在任何特定用途的适销性、适用性以及不侵犯任何第三方知识产权方面做出任何明示或暗示保证、陈述或担保。地平线不承担产品使用所引起的任何责任，包括但不限于直接或间接损失赔偿。

买方和正在基于地平线产品进行开发的其他方（以下统称为“用户”）理解并同意，用户在设计产品应用时应承担独立分析、评估和判断的责任。用户应对其应用（以及用于其应用的所有地平线产品）的安全性承担全部责任，并保证符合所有适用法规、法律和其它规定的要求。地平线产品简介和产品规格中提供的“典型”参数在不同应用下可能会不同，实际性能也可能随时间而变化。所有工作参数，包括“典型”参数，都必须由用户自己针对每项用户应用进行验证。

用户同意如因用户未经授权使用地平线产品或因不遵守本说明中的条款，造成任何索赔、损害、成本、损失和（或）责任，用户将为地平线及其代表提供全额赔偿。

修订记录

版本	修订日期	修订说明
1.0	2020-11-14	初始版本
1.1	2021-02-20	添加 4.7 眼图工具 4.8 展频 4.6.1 BOARD-ID 选择 DDR 参数章节

目录

目录

1 范围

本文描述了 X3J3 系列芯片 DDR 分区基本情况，以及如何添加一套 DDR 参数，包括 DDR 分区的布局，DDR 各参数的作用，DDR 参数对应的文件，以及如何添加一套 DDR 参数

本文以海力士 lpddr4 为主介绍了添加 DDR 参数的步骤，其他厂商的情况与此类似。

2 术语、定义和缩略语

2.1 术语和定义

本 DDR 编程指南提供了系统启动和 DDR 代码体系结构的概述以及添加 DDR 参数详细编程过程。 允许客户修改和添加多组 DDR 参数，以支持各种 DDR 颗粒。 有关 DDR 参数的详细定义，请参考文档“ X3J3 DDR Tuning Guide”。

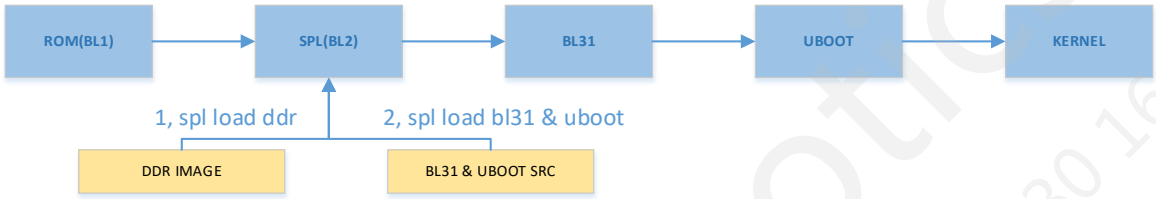
2.2 缩略语

缩略语	英文全称	中文解释
SoC	System on Chip	片上系统
SPL	Secondary Program Loader	二级程序加载器
DDR	Double Data Rate SDRAM	双倍速率同步动态随机存储器
LPDDR	Low Power Double Data Rate SDRAM	低功耗双倍数据速率内存

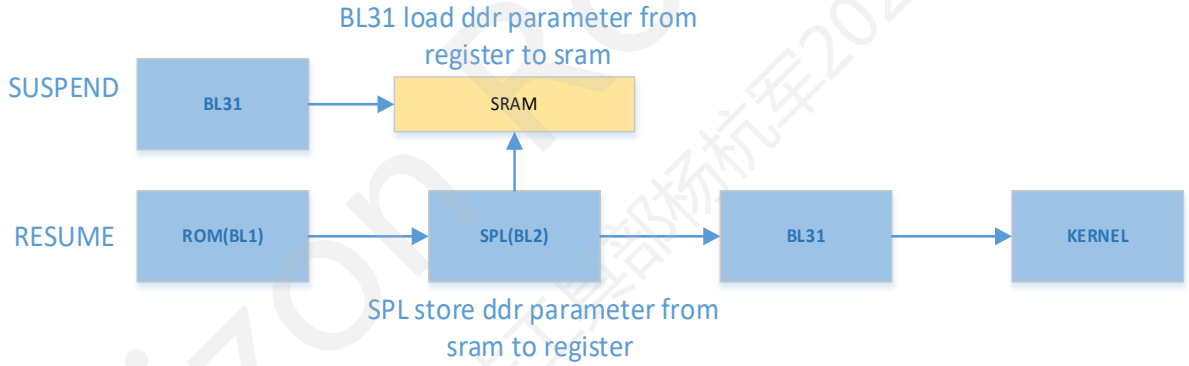
3 系统启动流程及 DDR 分区的介绍

3.1 系统启动流程

系统启动有两种情况：冷启动和热启动；冷启动过程从上电复位开始，系统将从 BOOTROM 代码开始执行，BOOTROM 代码是存储在片上 ROM 中的代码，执行 BOOTROM 代码后，x3J3 依次开始执行 SPL，BL31 和 UBOOT。 DDR 镜像、BL31 和 UBOOT 镜像都由 SPL 加载。DDR 镜像用于设置 DDR 参数。



热启动也就是休眠唤醒，除此以外的启动都是冷启动。在休眠之前，BL31 将 DDR 参数从 DDR 寄存器中读出保存到 SRAM 中，在唤醒的时候，SPL 负责将 SRAM 中的 DDR 参数再写回到 DDR 寄存器中。本文讨论的 DDR 参数是在冷启动所用。



3.2 DDR 分区架构

图 3 是镜像的分区示意图。DDR 镜像的位置记录在 MBR 中。 SPL 读取 MBR 获得 DDR 镜像的位置。

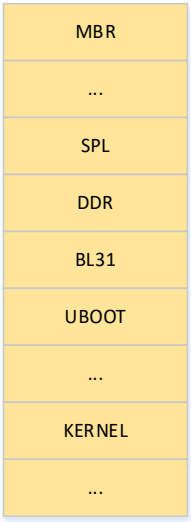


图 3 镜像分区示意图

DDR 分区中存放多套 DDR 参数，SPL 通过外部 GPIO 或 board-id 的方式，从 DDR 分区获取到正确参数。

DDR 的参数主要包含：

- 指令参数: ddr training 用到的指令，ddr training 分为两步，对应也就有第一次 training 用到的指令参数 imem1，和第二次 training 用到的指令 imem2.
- 数据参数：ddr training 用到的数据参数，同样也有两份，dmem1 和 dmem2.
- ddr 控制器参数：配置 ddr 控制器的参数
- ddr phy 参数：配置 ddr phy 用到的参数
- ddr phy init engine 参数：配置 ddr phy init engine 用到的参数
- 地址映射参数：配置 soc 与 ddr 颗粒之间走线的参数

DDR 分区的布局如图 4 所示，开头的 DDR 分区的 header，随后便是多套 ddr 参数。

图 4 DDR 分区布局

DDR header 的格式如下

```
struct hb_ddt_mem {
        unsigned int addr;
```

```
        unsigned int size;
};

/*ddr data*/
struct hb_ddr_data {
        unsigned int ddr_type;
        unsigned int ddr_vendor;
        unsigned int ddr_freq;
        unsigned int part_number;

        struct hb_ddt_mem imem1;        /* ddr imem */
        struct hb_ddt_mem imem2;
        struct hb_ddt_mem dmem1;        /* ddr dmem */
        struct hb_ddt_mem dmem2;

        struct hb_ddt_mem ddr_ddrc;
        struct hb_ddt_mem ddr_ddrp;
        struct hb_ddt_mem ddr_pie;

        struct hb_ddt_mem ddr_ddrc_freqs;
        struct hb_ddt_mem ddr_ddrp_freqs;
        struct hb_ddt_mem ddr_pie_freqs;
        struct hb_ddt_mem addr_map;
};

/*ddr header*/
struct hb_ddr_hdr {
        unsigned int magic;  /* HBOT */
        unsigned int count;
        unsigned short ecc_gran[4];
        unsigned short ecc_map[4];
        unsigned int ddr_para_addr;
        unsigned int ddr_para_size;

        struct hb_ddr_data ddr[20];
        char ddr_pin[3];
        unsigned int sscg_parm[3];
        unsigned int eye_tool_pin;
        ...
```

```
};
```

对上述参数进行介绍：

hb_ddr_hdr 是 ddr header 的结构

- magic: 魔数，初始化时将设置为“HBOT”
- count: ddr 分区中总的参数数目
- ecc_gran 和 ecc_map 是 ecc 相关的
- ddr_para_addr: ddr 参数的起始地址
- ddr_para_size: ddr 参数总的大小
- ddr[20]: 具体每套参数的信息，目前最大支持 20 套参数
- ddr_pin[3]: 搜寻 ddr 所用的 pin，详细解释见 4.6 章节
- **sscg_parm[3]:** 配置 DDR 展频功能,详细解释见 4.8 章节
- eye_tool_pin: 设置眼图工具的 GPIO pin，详细解释见 4.7 章节

hb_ddr_data 是对应每一套参数的结构

- ddr_type: ddr 类型，如 ddr4/lpddr4/lpddr4x 等
- ddr_vendor: 厂商，如海力士，镁光，三星
- ddr_freq: ddr 频率
- part_num:对应具体的 ddr 颗粒
- imem1:imem1 参数的地址和大小
- dmem1:dmem1 参数的地址和大小
- imem2:imem2 参数的地址和大小
- dmem2:dmem2 参数的地址和大小
- ddr_ddrc: ddr controller 参数的地址和大小
- ddr_ddrp: ddr phy 参数的地址和大小
- ddr_pie: ddr phy init engine 参数的地址和大小
- ddr_ddrc_freqs: 变频情况下其他频率 ddr controller 的地址和大小
- ddr_ddrp_freqs: 变频情况下其他频率 ddr phy 的地址和大小
- ddr_pie_freqs: 变频情况下其他频率 ddr pie 的地址和大小
- addr_map: SoC 与 DDR 颗粒之间的地址映射情况

hb_ddt_mem 是每一套参数中具体参数的结构

- addr：地址信息
- size: 大小

每一套参数布局如图 5 下，其中变频相关的参数为可选项，其他为必选项。

图 5 DDR 参数布局

4 添加 DDR 参数

DDR 参数的源码位于 build/ddr 下，其中：

- hb_imem_parameter.c 是 imem 参数文件，包括 imem1 和 imem2 参数
- hb_dmem_parameter.c 是 dmem 参数文件，包括 dmem1 和 dmem2 参数
- hb_ddrc_parameter.c 是 DDR controller 的参数文件，包括 DDRc 参数和 ddrc_freqs 参数
- hb_ddrp_parameter.c 是 DDR phy 的参数文件，包括 ddrp 参数和 ddrp_freqs 参数
- hb_pie_parameter.c 是 DDR pie 的参数文件，包括 pie 参数和 pie_freqs 参数
- hb_addrmap_parameter.c 是 address map 的参数文件

其中指令参数是通用的，只需要区分 DDR 的类型是 DDR4 或是 LPDDR4。所以为节省空间，指令参数只用包含两份即可，不需要每份 DDR 参数都包含。按这种方式，DDR 分区和 DDR 参数的布局如图 6 所示：

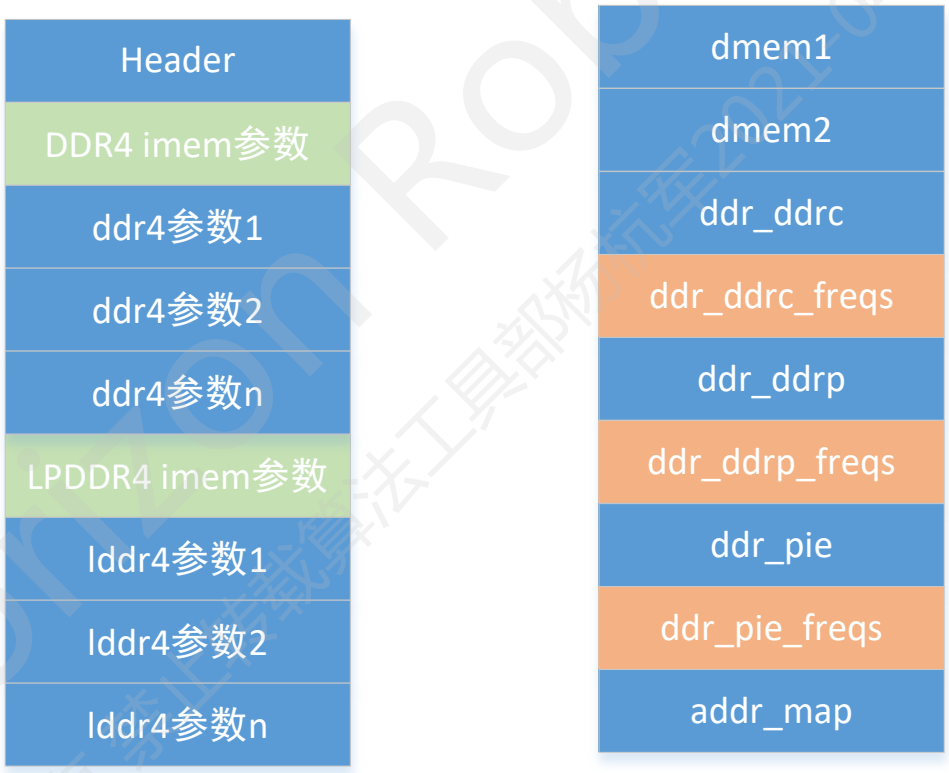
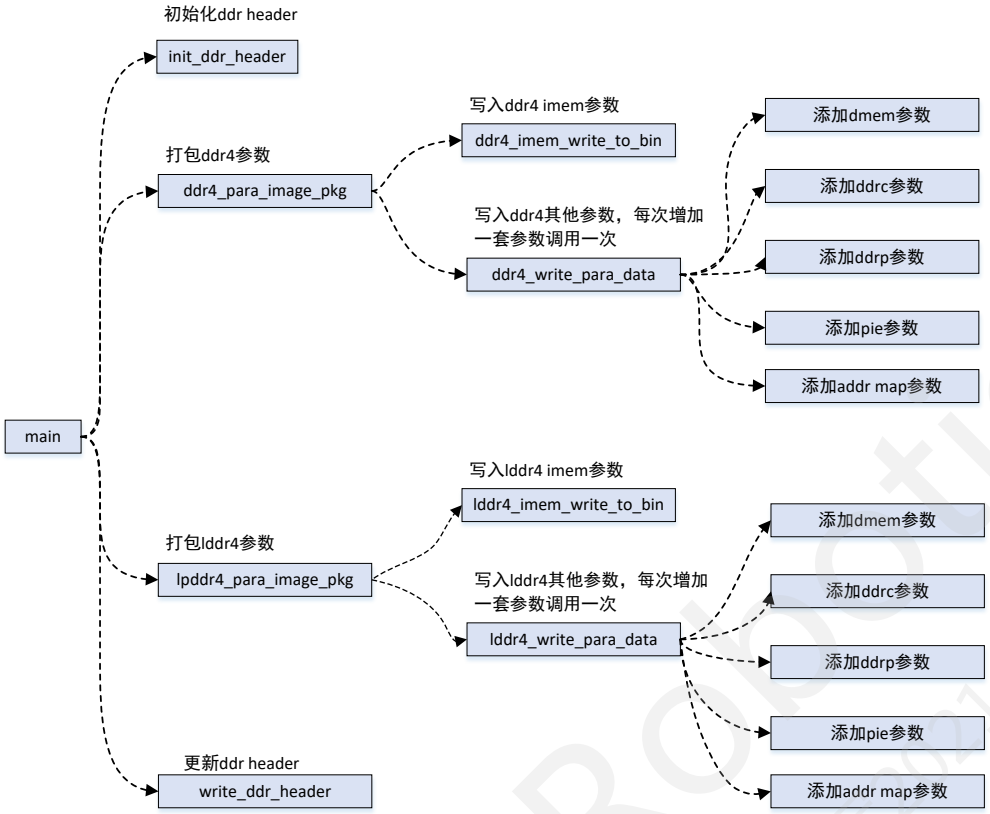


图 6 DDR 分区和 DDR 参数的布局

添加参数的流程如下图：



由于指令参数是一致的，所以对于 DDR4/LPDDR4 的参数添加一次即可，而其他参数则要根据具体情况对参数做出修改。都在/ddr4_write_para_data/lpddr4_write_para_data 中体现，我们以海力士为例：

```
static void lpddr4_write_para_data(char *file, unsigned int freq,
  unsigned int index, unsigned int part_number, unsigned int alter_para)
{
  /* ①init head */
  hdr_ddr.count = hdr_ddr.count + 1;
  hdr_ddr.ddr[index].ddr_type = DDR_TYPE_LPDDR4;
  hdr_ddr.ddr[index].ddr_freq = freq;
  hdr_ddr.ddr[index].part_number = part_number;
  ...

  if ((part_number == MT53D1024M32D4DT) || (part_number == PART_DEFAULT))
{
    /* lpddr4 micron dmem */
  } else if (part_number == H9HCNNN8KUMLHR) {
    /* lpddr4 x3 hynix dmem */
    hdr_ddr.ddr[index].ddr_vendor = DDR_MANU_HYNIX; /*①*/
    lpddr4_hynix_dmem_reconfig_init(freq, part_number,
```

```
    alter_para);/*②*/
        lpddr4_dmem_write_to_bin(file, index);

        /* ddrc, ddrp, pie, mstr */
        lpddr4_ddrc_hynix_write_to_bin(file, freq, index, part_number,
    alter_para);/*③*/
        lpddr4_ddrp_hynix_write_to_bin(file, freq, index, part_number,
    alter_para);/*④*/

        lpddr4_pie_write_to_bin(file, freq, index, part_number);/*⑤*/

        x3_hynix_write_to_bin(file, index);/*⑥*/
      } else if (part_number == H9HCNNNBKUMLHE) {
        /* lpddr4 j3 hynix dmem */
        ...
      } else if (part_number == K4F8E304HBMGCJ || part_number ==
    K4F6E3S4HMMGCJ) {
        /* lpddr4 xg samsung dmem */
        ...
      } else if (part_number == 0) {
        /* lpddr4 x3j3 default hynix paramter */
        ...
      }
    }
```

① 初始化 DDR header 中对应 index 的 DDR data
② 对 dmem 参数做出相应调整后写入镜像
③ 对 DDRc 参数做出相应调整后写入镜像
④ 对 DDRphy 参数做出相应调整后写入镜像
⑤ 对 DDR pie 参数做出相应调整后写入镜像
⑥ 对 addrmap 参数做出相应调整后写入镜像
下面的章节我们分别对以上参数的添加。
注：part-number 是用来区分具体颗粒的参数，高 8bit 表示厂商，低 8bit 代表 DDR 大小。您可以使用自己的方法对 DDR 颗粒作区分，如果您不使用 part-number，将它设置为 0 即可。

4.1 添加 dmem 参数

数据参数根据 DDR 类型（DDR4/LPDDR4）先建立了四个通用的数组，数组位于 *hb_dmem_parameter.c* 文件中。

```
    unsigned short int phyinit_dmem_lpddr4[] = {
```

```
    0x0600, 0x0000, 0x0000, 0x10aa, 0x0002, 0x0000, 0x0014, 0x0000,
    0x131f, 0x00c8, 0x0000, 0x0002, 0x0001, 0x0000, 0x0000, 0x0100,
    0x0000, 0x0000, 0x0310, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x3f74, 0x0033, 0x4d64, 0x4f08, 0x0000, 0x0004, 0x3f74,
    0x0033, 0x4d64, 0x4f08, 0x0000, 0x0004, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x1000, 0x0003, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x7400, 0x333f, 0x6400, 0x084d, 0x004f, 0x0400,
    ...
}

unsigned short int phyinit_dmem_lpddr4_2D[] = {
    0x0600, 0x0000, 0x0000, 0x10aa, 0x0002, 0x0000, 0x0014, 0x0000,
    0x0061, 0x00c8, 0x0000, 0x0002, 0x0001, 0x0000, 0x0000, 0x0100,
    0x8020, 0x0000, 0x0310, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x3f74, 0x0033, 0x4d64, 0x4f08, 0x0000, 0x0004, 0x3f74,
    0x0033, 0x4d64, 0x4f08, 0x0000, 0x0004, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x1000, 0x0003, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x7400, 0x333f, 0x6400, 0x084d, 0x004f, 0x0400,
    0x7400, 0x333f, 0x6400, 0x084d, 0x004f, 0x0400, 0x0000, 0x0000,
    ...
}

unsigned short int phyinit_dmem_ddr4[] = {
    0x0060, 0x0000, 0x0000, 0x0a6a, 0x0002, 0x0000, 0x0260, 0x2000,
    0x0101, 0x0a00, 0x0100, 0x031f, 0x00c8, 0x0100, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0001, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0a44,
    0x0603, 0x0830, 0x0200, 0x1800, 0x0440, 0x0c18, 0x0101, 0x0000,
    ...
}

unsigned short int phyinit_dmem_ddr4_2D[] = {
    0x0060, 0x0000, 0x0000, 0x0a6a, 0x0002, 0x0000, 0x0256, 0x2000,
    0x0101, 0x0a00, 0x0100, 0x0061, 0x00c8, 0x0100, 0x8020, 0x0000,
    0x0000, 0x0000, 0x0001, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
```

```
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0a44,
    0x0603, 0x0830, 0x0200, 0x1800, 0x0440, 0x0c18, 0x0101, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1221,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
...
}
```

根据厂商、频率、颗粒等因素对通用数组做出调整，然后再写入 DDR image 中。我们以海力士为例。

```
void lpddr4_hynix_dmem_reconfig_init(unsigned int freq,
  unsigned int part_number, unsigned int alter_para)
{
  /*①*/
  lpddr4_hynix_dmem_reconfig_silicon(part_number, alter_para, freq);
  /*②*/
  /* 3733 3200 2666 667 */
  if (freq == DDR_FREQC_3733) {
      lpddr4_dmem_reconfig_3733();
      lpddr4_dmem_2d_reconfig_3733();
  } else if (freq == DDR_FREQC_3200) {
      if (part_number == H9HCNNN8KUMLHR) {
          lpddr4_dmem_reconfig_3200_xh();
          lpddr4_dmem_2d_reconfig_3200_xh();
      } else {
          lpddr4_dmem_reconfig_3200();
          lpddr4_dmem_2d_reconfig_3200();
      }
  } else if (freq == DDR_FREQC_2666) {
      if (part_number == H9HCNNN8KUMLHR) {
          lpddr4_dmem_reconfig_2666_xh();
          lpddr4_dmem_2d_reconfig_2666_xh();
      } else {
          lpddr4_dmem_reconfig_2666();
          lpddr4_dmem_2d_reconfig_2666();
      }
  } else if (freq == DDR_FREQC_667) {
```

```
            lpddr4_dmem_reconfig_667();
            lpddr4_dmem_2d_reconfig_667();
        } else if (freq == DDR_FREQC_100) {
            lpddr4_dmem_reconfig_667();
            lpddr4_dmem_2d_reconfig_667();
        }
    }
```

① 厂商有关的更改
② 频率相关的更改
在写入 DDR image 之前，调用该函数更改通用数组中相应的值，然后就写入 DDR image 中去

4.2 添加 DDR controller 参数

根据 DDR 类型（DDR4/LPDDR4）建立通用的参数数组，再根据厂商、频率、颗粒等添加不同的参数，代码位于 *hb_ddrc_parameter.c*

```
    struct DRAM_CFG_PARAM ddr4_ddrc_cfg[] = {
        /* Start to config, default 3200mbps */
        {uMCTL2_MSTR, 0x83040010},
        {uMCTL2_MRCTRL0, 0x40004030},
        {uMCTL2_MRCTRL1, 0x2d11d},
        {uMCTL2_MRCTRL2, 0xc21162f7},
        {uMCTL2_DERATEEN, 0x1404},
        {uMCTL2_DERATEINT, 0x109a928d},
        {uMCTL2_MSTR2, 0x1},
        {uMCTL2_DERATECTL, 0x1},
        {uMCTL2_PWRCTL, 0x0},
        {uMCTL2_PWRTMG, 0x40fa04},
        {uMCTL2_HWLPCTL, 0x730002},
        {uMCTL2_RFSHCTL0, 0x210000},
        {uMCTL2_RFSHCTL1, 0x920016},
        ...
    }
    struct DRAM_CFG_PARAM lpddr4_ddrc_cfg[] = {
    #if (CVB_M_ENABLE == 1)
        {uMCTL2_MSTR, 0x83080020},
        {uMCTL2_RFSHTMG, 0x82080096},
        {uMCTL2_DRAMTMG0, 0x2121242D},
        {uMCTL2_DRAMTMG1, 0x00090941},
```

```
    {uMCTL2_DRAMTMG2, 0x09121519},
    {uMCTL2_DRAMTMG3, 0x00F0F000},
    {uMCTL2_DRAMTMG4, 0x14040914},
    {uMCTL2_DRAMTMG5, 0x02061111},
    {uMCTL2_DRAMTMG6, 0x0000000A},
    {uMCTL2_DRAMTMG7, 0x00000602},
    {uMCTL2_DRAMTMG8, 0x00000000},
    {uMCTL2_DRAMTMG9, 0x00000000},
    {uMCTL2_DRAMTMG10, 0x00000000},
    ...
}
```

上述数组是通用配置数组，然后根据厂商、频率、颗粒的不同再添加相应的参数

```
void lpddr4_ddrc_hynix_write_to_bin(char *file, unsigned int freq,
  unsigned int index, unsigned int part_number, unsigned int alter_para)
{
  FILE *fp = NULL;
  int ddrc_size = 0, ddrc_hynix_size = 0, ddrc_freq_size = 0,
ddrc_dfs_size = 0;
  int padding_size = 0, i;
  char padding_value = 0;

  fp = fopen(file, "ab+");

  /* ①ddrc_cfg */
  ddrc_size = sizeof(lpddr4_ddrc_cfg);
  fwrite(lpddr4_ddrc_cfg, 2, ddrc_size / 2, fp);

  /* ②ddrc_hynix_cfg */
  ddrc_hynix_size = sizeof(lpddr4_ddrc_cfg_hynix);
  fwrite(lpddr4_ddrc_cfg_hynix, 2, ddrc_hynix_size / 2, fp);

  /* ③ddrc_freq */
  if (freq == DDR_FREQC_3733) {
     ...
  } else if (freq == DDR_FREQC_3200) {
     ddrc_freq_size = sizeof(lpddr4_3200_ddrc_cfg_hynix);
     fwrite(lpddr4_3200_ddrc_cfg_hynix, 2,
        sizeof(lpddr4_3200_ddrc_cfg_hynix) / 2, fp);
```

```
        /*④ */
        if (part_number == H9HCNNNBKUMLHE) {
            /* JH32_A1 */
            ddrc_freq_size += sizeof(lpddr4_3200_ddrc_cfg_jh32_a1);
            fwrite(lpddr4_3200_ddrc_cfg_jh32_a1, 2,
                sizeof(lpddr4_3200_ddrc_cfg_jh32_a1) / 2, fp);
        } else {
            ddrc_freq_size += sizeof(lpddr4_3200_ddrc_cfg);
            fwrite(lpddr4_3200_ddrc_cfg, 2,
                sizeof(lpddr4_3200_ddrc_cfg) / 2, fp);
        }
    } else if (freq == DDR_FREQC_2666) {
        ...
    } else if (freq == DDR_FREQC_667) {
        ...
    }


    /*⑤*/
    ddrc_size = ddrc_size + ddrc_hynix_size + ddrc_freq_size;
    if (ddrc_size % 512)
        padding_size = 512 - ((ddrc_size % 512));

    for (i = 0; i < padding_size; i++) {
        fwrite(&padding_value, 1, 1, fp);
    }
   /*⑥*/
    hdr_ddr.ddr[index].ddr_ddrc.addr = hdr_ddr.ddr[index].dmem2.addr +
        ALIGN_512(hdr_ddr.ddr[index].dmem2.size);
    hdr_ddr.ddr[index].ddr_ddrc.size = ddrc_size;
    /*dfs ddr_ddrc_freqs*/
    hdr_ddr.ddr[index].ddr_ddrc_freqs.addr =
hdr_ddr.ddr[index].ddr_ddrc.addr +
        ALIGN_512(hdr_ddr.ddr[index].ddr_ddrc.size);
    hdr_ddr.ddr[index].ddr_ddrc_freqs.size = ddrc_dfs_size;
```

```
    pclose(fp);
    return;
}
```

① 将通用参数写入文件中

② 根据厂商不同，写入厂商相关的参数

③ 根据频率不同，添加频率相关的参数

④ 相同频率下，不同的颗粒，参数也有不同

⑤ 每种参数都 512 字节对齐，向文件中填充 padding

⑥ 更新 DDR header。

4.3 添加 DDR phy 参数

根据 DDR 类型（DDR4/LPDDR4）建立通用的参数数组，再根据厂商、频率、颗粒等添加不同的参数，代码位于 *hb_ddrp_parameter.c*

```
struct DRAM_CFG_PARAM lpddr4_ddrphy_cfg[] = {
    {DWC_DDRPHYA_DBYTE0__TxSlewRate_b0_p0, TxSlewRate_LPDDR4_MICRON},
    {DWC_DDRPHYA_DBYTE0__TxSlewRate_b1_p0, TxSlewRate_LPDDR4_MICRON},
    {DWC_DDRPHYA_DBYTE1__TxSlewRate_b0_p0, TxSlewRate_LPDDR4_MICRON},
    {DWC_DDRPHYA_DBYTE1__TxSlewRate_b1_p0, TxSlewRate_LPDDR4_MICRON},
    {DWC_DDRPHYA_DBYTE2__TxSlewRate_b0_p0, TxSlewRate_LPDDR4_MICRON},
    {DWC_DDRPHYA_DBYTE2__TxSlewRate_b1_p0, TxSlewRate_LPDDR4_MICRON},
    {DWC_DDRPHYA_DBYTE3__TxSlewRate_b0_p0, TxSlewRate_LPDDR4_MICRON},
    {DWC_DDRPHYA_DBYTE3__TxSlewRate_b1_p0, TxSlewRate_LPDDR4_MICRON},
    ...
}
struct DRAM_CFG_PARAM ddr4_ddrphy_cfg[] = {
    {DWC_DDRPHYA_DBYTE0__TxSlewRate_b0_p0, TxSlewRate_DDR4_MICRON},
    {DWC_DDRPHYA_DBYTE0__TxSlewRate_b1_p0, TxSlewRate_DDR4_MICRON},
    {DWC_DDRPHYA_DBYTE1__TxSlewRate_b0_p0, TxSlewRate_DDR4_MICRON},
    {DWC_DDRPHYA_DBYTE1__TxSlewRate_b1_p0, TxSlewRate_DDR4_MICRON},
    {DWC_DDRPHYA_DBYTE2__TxSlewRate_b0_p0, TxSlewRate_DDR4_MICRON},
    {DWC_DDRPHYA_DBYTE2__TxSlewRate_b1_p0, TxSlewRate_DDR4_MICRON},
    {DWC_DDRPHYA_DBYTE3__TxSlewRate_b0_p0, TxSlewRate_DDR4_MICRON},
    {DWC_DDRPHYA_DBYTE3__TxSlewRate_b1_p0, TxSlewRate_DDR4_MICRON},
    ...
}
```

上述数组是通用配置数组，然后根据厂商、频率、颗粒的不同再添加相应的参数数据

```
void lpddr4_ddrp_hynix_write_to_bin(char *file, unsigned int freq,
  unsigned int index, unsigned int part_number, unsigned int alter_para)
{
  FILE *fp = NULL;
  int ddrp_size = 0, ddrp_hynix_size = 0, ddrp_freq_size = 0,
ddrp_dfs_size = 0;
  int padding_size = 0, i;
  char padding_value = 0;

  fp = fopen(file, "ab+");

  /*① ddrp_cfg */
  ddrp_size = sizeof(lpddr4_ddrphy_cfg);
  fwrite(lpddr4_ddrphy_cfg, 2, ddrp_size / 2, fp);

  /* ②ddrp_hynix_cfg */
  if (part_number == H9HCNNN8KUMLHR) {
      /*③*/
      ddrp_hynix_size = sizeof(lpddr4_ddrphy_hynix_cfg_xh);
      fwrite(lpddr4_ddrphy_hynix_cfg_xh, 2, ddrp_hynix_size / 2, fp);
  } else if (part_number == H9HCNNNBKUMLHE) {
      if (freq == DDR_FREQC_3200) {
          ddrp_hynix_size = sizeof(lpddr4_ddrphy_hynix_cfg_jh32_a1);
          fwrite(lpddr4_ddrphy_hynix_cfg_jh32_a1, 2, ddrp_hynix_size / 2,
fp);
      } else {
          ddrp_hynix_size = sizeof(lpddr4_ddrphy_hynix_cfg_jh);
          fwrite(lpddr4_ddrphy_hynix_cfg_jh, 2, ddrp_hynix_size / 2, fp);
      }
  } else {
      ddrp_hynix_size = sizeof(lpddr4_ddrphy_hynix_cfg);
      fwrite(lpddr4_ddrphy_hynix_cfg, 2, ddrp_hynix_size / 2, fp);
  }

  /* ④ddrc_freq */
  if (freq == DDR_FREQC_3733) {
      ...
```

```
    } else if (freq == DDR_FREQC_3200) {
        ddrp_freq_size = sizeof(lpddr4_3200_ddrphy_cfg);
        fwrite(lpddr4_3200_ddrphy_cfg, 2, ddrp_freq_size / 2, fp);
    } else if (freq == DDR_FREQC_2666) {
        ...
    } else if (freq == DDR_FREQC_667) {
        ...
    }


    /* ⑤align 512 */
    ddrp_size = ddrp_size + ddrp_hynix_size + ddrp_freq_size;
    if (ddrp_size % 512)
        padding_size = 512 - ((ddrp_size % 512));

    for (i = 0; i < padding_size; i++) {
        fwrite(&padding_value, 1, 1, fp);
    }

    /*⑥*/
    hdr_ddr.ddr[index].ddr_ddrp.addr =
hdr_ddr.ddr[index].ddr_ddrc_freqs.addr +
        ALIGN_512(hdr_ddr.ddr[index].ddr_ddrc_freqs.size);
    hdr_ddr.ddr[index].ddr_ddrp.size = ddrp_size;
    /*dfs ddr_ddrc_freqs*/
    hdr_ddr.ddr[index].ddr_ddrp_freqs.addr =
hdr_ddr.ddr[index].ddr_ddrp.addr +
        ALIGN_512(hdr_ddr.ddr[index].ddr_ddrp.size);
    hdr_ddr.ddr[index].ddr_ddrp_freqs.size = ddrp_dfs_size;

    pclose(fp);
    return;
}
```

① 将通用参数写入文件中

② 根据厂商不同，写入厂商相关的参数

③ 相同的厂商，不同的颗粒，DDRphy 的参数也有不同

④ 根据频率不同，添加频率相关的参数

⑤ 每种参数都 512 字节对齐，向文件中填充 padding

⑥ 更新 DDR header。

4.4 添加 DDR pie 参数

根据 DDR 类型（DDR4/LPDDR4）建立通用的参数数组，再根据频率添加不同的参数，代码位于 *hb_pie_parameter.c*

```
/* DRAM PHY init engine image */
struct DRAM_CFG_PARAM lpddr4_phy_pie[] = {
  {DWC_DDRPHYA_APBONLY0__MicroContMuxSel, 0x0},
  {DWC_DDRPHYA_INITENG0__PreSequenceReg0b0s0, 0x10},
  {DWC_DDRPHYA_INITENG0__PreSequenceReg0b0s1, 0x400},
  {DWC_DDRPHYA_INITENG0__PreSequenceReg0b0s2, 0x10e},
  {DWC_DDRPHYA_INITENG0__PreSequenceReg0b1s0, 0x0},
  {DWC_DDRPHYA_INITENG0__PreSequenceReg0b1s1, 0x0},
  {DWC_DDRPHYA_INITENG0__PreSequenceReg0b1s2, 0x8},
  ...
}

/* DRAM PHY init engine image */
struct DRAM_CFG_PARAM ddr4_phy_pie[] = {
  {DWC_DDRPHYA_APBONLY0__MicroContMuxSel, 0x0},
  {DWC_DDRPHYA_INITENG0__PreSequenceReg0b0s0, 0x10},
  {DWC_DDRPHYA_INITENG0__PreSequenceReg0b0s1, 0x400},
  {DWC_DDRPHYA_INITENG0__PreSequenceReg0b0s2, 0x10e},
  {DWC_DDRPHYA_INITENG0__PreSequenceReg0b1s0, 0x0},
  {DWC_DDRPHYA_INITENG0__PreSequenceReg0b1s1, 0x0},
  {DWC_DDRPHYA_INITENG0__PreSequenceReg0b1s2, 0x8},
  {DWC_DDRPHYA_INITENG0__SequenceReg0b0s0, 0xb},
  ...
}
```

上述数组是通用配置数组，pie 的数据只和频率相关，所以只用针对频率做出相应的修改即可。

```
void lpddr4_pie_write_to_bin(char *file, unsigned int freq,
  unsigned int index, unsigned int part_number)
```

```
{
    FILE *fp = NULL;
    int pie_size = 0, pie_freq_size = 0, ddr_pie_dfs_size = 0;
    int padding_size = 0, i;
    char padding_value = 0;

    fp = fopen(file, "ab+");

    /* ①ddr pie_cfg */
    pie_size = sizeof(lpddr4_phy_pie);
    fwrite(lpddr4_phy_pie, 2, pie_size / 2, fp);

    /* ②ddrpie_freq */
    if (freq == DDR_FREQC_3733) {
        pie_freq_size = sizeof(lpddr4_3733_phy_pie);
        fwrite(lpddr4_3733_phy_pie, 2, pie_freq_size / 2, fp);
    } else if (freq == DDR_FREQC_3600) {
        pie_freq_size = sizeof(lpddr4_3600_phy_pie);
        fwrite(lpddr4_3600_phy_pie, 2, pie_freq_size / 2, fp);
    } else if (freq == DDR_FREQC_3200) {
        pie_freq_size = sizeof(lpddr4_3200_phy_pie);
        fwrite(lpddr4_3200_phy_pie, 2, pie_freq_size / 2, fp);
    } else if (freq == DDR_FREQC_2666) {
        pie_freq_size = sizeof(lpddr4_2666_phy_pie);
        fwrite(lpddr4_2666_phy_pie, 2, pie_freq_size / 2, fp);
    } else if (freq == DDR_FREQC_667) {
        pie_freq_size = sizeof(lpddr4_667_phy_pie);
        fwrite(lpddr4_667_phy_pie, 2, pie_freq_size / 2, fp);
    } else if (freq == DDR_FREQC_100) {
        pie_freq_size = sizeof(lpddr4_100_phy_pie);
        fwrite(lpddr4_100_phy_pie, 2, pie_freq_size / 2, fp);
    }

    /*③ align 512 */
    pie_size = pie_size + pie_freq_size;
    if (pie_size % 512)
        padding_size = 512 - ((pie_size % 512));

    for (i = 0; i < padding_size; i++) {
```

```
        fwrite(&padding_value, 1, 1, fp);
    }

     /*④*/
    hdr_ddr.ddr[index].ddr_pie.addr =
hdr_ddr.ddr[index].ddr_ddrp_freqs.addr +
        ALIGN_512(hdr_ddr.ddr[index].ddr_ddrp_freqs.size);
    hdr_ddr.ddr[index].ddr_pie.size = pie_size;
    /*dfs ddr_ddrc_freqs*/
    hdr_ddr.ddr[index].ddr_pie_freqs.addr = hdr_ddr.ddr[index].ddr_pie.addr
+
        ALIGN_512(hdr_ddr.ddr[index].ddr_pie.size);
    hdr_ddr.ddr[index].ddr_pie_freqs.size = ddr_pie_dfs_size;

    pclose(fp);
    return;
}
```

① 将通用参数写入文件中

② 根据频率的不同添加不同的参数

③ 每种参数都 512 字节对齐，向文件中填充 padding

④ 更新 ddr header

4.5 添加 address map 参数

address map 参数是配置 SoC 与 DDR 颗粒之间走线的参数，只有 LPDDR4 需要对该参数进行设置。代码中提供了三套参数，分别对应 CVB 板，X3SOM 板和 J3SOM 板。

```
/* #define H9HCNNN8KUMLHR  0x0608 */
struct DRAM_CFG_PARAM lpddr4_mstr_x3_hynix[12] = {
    {uMCTL2_ADDRMAP0, 0x001F1F1F},
    {uMCTL2_ADDRMAP1, 0x00080808},
    {uMCTL2_ADDRMAP2, 0x00000000},
    {uMCTL2_ADDRMAP3, 0x00000000},
    {uMCTL2_ADDRMAP4, 0x00001F1F},
    {uMCTL2_ADDRMAP5, 0x070F0707},
    {uMCTL2_ADDRMAP6, 0x0F070707},
    {uMCTL2_ADDRMAP7, 0x00000F0F},
    {uMCTL2_ADDRMAP8, 0x00003F3F},
    {uMCTL2_ADDRMAP9, 0x07070707},
```

```
    {uMCTL2_ADDRMAP10, 0x07070707},
    {uMCTL2_ADDRMAP11, 0x001F1F07}
};

/* #define H9HCNNNBKUMLHE  0x0610 */
struct DRAM_CFG_PARAM lpddr4_mstr_j3_hynix[12] = {
    {uMCTL2_ADDRMAP0, 0x001F1F1F},
    {uMCTL2_ADDRMAP1, 0x00080808},
    {uMCTL2_ADDRMAP2, 0x00000000},
    {uMCTL2_ADDRMAP3, 0x00000000},
    {uMCTL2_ADDRMAP4, 0x00001F1F},
    {uMCTL2_ADDRMAP5, 0x070F0707},
    {uMCTL2_ADDRMAP6, 0x07070707},
    {uMCTL2_ADDRMAP7, 0x00000F0F},
    {uMCTL2_ADDRMAP8, 0x00003F3F},
    {uMCTL2_ADDRMAP9, 0x07070707},
    {uMCTL2_ADDRMAP10, 0x07070707},
    {uMCTL2_ADDRMAP11, 0x001F1F07}
};

/* #define MT53D1024M32D4DT 0xff10 */
struct DRAM_CFG_PARAM lpddr4_mstr_micron[12] = {
    {uMCTL2_ADDRMAP0, 0x001F1F17},
    {uMCTL2_ADDRMAP1, 0x00080808},
    {uMCTL2_ADDRMAP2, 0x00000000},
    {uMCTL2_ADDRMAP3, 0x00000000},
    {uMCTL2_ADDRMAP4, 0x00001F1F},
    {uMCTL2_ADDRMAP5, 0x070F0707},
    {uMCTL2_ADDRMAP6, 0x07070707},
    {uMCTL2_ADDRMAP7, 0x00000F0F},
    {uMCTL2_ADDRMAP8, 0x00003F3F},
    {uMCTL2_ADDRMAP9, 0x07070707},
    {uMCTL2_ADDRMAP10, 0x07070707},
    {uMCTL2_ADDRMAP11, 0x001F1F07}
};
```

若客户的板子参考自上述三种，直接用定义好的参数即可。如果客户自己设计 SoC 与 DDR 颗粒的走线，需按照客户走线方式配置改参数。

```
void j3_hynix_write_to_bin(char *file, unsigned int index)
```

```
{
    FILE *fp = NULL;
    int mstr_size = 0;
    int padding_size = 0, i;
    char padding_value = 0;

    fp = fopen(file, "ab+");

    /* ①mstr */
    mstr_size = sizeof(lpddr4_mstr_j3_hynix);
    printf("ddr4 pie size: %d\n", mstr_size);
    fwrite(lpddr4_mstr_j3_hynix, 2, mstr_size / 2, fp);

    /*② align 512 */
    if (mstr_size % 512)
        padding_size = 512 - (mstr_size % 512);

    for (i = 0; i < padding_size; i++) {
        fwrite(&padding_value, 1, 1, fp);
    }
    /*③*/
    hdr_ddr.ddr[index].addr_map.addr =
hdr_ddr.ddr[index].ddr_pie_freqs.addr +
        ALIGN_512(hdr_ddr.ddr[index].ddr_pie_freqs.size);
    hdr_ddr.ddr[index].addr_map.size = mstr_size;

    pclose(fp);
    return;
}
```

① 将 address map 参数写入文件中

② 每种参数都 512 字节对齐，向文件中填充 padding

③ 更新 DDR header

4.6 获取 DDR 参数

经过上述步骤，添加 DDR 参数完毕。本章节将介绍如何使 SPL 获取到添加的 DDR 参数。

4.6.1 BOARD-ID 指定

SPL 通过厂商+DDR 类型+频率+index 的方式查找到正确的 DDR 参数，这些作为 BOARD-ID 存储在 MBR 中，BOARD-ID 占 32bit，厂商、频率、DDR 类型和 index 各占 4 个 bit。

bit map	[28:32]	[24:27]	[20:23]	[4:7]
description	manufacture	DDR type	frequency	index

BOARD-ID 是编译时被指定的，目前支持的厂商/DDR 类型/频率如下

厂商：

value	1	2	3
manufacture	hynix	micron	samsung

频率：

value	1	2	3	4	5	6
frequency	667	1600	2133	2666	3200	3733
value	7	8	9	10	11	
frequency	4266	1866	2400	100	3600	

DDR 类型

value	1	2	3	4
DDR mode	LPDDR4	LPDDR4X	DDR4	DDR3L

编译时 DDR 相关配置参数

parameter	description
-m	厂商: “hynix” “micron” “samsung”
-t	DDR 类型: “LDDR4” “LPDDR4X” “DDR4” “DDR3L”
-f	频率：根据上面频率表中设定即可
-i	index，*-i* 与 *index value* 中间无空格

假如有两套参数，厂商、DDR 类型、频率都是海力士、LPDDR4、3200。这两套参数的不同可能是 DDR 大小不同，或是 rank 值不同，这种情况就用 index 区分。假设您要用第二套海力士 LPDDR4,频率为 3200 的参数，编译方法如下

```
build.sh -m hynix -t LPDDR4 -f 3200 -i2
```

如果厂商/DDR 类型/频率不在上述表格中，您需要手动添加。需要修改 *build/build.sh* 脚本，以添加一个厂商为例，假设你使用的 DDR 厂商的名称为“*customer*”

- 在厂商列表中添加新的厂商名

```
ddr_mfg_name_arr=("hynix" "micron" "samsung" "ddrphy_phyinit" "customer")
```

添加厂商对应的 value 值，注意要与 DDR header 中的厂商对应匹配。

```
case $ddr_manufacture in
hynix)
    index=1
    ;;
micron)
    index=2
    ;;
samsung)
    index=3
    ;;
customer)
    index=4
    ;;
*)
    index=0
```

4.6.2 外部 PIN 指定

如果您想在一套产品上使用不同的 DDR 颗粒，并且使用同一套软件镜像。可以通过外部 GPIO 的方式指定 DDR 参数。在 DDR header 中，数组 *ddr_pin[3]*用于指定使用哪几个 GPIO 来决定 DDR 参数的 index。*ddr_pin[0]*，*ddr_pin[1]*，*ddr_pin[2]*分别对应 bit0，bit1，bit2，最大支持 8 套参数。您可以使用 *ddr_pin[0]*，*ddr_pin[0]+ddr_pin[1]*或者三个 pin 都使用，取决于您需要区分几套 DDR 参数。将 *ddr_pin[x]*置 0 表示不使用这个 pin。

编译阶段，不需要设置 DDR 相关的参数，SPL 根据外部 PIN 得到 *index* 选择 DDR 参数。

外部 PIN 方式的优先级更高，只有当所有 *ddr_pin[x]*被设置为 0 的时候，SPL 才会通过 BOARD-ID 的方式搜寻 DDR 参数。

4.7 眼图工具

眼图是用于从经过训练的 DDR 中获取诊断信息的软件工具。提供了两种方法来启用眼图工具。

- 外部 PIN: 在 DDR header 中 *eye_tool_pin* 元素用于设置将哪个外部 PIN 作为眼图工具的调试 PIN。这个 PIN 为高电平，SPL 中将会使能眼图工具。
- 重启命令: 在 kernel 中执行"reboot eye"命令，重启后 SPL 将使能眼图工具，这是一个一次性的命令，再次重启后将从正常流程启动。

4.8 展频

在 DDR header 中的 *sscg_parm[]*数组用于配置展频功能

	description
sscg_parm[0]	展频级别: SSCG_DISABLE: 不开启展频 SSCG_LVL1: 1% SSCG_LVL2: 2% SSCG_DBG: 使用 *sscg_parm[1]* 和 *sscg_param[2]* 配置展频
sscg_parm[1]	配置展频范围 展频区间在当前频率与（当前频率- X%）之间 只有当 *sscg_param[0]* 设置为 SSCG_DBG, *sscg_parm[1]* 才有效
sscg_parm[2]	配置产品值 展频区间在当前频率与（当前频率- Y）之间 只有 *sscg_param[0]* 设置为 SSCG_DBG, *sscg_parm[2]* 才有效