地 平 线
Horizon Robotics

X3J3 DDR Programming Guide

V1.0

2021-02

Important Notice and Disclaimer

Information in this document is provided solely to enable system and software implementers to use Horizon products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

All statements, information and recommendations in this document are provided "AS IS". Horizon makes no warranty, representation or guarantee of any kind, express or implied, regarding the merchantability, fitness or suitability of its products for any particular purpose, and non-infringement of any third party intellectual property rights, nor does Horizon assume any liability arising out of the application or use of any product, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

"Typical" parameters that may be provided in Horizon datasheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Buyers and others who are developing systems that incorporate Horizon products (collectively, "Users") understand and agree that Users shall remain responsible for using independent analysis, evaluation and judgment in designing their applications and that Users have full and exclusive responsibility to assure the safety of Users' applications and compliance of their applications (and of all Horizon products used in or for Users' applications) with all applicable regulations, laws and other applicable requirements.

User agrees to fully indemnify Horizon and its representatives against any claims, damages, costs, losses and/or liabilities arising out of User's unauthorized application of Horizon products and non-compliance with any terms of this notice.

Horizon Robotics, Inc.

https://www.horizon.ai

Revision History

Date	Revision	Descriptions
2020.12	V0.1	Document Created
2021.01	V0.2	update purpose and introduction of DDR partition
2021.02	V1.0	add 3.7 diagnostics tools, 3.8 spread spectrum and 3.6.1 board-id specify; Release Document

Contents

1 Introduction

1.1 Purpose

This DDR programming guide provides the detail programming procedures for DDR tuning, the overview of system boot and DDR code architecture. By programming the DDR code, the customers are allowed to modify and add multiple sets of DDR parameters in order to support various DDR components. For detail definitions of DDR parameters, please refer to the documentation "*X3J3 DDR Tuning Guide*".

1.2 Terminology

Abbr	Description
MBR	Master Boot Record
SoC	System on Chip
BL[x]	Boot Loader Stage [x]
SPL	Secondary Program Loader
DDR	Double Data Rate
LPDDR	Low Power Double Data Rate
SRAM	Static Random Access Memory

2 Device Initialization

2.1 System Boot Overview

The system boot consists of two types: cold boot and warm boot. The cold boot process begins at the Power-On Reset (POR) where the hardware reset logic forces the X3J3 chip to begin the execution starting from the on-chip boot ROM. ROM Code is a software that resides in the on-chip read-only memory (ROM). During the cold boot process, the boot image can be loaded into device memory and executed. Below Figure 1 details the flow of cold boot process.

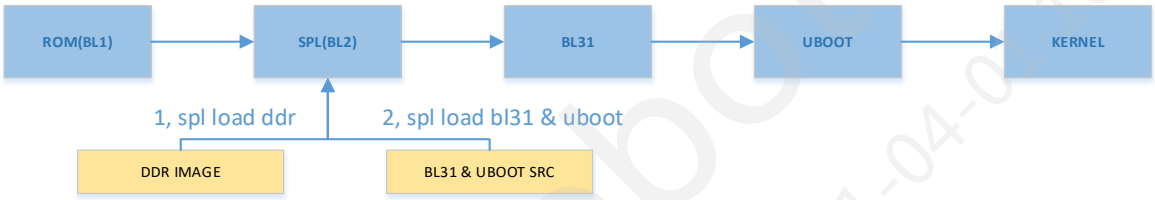


Figure 1: cold boot process

Once the ROM code has been executed, the x3J3 then starts the executions of SPL, BL31 and UBOOT in sequence. Both DDR image and BL31/UBOOT image would be loaded by SPL. The DDR image is used for setting DDR parameters. Once UBOOT has been executed, it completes the loading of the kernel, and finally jumps to the kernel to run.

Warm boot is also known as the suspend/resume process, below Figure 2 details the flow of warm boot process. .

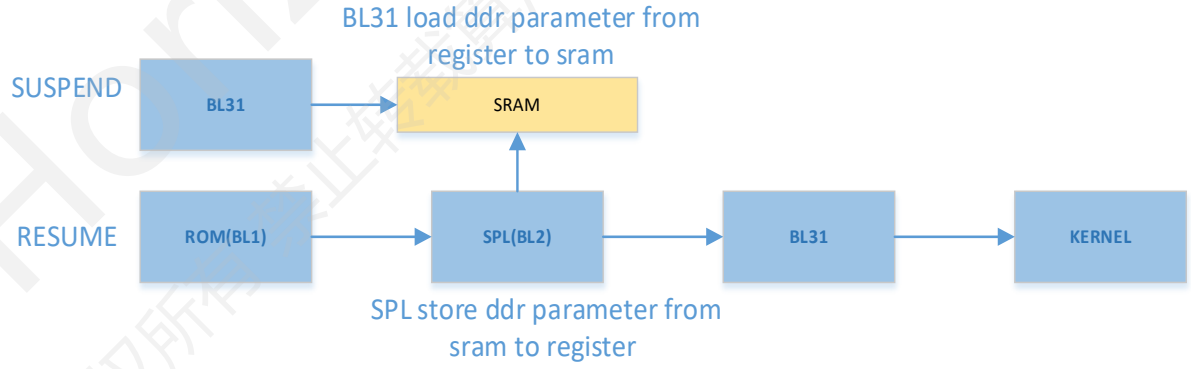


Figure2: warm boot process

Before suspend, the BL31 stores the parameters of the DDR register on SRAM. When waking up, the SPL retrieves the DDR parameters from the SRAM and writing them to the DDR register again.

2.2 DDR Code Architecture

Below Figure3 shows the system image map, while non-DDR-related portions have been skipped intentionally. The location of the DDR image is recorded in the MBR. SPL obtains the location of the DDR image by reading the MBR.

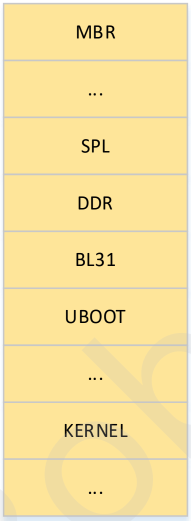


Figure3：Partition layout of overall image

DDR parameters are stored in the DDR partition. The DDR code contains the Header and Multiple sets of DDR parameters. After the SPL detects the DDR model and frequency through external pin or board-id, the corresponding DDR parameters will be obtained from the DDR code.

The DDR parameters have the following components:

- **Instruct parameters**: The instruct parameters, which consists of imem1 and imem2, are used for DDR training. imem1 is associated with the DDR delay fine-tuning, while imem2 is associated with the DDR voltage fine-tuning.
- **Data parameters**: data parameters, which consists of dmem1 and dmem2, are used for DDR training.
- **ddr controller parameters**: configure parameters of the ddr controller.
- **ddr phy parameters**: configure parameters of ddr phy.
- **ddr phy engine parameters**: configure parameter of ddr phy engine
- **Address mapping parameters**: configure parameters for DDR signal addressing between X3J3 and DDR component.

Below Figure 4 shows the DDR partition layout. The DDR code contains the header in the beginning, followed by multiple sets of DDR parameters.

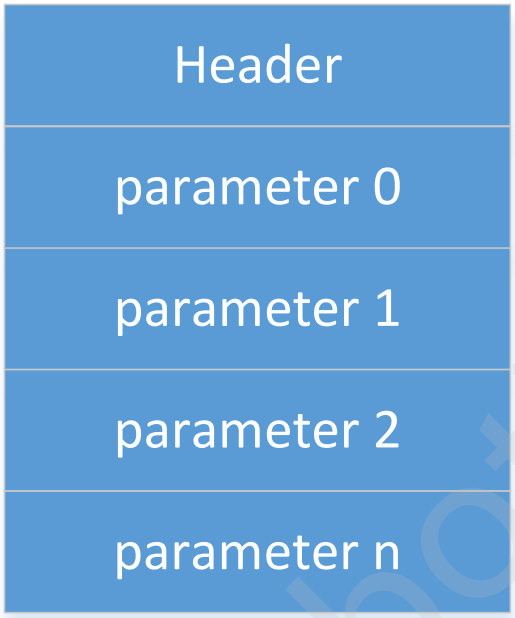


Figure4: DDR partition layoutcode architecture

Below example demonstrates the format of the DDR header:

```
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;
        ...
};
```

hb_ddr_hdr is the structure of the ddr header

- **magic**: magic number, will be set to "HBOT" during initialization
- **count**: the total number of parameters in the ddr partition
- **ddr_para_addr**: the starting address of the ddr parameter
- **ddr_para_size**: total size of ddr parameters
- **ddr[20]**: specific information of each set of parameters, currently supports up to 20 sets of parameters
- **ddr_pin[3]**: Search pin used by ddr, see chapter 3.6 for detailed explanation
- **sscg_parm[3]:** configure DDR spread spectrum feature, see chapter 3.8 for detailed explanation
- **eye_tool_pin**: GPIO pin to set diagnostics tool

hb_ddr_data is the structure corresponding to each set of parameters

- **ddr_type**: ddr type, such as ddr4/lpddr4/lpddr4x, etc.

- **ddr_vendor:** manufacturers, such as Hynix, Micron, Samsung
- **ddr_freq**: ddr frequency
- **part_num**: corresponding to specific ddr
- **imem1**: the address and size of the imem1 parameter
- **dmem1**: the address and size of the dmem1 parameter
- **imem2**: the address and size of the imem2 parameter
- **dmem2**: the address and size of the dmem2 parameter
- **ddr_ddrc:** the address and size of the ddr conctroller parameter
- **ddr_ddrp:** the address and size of the ddr phy parameter
- **ddr_pie**: the address and size of the ddr phy init engine parameter
- **ddr_ddrc_freqs**: address and size of other frequency ddr controller parameter under variable frequency
- **ddr_ddrp_freqs**: address and size of other frequency ddr phy parameter under variable frequency
- **ddr_pie_freqs**: address and size of other frequency ddr phy init engine parameter under variable frequency
- **addr_map**: DQ map between soc and ddr

hb_ddt_mem is the structure of specific parameters in each set of parameters

- **addr**: address information
- **size**: size information

The layout of each set of parameters is shown in Figure 5, where the parameters related to frequency conversion are optional, and the others are mandatory.

Figure 5: DDR parameter layout

3 Steps of adding DDR parameters

The source code of the DDR parameter is located under *build/ddr*, where:

- hb_imem_parameter.c: instruction parameters file, including imem1 and imem2 parameters
- hb_dmem_parameter.c: data parameters file, including dmem1 and dmem2 parameters
- hb_ddrc_parameter.c: DDR controller parameters file, including DDR controller parameters and ddrc_freqs parameters
- hb_ddrp_parameter.c: DDR phy parameters file, including DDR phy parameters and ddrp_freqs parameters
- hb_pie_parameter.c: DDR pie parameters file, including pie parameters and pie_freqs parameters
- hb_addrmap_parameter.c: address map parameters file

There are two sets of instruction parameters, one for DDR4, another for LPDDR4. Therefore, only one copy of each DDR type(DDR4 and LPDDR4) is included in DDR partition. The detailed layout of DDR partitions and DDR parameters is shown in Figure 6:

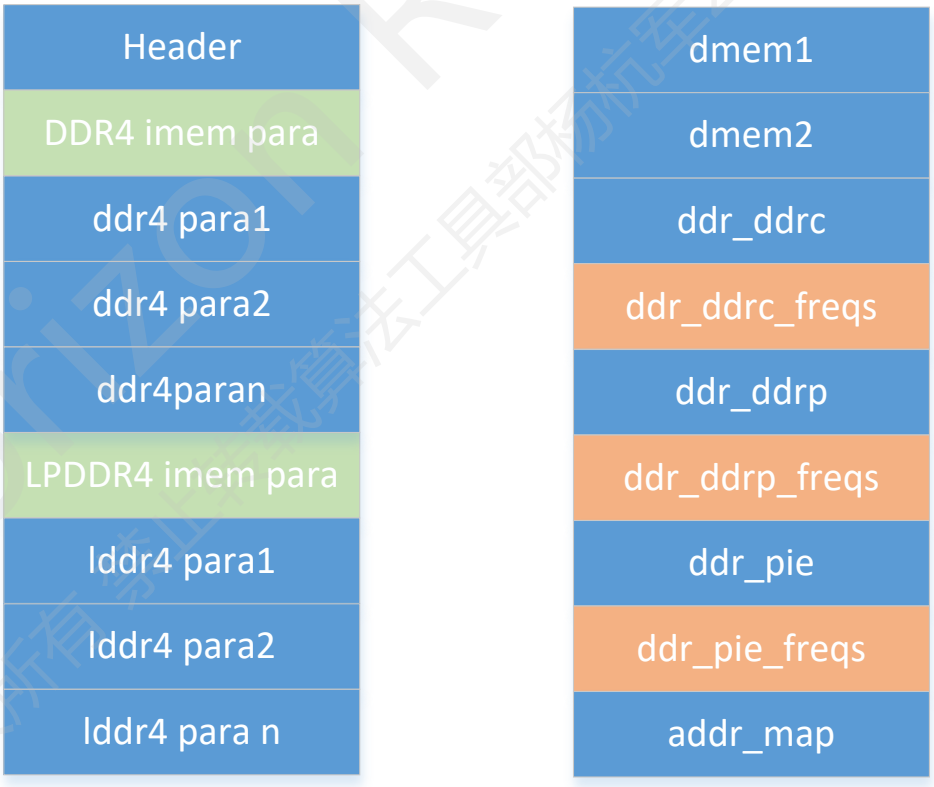
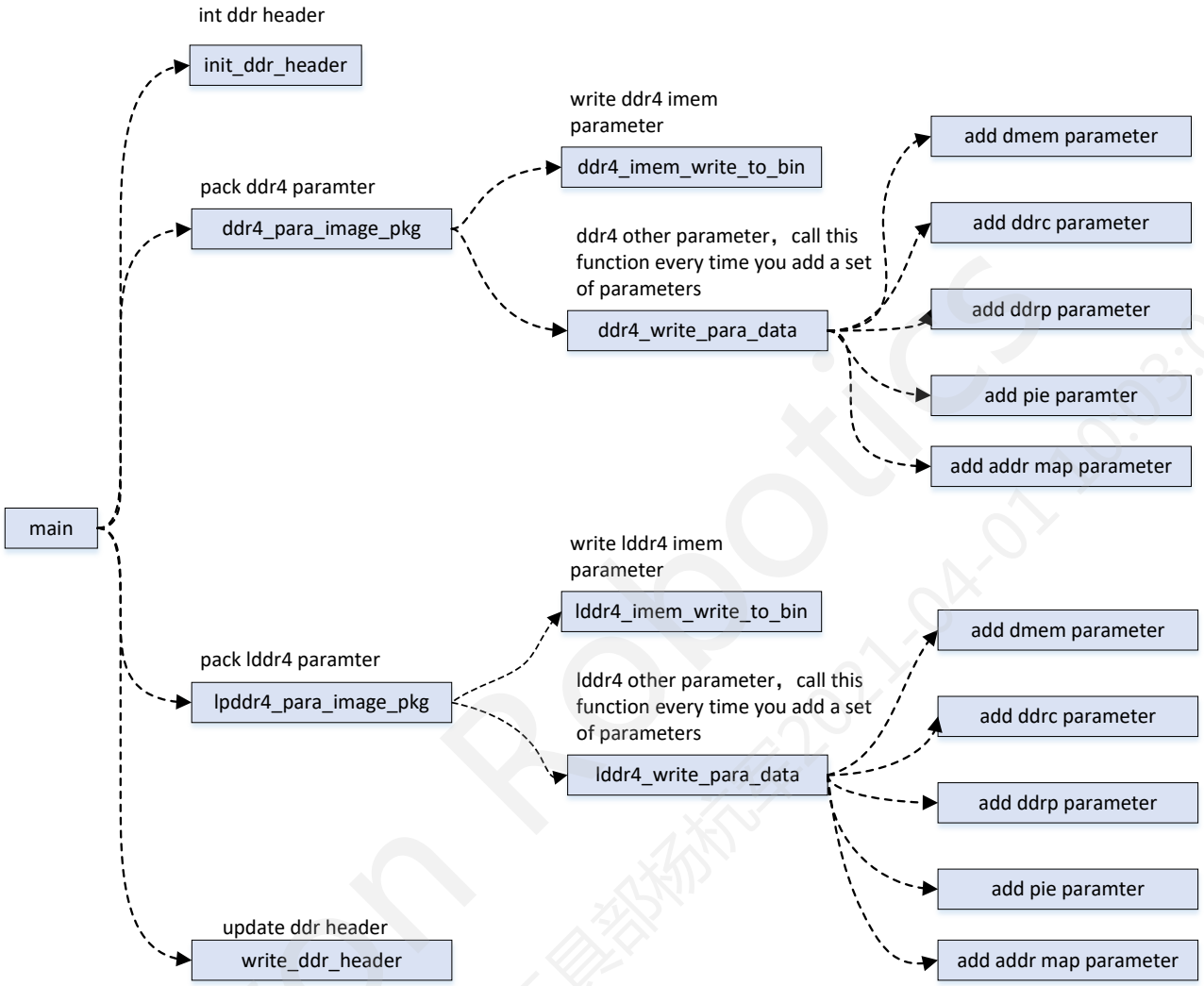


Figure 6: Layout of DDR partition and DDR parameters

The steps of adding parameters is as follows:



Since the instruction parameter is universal, it is sufficient to add the parameters of ddr4/lpddr4 once, while other parameters(DDR data parameters, DDR controller parameters etc.) need to be modified according to the specific scenario. The above steps are implemented in function *ddr4_write_para_data/lpddr4_write_para_data* . The flowing steps use Hynix as an example:

```
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 Initialize the DDR data corresponding to the index in the DDR image header

② Make corresponding adjustments to the dmem parameters and write to the mirror

③ Make corresponding adjustments to the ddrc parameters and write to the mirror

④ Make corresponding adjustments to the ddrphy parameters and write to the mirror

⑤ Make corresponding adjustments to the ddr pie parameters and write to the mirror image

⑥ Make corresponding adjustments to the addrmap parameters and write to the mirror

Note1: Part number is a parameter used to distinguish different DDR chip, where the high byte is the vendor id and the low byte is the ddr size. You can have your own way to distinguish different ddr chips. If you do not use part number, just set it to zero.

In the following chapters we add the above parameters respectively.

Note2: In the following chapters, "baseline" parameters need not be changed. Customized parameters should be added after the "baseline" to overwrite or to supplement the original parameters.

3.1 Add dmem parameters

According to the DDR type (DDR4/LPDDR4) and the training process, four baseline arrays are provided for the data parameters. The arrays are located in the *hb_dmem_parameter.c* file.

```
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,
...
}
```

Before writing to the DDR image, you need to modify parameters according to the manufacturer, frequency, DDR size, etc. The following implementation uses Hynix DDR as an example.

```
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 Vendor-related changes

② Frequency related changes

Before writing to DDR image, call the implemented function to change the corresponding value in the baseline arrays, and then you can write to the DDR image.

3.2 Add DDR controller parameters

Similar to 3.1, a baseline parameter array is provided for each DDR type(DDR4/LPDDR4). Add different parameters according to the manufacturer, frequency, particle, etc. The code is located in *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},
  ...
}
```

Before writing to the DDR image, you need to add additional parameters according to the manufacturer, frequency, DDR size, etc. The following implementation uses Hynix DDR as an example.

```
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 Write baseline parameters into the DDR output image

② According to different manufacturers, write manufacturer-related parameters

③ Add frequency-related parameters according to different frequencies

④ At the same frequency, different DDR chip have different parameters

⑤ Each parameter must be 512 bytes aligned, thus padding is added

⑥ Update the DDR header.

3.3 Add DDR phy parameters

Similar to 3.1, a baseline parameter array is provided for each DDR type(DDR4/LPDDR4). Add different parameters according to the manufacturer, frequency, particle, etc. The code is located in *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},
   ...
}
```

Before writing to the DDR image, you need to add additional parameters according to the manufacturer, frequency, DDR size, etc. The following implementation uses Hynix DDR as an example.

```
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 Write baseline parameters into the file

② According to different manufacturers, write manufacturer-related parameters

③ The DDR phy parameters are different between different chips of the same manufacturer

④ Add frequency-related parameters according to different frequencies

⑤ Each parameter must be 512 bytes aligned, thus padding is added

⑥ Update the DDR header.

3.4 Add DDR pie parameters

Similar to 3.1, a baseline parameter array is provided for each DDR type(DDR4/LPDDR4). Add different parameters according to the manufacturer, frequency, particle, etc. The code is located in *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},
   ...
}
```

Before writing to the DDR image, you need to add additional parameters according to frequency ONLY. The following implementation uses Hynix DDR as an example.

```
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;
}
```

①Write baseline parameters into the file

②Add customized parameters according to the frequency

③Each parameter is 512 bytes aligned, thus padding is added

④Update DDR header

3.5 Add address map parameters

The address map parameter is a parameter for configuring the wiring between the SoC (Horizon core) and DDR chips. There is no need to configure it for DDR4 type. Three sets of parameters are provided in the source code, corresponding to the CVB board, the X3SOM board and the J3SOM board.

```
/* #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}
};
```

If the your board refers to the above three boards, use the defined parameters directly. If

you designed the wiring between the SoC and DDR chip by yourself, the parameters need to be changed according to the your wiring configuration

```
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;
}
```

①Write the address map parameter to the image

②Each parameter is 512 bytes aligned, thus padding is added

③Update DDR header

3.6 Choose DDR parameters

After the above steps, customized DDR parameters will be added to the DDR image. This chapter will introduce you the methods to using the added parameters.

3.6.1 Board-id specify

SPL chooses the DDR parameter in DDR image by manufacture + frequency + DDR type +index, which are stored as board-id in MBR. The board-id is 32 bits in length, and the manufacturer/frequency/DDR type/index is 4 bits each.

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

The board-id is specified during the compilation stage, and the currently supported manufacturers/frequency/DDR types are as follows:

manufacture:

value	1	2	3
manufacture	hynix	micron	samsung

frequency:

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 mode:

value	1	2	3	4
DDR mode	LPDDR4	LPDDR4X	DDR4	DDR3L

DDR-related configuration parameters during compilation

parameter	description
-m	manufacture: “hynix” “micron” “samsung”
-t	DDR type: “LDDR4” “LPDDR4X” “DDR4” “DDR3L”
-f	frequency：set frequency according to the above frequency table

-i	index, no space between *-i* and index value

For example, suppose there are two sets of DDR parameters, the manufacturer, DDR type and frequency are Hynix, LPDDR4, and 3200 respectively. The difference between the two DDR chips may be in sizes or rank values. Use the *index* parameter to distinguish. Suppose you want to use the second set of Hynix LPDDR4 3200 frequency parameter, the compilation command is as follows:

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

If the manufacturer, DDR type or frequency is not in the above list, you need to modify the *build/build.sh* script. Take adding a vendor as an example, suppose the name of the newly added vendor is "customer"

- Add the new vendor “customer” to the vendor list

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

- Configure the index of new vendor, the index should be consistent with the manufacture number in DDR image

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

3.6.2 External pin detect

If you have different DDR chips for one product and want to use the same software

image, you can define different GPIO pin high/low status to choose DDR parameters. SPL utilize 3 external GPIO pins to choose different DDR parameters. The "*ddr_pin[3]*" array in the DDR header is used to specify which pins are used to determine the ddr index, "*ddr_pin[0]*", "*ddr_pin[1]*", and "*ddr_pin[2]*" respectively correspond to bit0, bit1, and bit2, and a maximum of 8 sets of parameters are supported . The init_ddr_header function in hb_imem_parameter.c can be set accordingly. You can only use "*ddr_pin[0]*", or "*ddr_pin[0]+ddr_pin[1]*", or all three GPIO pins, depending on how many DDR types you need to differentiate. Write zero to "*ddr_pin[x]*" to disable the corresponding GPIO pin[x].

During the compilation, there is no need to set DDR related parameters. SPL selects DDR parameters according to the index.

Detecting by external pin has higher priority than board-id, only all "*ddr_pin[x]*" are set to 0, SPL will choose DDR parameter by board-id.

3.7 Diagnostics Tool

The Diagnostics Firmware is a software tool to obtain diagnostic information from a trained DDR system. Two methods are provided to enable diagnostics tool.

- external pin: *eye_tool_pin* element in DDR header is used to configure which external pin is used as the debug pin of the diagnostics tool. If the external pin is set to 1, diagnostics tool will be enabled in SPL.
- reboot command in kernel: the command "reboot eye" in kernel will enable diagnostics tool. This is is a one-time command, restart to return to normal start-up mode

3.8 Spread spectrum

"*sscg_parm[]*" array is used to configure DDR spread spectrum feature. You can set this array to configure spread spectrum

	description
sscg_parm[0]	spread spectrum range level: SSCG_DISABLE: disable spread spectrum SSCG_LVL1: 1% SSCG_LVL2: 2% SSCG_DBG: use *sscg_parm[1]* and *sscg_param[2]* to configure
sscg_parm[1]	spread spectrum range

	The spread frequency range is between the current frequency and the frequency-X% ONLY *sscg_param[0]* is SSCG_DBG, *sscg_parm[1]* is valid
sscg_parm[2]	spread spectrum value The spread frequency range is between the current frequency and the frequency-Y ONLY *sscg_param[0]* is SSCG_DBG, *sscg_parm[2]* is valid