AN0089
应用笔记

AT32 MCU SDRAM Application Note

前言

本文主要讲解 AT32 SDRAM 控制器的使用。

支持型号列表：

支持型号	具备 SDRAM 的型号

目录

表目录

图目录

1 SDRAM 介绍

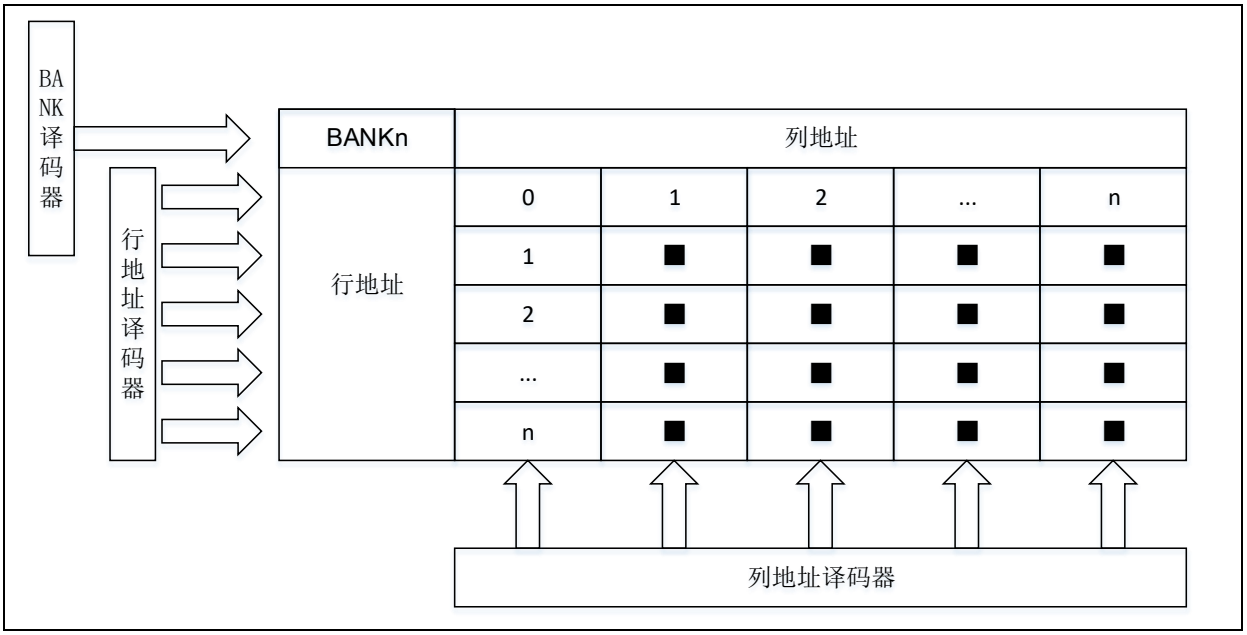
同步动态随机存储器（SDRAM）特点：

- 同步：memory 工作时需要同步时钟
- 动态：存储阵列需要不断刷新
- 随机：自由指定地址读写数据
- 容量大价格便宜

1.1 SDRAM 存储结构

SDRAM 支持多 BANK，通过指定 BANK 号，行地址，列地址找到目标存储单元。

图 1 存储结构



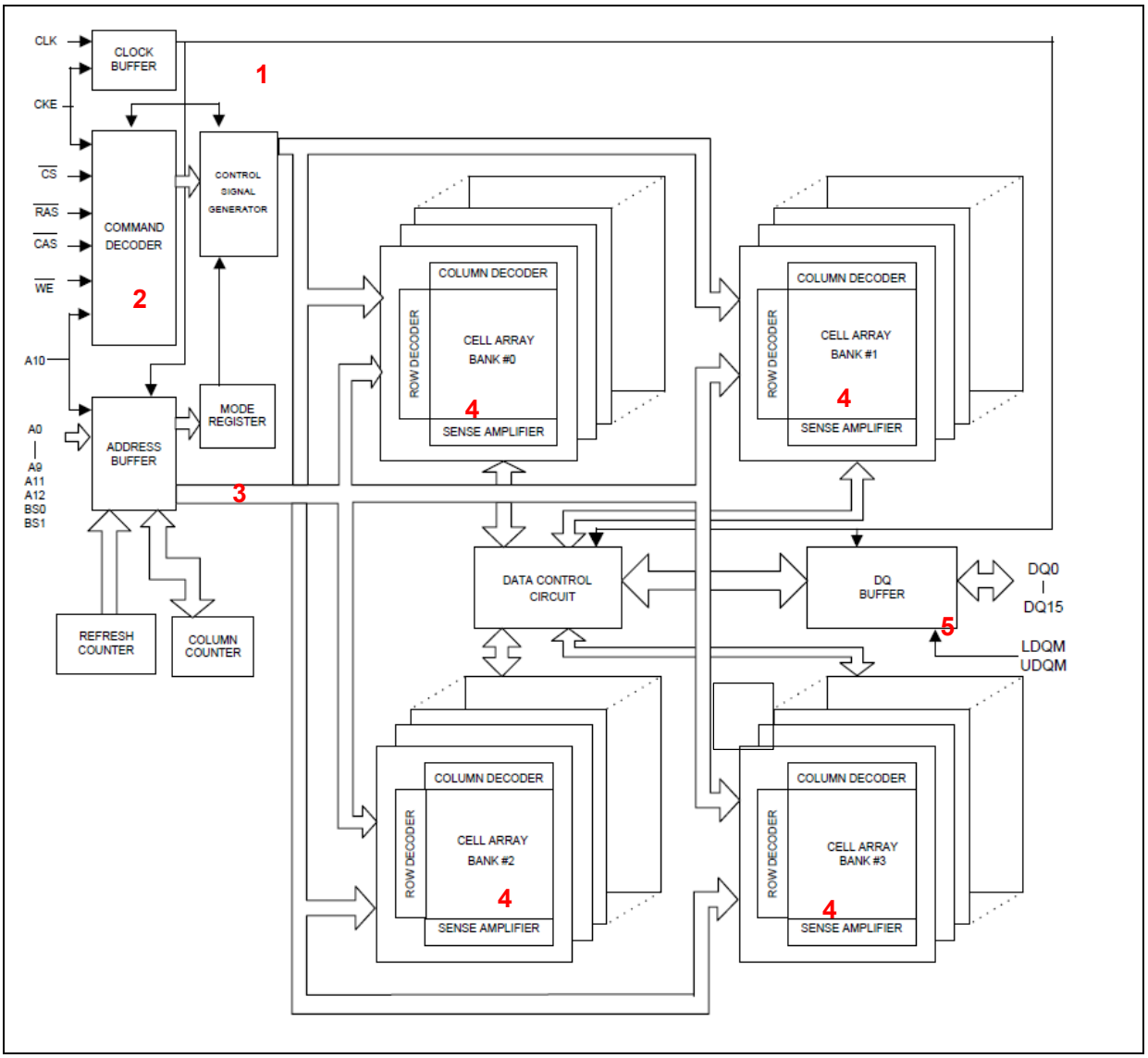
1.2 SDRAM 信号线

信号线	说明
CLK	时钟信号
CKE	时钟信号使能
CS	片选信号
RAS	行地址选通信号
CAS	列地址选通信号
WE	写使能
A0-An	地址线（行列地址复用）
BS0-BS1	BANK 地址线
DQ0-DQn	数据线
DQM	数据掩码(表示数据有效部分)

1.3 SDRAM 内部框图

如下以 W9825G6KH 内部框图举例：

图 2 W9825G6KH 框图



- 1 时钟控制
- 2 命令控制
- 3 地址控制
- 4 存储阵列，4 个 BANK
- 5 数据

1.4 SDRAM 常用命令表

SDRAM 通过信号线上的不同状态来产生各种命令。

表 1 SDRAM 命令表

命令	CS	RAS	CAS	WE	DQM	ADDR	DQ
No-Operation	L	H	H	H	X	X	X
Bank/Row active	L	L	H	H	X	BANK/Row	X
Read	L	H	L	H	L/H	BANK/Col	Data
Write	L	H	L	L	L/H	BANK/Col	Data
Precharge	L	L	H	L	X	A10=H/L	X
Refresh	L	L	L	H	X	X	X
Mode Register	L	L	L	L	X	MODE	X

注意：L=Low Level　H=High Level　X= Don't Care

A10=H 表示 Precharge all bank，A10=L 表示 Precharge 选择的 BANK

- No-Operation

  表示选中当前设备，当前没有操作。

- Bank/Row active

  在对 SDRAM 进行读写时，需要先激活对应的 bank 和行，该命令用于选择一个 bank 的一行进行激活，以便接下来进行读写访问。

- Read

  激活的行有效之后，对列地址进行寻址，读出数据。

- Write

  激活的行有效之后，对列地址进行寻址，写入数据。

- Precharge

  预充电命令，在某一行上的读写完成之后，关闭现有的行，准备激活新行。

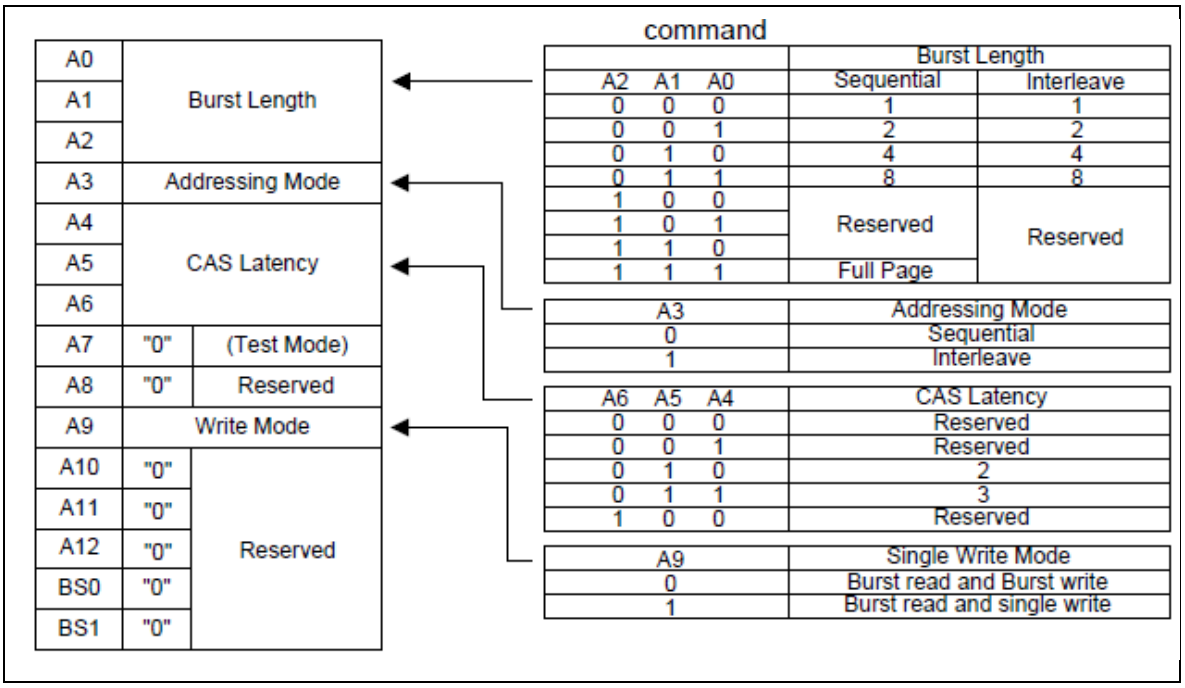
- Refresh

  刷新命令，SDRAM 需要不断的刷新操作才能保存数据，根据 SDRAM 设备参数按照固定周期进行刷新。

- Load Mode Register

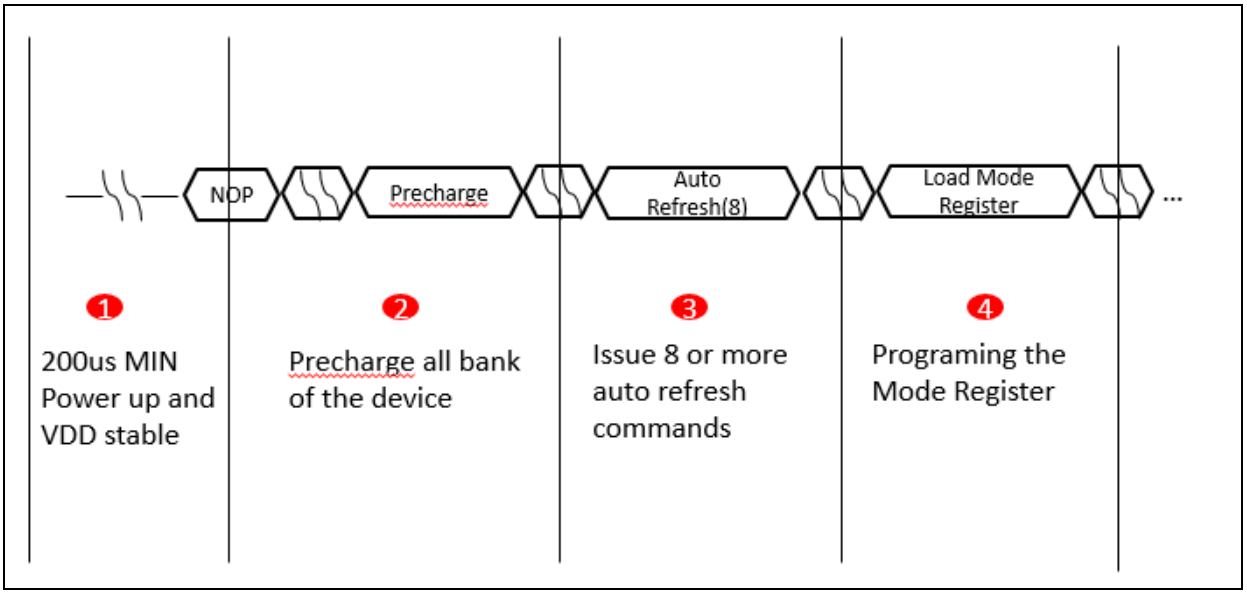
  加载模式寄存器，修改 SDRAM 设备的功能参数，burst 模式，latency 等。

图 3 mode register



1.5 SDRAM Power On Sequence

图 4 SDRAM Power On



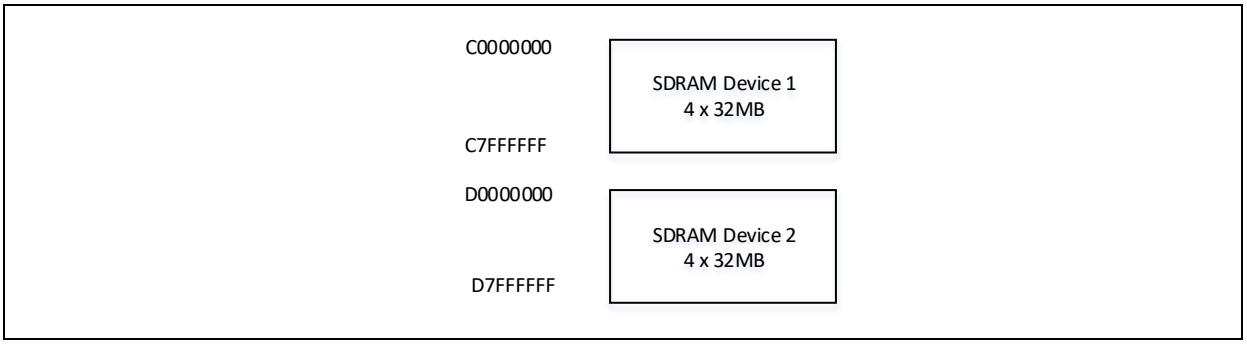
2 AT32 SDRAM 控制器

SDRAM 控制器主要特点如下：

- 同时支持两个 SDRAM 设备
- 支持 8 位/16 位数据总线宽度
- 支持 13 位行地址，11 位列地址（最大可以支持 4x16Mx16bit=128MB）
- 支持 4 个内部 Bank
- 支持 word/half word/byte 访问
- 支持 Burst Read，有 6x32bit 读 FIFO 缓存
- SDRAM 时钟支持 HCLK/2, HCLK/3, HCLK/4
- 支持低功耗模式（自刷新模式，掉电模式）

2.1 地址映射

图 5 SDRAM 地址映射



SDRAM Device1 起始地址：0xC0000000

SDRAM Device2 起始地址：0xD0000000

2.2 I/O 引脚配置

表 2 SDRAM IO 引脚列表

信号线	IOMUX_12	IOMUX_10	IOMUX_14
XMC_SDCLK	PG8	PD13	-
XMC_SDCKE0	PC3/PC5	-	-
XMC_SDCKE1	PB5	-	-
XMC_SDCS0	PC2/PC4	-	-
XMC_SDCS1	PB6	-	
XMC_A0	PF0	PC6	PC3
XMC_A1	PF1	PC7	-
XMC_A2	PF2	PC8	-
XMC_A3	PF3	PC9	-
XMC_A4	PF4		PA8
XMC_A5	PF5	PD0	-
XMC_A6	PF12	PD1	-
XMC_A7	PF13	PD2	-
XMC_A8	PF14	PD3	-
XMC_A9	PF15	PD4	-
XMC_A10	PG0	PD5	-

信号线	IOMUX_12	IOMUX_10	IOMUX_14
XMC_A11	PG1	PD6	-
XMC_A12	PG2	PD7	-
XMC_D0	PD14	-	PB14
XMC_D1	PD15	-	PC6
XMC_D2	PD0	-	PC11
XMC_D3	PD1	-	PC12
XMC_D4	PE7	-	PA2
XMC_D5	PE8	-	PA3
XMC_D6	PE9	-	PA4
XMC_D7	PE10	-	PA5
XMC_D8	PE11	-	-
XMC_D9	PE12	-	-
XMC_D10	PE13	-	-
XMC_D11	PE14	-	-
XMC_D12	PE15	-	-
XMC_D13	PD8	-	PB12
XMC_D14	PD9	-	-
XMC_D15	PD10	-	-
XMC_SDBA0	PG4	PD11	-
XMC_SDBA1	PG5	PD12	-
XMC_SDNRAS	PF11	PE6	-
XMC_SDNCAS	PG15	PE2	-
XMC_SDNWE	PC0/PA7	-	-
XMC_SDDQML	PE0	-	-
XMC_SDDQMH	PE1	-	-

使用 SDRAM IO 引脚初始化如下，可根据具体使用引脚进行修改：

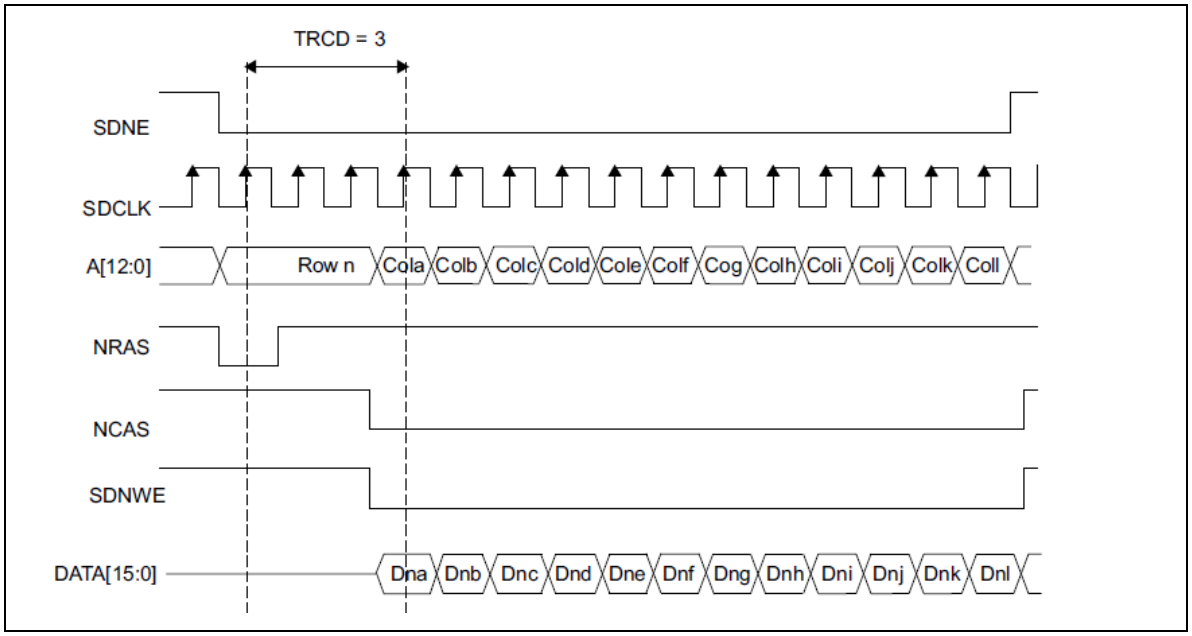
```
/*-- gpio configuration ----------------------------------------------------*/

  gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE5, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE6, GPIO_MUX_12);

  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE0, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE2, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE3, GPIO_MUX_12);

  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE0, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE8, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE9, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE10, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE14, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE15, GPIO_MUX_12);
```

```
  gpio_pin_mux_config(GPIOE, GPIO_PINS_SOURCE0, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOE, GPIO_PINS_SOURCE1, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOE, GPIO_PINS_SOURCE7, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOE, GPIO_PINS_SOURCE8, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOE, GPIO_PINS_SOURCE9, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOE, GPIO_PINS_SOURCE10, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOE, GPIO_PINS_SOURCE11, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOE, GPIO_PINS_SOURCE12, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOE, GPIO_PINS_SOURCE13, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOE, GPIO_PINS_SOURCE14, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOE, GPIO_PINS_SOURCE15, GPIO_MUX_12);

  gpio_pin_mux_config(GPIOF, GPIO_PINS_SOURCE0, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOF, GPIO_PINS_SOURCE1, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOF, GPIO_PINS_SOURCE2, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOF, GPIO_PINS_SOURCE3, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOF, GPIO_PINS_SOURCE4, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOF, GPIO_PINS_SOURCE5, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOF, GPIO_PINS_SOURCE11, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOF, GPIO_PINS_SOURCE12, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOF, GPIO_PINS_SOURCE13, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOF, GPIO_PINS_SOURCE14, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOF, GPIO_PINS_SOURCE15, GPIO_MUX_12);

  gpio_pin_mux_config(GPIOG, GPIO_PINS_SOURCE0, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOG, GPIO_PINS_SOURCE1, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOG, GPIO_PINS_SOURCE2, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOG, GPIO_PINS_SOURCE4, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOG, GPIO_PINS_SOURCE5, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOG, GPIO_PINS_SOURCE8, GPIO_MUX_12);
  gpio_pin_mux_config(GPIOG, GPIO_PINS_SOURCE15, GPIO_MUX_12);

  /* address lines configuration */
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;

  gpio_init_struct.gpio_pins = GPIO_PINS_5 | GPIO_PINS_6;
  gpio_init(GPIOB, &gpio_init_struct);

  gpio_init_struct.gpio_pins = GPIO_PINS_0 | GPIO_PINS_2 | GPIO_PINS_3;
  gpio_init(GPIOC, &gpio_init_struct);

  gpio_init_struct.gpio_pins = GPIO_PINS_0 | GPIO_PINS_1 | GPIO_PINS_8 | GPIO_PINS_9 | GPIO_PINS_10 |
GPIO_PINS_14 | GPIO_PINS_15;
```

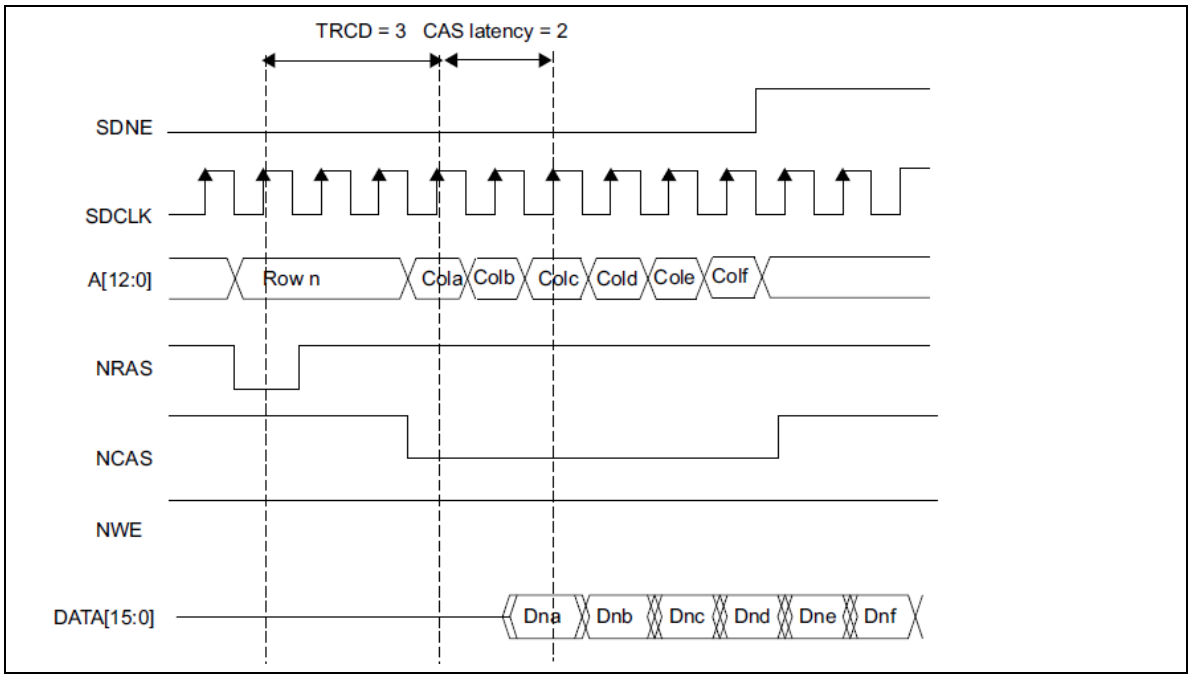
```
gpio_init(GPIOD, &gpio_init_struct);

gpio_init_struct.gpio_pins = GPIO_PINS_0 | GPIO_PINS_1 | GPIO_PINS_7 | GPIO_PINS_8 | GPIO_PINS_10 |
                             GPIO_PINS_9 | GPIO_PINS_11 | GPIO_PINS_11 | GPIO_PINS_12 |
                             GPIO_PINS_13 | GPIO_PINS_14 | GPIO_PINS_15;
gpio_init(GPIOE, &gpio_init_struct);

gpio_init_struct.gpio_pins = GPIO_PINS_0 | GPIO_PINS_1 | GPIO_PINS_2 | GPIO_PINS_3 |
                             GPIO_PINS_4 | GPIO_PINS_5 | GPIO_PINS_11 | GPIO_PINS_12 |
                             GPIO_PINS_13 | GPIO_PINS_14 | GPIO_PINS_15;
gpio_init(GPIOF, &gpio_init_struct);

gpio_init_struct.gpio_pins = GPIO_PINS_0 | GPIO_PINS_1 | GPIO_PINS_2 | GPIO_PINS_4 |
                             GPIO_PINS_5 | GPIO_PINS_8 | GPIO_PINS_15;
gpio_init(GPIOG, &gpio_init_struct);
```

2.3 SDRAM 读写时序

- 写 SDRAM 时序



- 读 SDRAM 时序



2.4 SDRAM 配置

通过配置寄存器 SDRAM_CTRLx 来设置 SDRAM 设备的容量，访问方式等，详细信息可参考 RM。此寄存器包括如下配置：（W9825G6KH 作为示例）

- 行地址/列地址配置

位 3：2	RA	0x0	rw	行地址 (Row address) 行地址位数，支持 11,12,13 位行地址。 00：11 位 01：12 位 10：13 位 11：保留，不使用。
位 1：0	CA	0x0	rw	列地址（Column Address） 列地址位数，支持 8,9,10,11 位列地址 00：8 位 01：9 位 10：10 位 11：11 位

行地址和列地址根据 SDRAM 设备地址位数进行配置，如下示例：

PIN NUMBER	PIN NAME	FUNCTION	DESCRIPTION
23–26, 22, 29–36	A0–A12	Address	A0–A8Multiplexed pins for row and column address. Row address: A0–A12. Column address: A0–A8.

- 数据总线宽度

位 5：4	DB	0x0	rw	SDRAM 数据总线宽度 (Data Bus)。 可以外接 8 位和 16 位数据总线宽度 S 00：8 位 01：16 位 10：保留，不使用。 11：保留，不使用。

根据 SDRAM 设备支持数据总线宽度进行配置，如下示例支持 16bit 数据宽度：

2, 4, 5, 7, 8, 10, 11, 13, 42, 44, 45, 47, 48, 50, 51, 53	DQ0–DQ15	Data Input/Output	Multiplexed pins for data output and input.

- 内部区块个数

位 6	INBK	0x0	rw	内部区块 (internal banks) 定义当前 SDRAM 内部 BANK 数目。 0：2 个内部 BANK 1：4 个内部 BANK

SDRAM 设备支持内部 bank 个数：

2. FEATURES

- 3.3V ± 0.3V Power Supply
- Up to 200 MHz Clock Frequency
- 4,194,304 Words × 4 Banks × 16 Bits Organization
- Self Refresh Mode: Standard and Low Power

- 列地址选通延迟（CAS）

位 8：7	CAS	0x0	rw	列地址选通延迟 (CAS Latency) 选择列地址选通延迟。 00：保留，不使用。 01：1 个周期 10：2 个周期 11：3 个周期

SDRAM 设备支持延迟：

2. FEATURES

- 3.3V ± 0.3V Power Supply
- Up to 200 MHz Clock Frequency
- 4,194,304 Words × 4 Banks × 16 Bits Organization
- Self Refresh Mode: Standard and Low Power
- CAS Latency: 2 and 3

- 写保护配置（WRP）

  如果配置了写保护，在写 SDRAM 设备时会参数 Bus error。
- XMC_SDCLK 时钟分频（CLKDIV）
- BSTR （连续读）
- RD （读延时）

配置代码例程：

```
sdram_init_struct.bank                  = XMC_SDRAM_BANK1;
sdram_init_struct.internel_banks        = XMC_INBK_4;
sdram_init_struct.clkdiv                = XMC_CLKDIV_2;
sdram_init_struct.write_protection      = FALSE;
sdram_init_struct.burst_read            = FALSE;
sdram_init_struct.read_delay            = XMC_READ_DELAY_1;
sdram_init_struct.column_address        = XMC_COLUMN_9;
```

```
sdram_init_struct.row_address          = XMC_ROW_13;
sdram_init_struct.cas                  = XMC_CAS_3;
sdram_init_struct.width                = XMC_MEM_WIDTH_16;
```

2.5 SDRAM 时序参数配置

要正常使用 SDRAM 设备，需要正确配置此部分的实现参数，此参数可在 SDRAM 设备的 datasheet 中找到对应值。

配置寄存器 SDRAM_TMx：

- TMRD（加载模式寄存器到激活延迟）
- TXSR（退出自刷新延迟）
- TRAS（自刷新周期）
- TRC（刷新命令到激活命令延迟）
- TWR（写命令到预充电命令延迟）
- TRP（预充电到激活命令延迟）
- TRCD（行激活到列延迟）

例 TRCD：最小 18ns， SDRAM 时钟 144MHz，一个 SDRAM 时钟大约为 7ns，因此 TRCD 至少要配置为延迟 3 个 SDRAM 时钟周期。

例 TWR：SDRAM 设备要求 2 个 SDRAM 时钟，因此配置为 2

SDRAM 设备对时序要求：（W9825G6KH 作为示例）

PARAMETER		SYM.	-5/-5I		-6		-6I/-6L		-75/75L		UNIT	NOTES
			MIN.	MAX.	MIN.	MAX.	MIN.	MAX.	MIN.	MAX.		
Ref/Active to Ref/Active Command Period		t_{RC}	55		60		60		65		nS	
Active to precharge Command Period		t_{RAS}	40	100000	42	100000	42	100000	45	100000		
Active to Read/Write Command Delay Time		t_{RCD}	15		15		18		20			
Read/Write(a) to Read/Write(b) Command Period		t_{CCD}	1		1		1		1		t_{CK}	
Precharge to Active Command Period		t_{RP}	15		15		18		20		nS	
Active(a) to Active(b) Command Period		t_{RRD}	2		2		2		2		t_{CK}	
Write Recovery Time	CL* = 2	t_{WR}	2		2		2		2		t_{CK}	
	CL* = 3		2		2		2		2			
CLK Cycle Time	CL* = 2	t_{CK}	7.5	1000	7.5	1000	7.5	1000	10	1000		
	CL* = 3		5	1000	6	1000	6	1000	7.5	1000		
CLK High Level width		t_{CH}	2		2		2		2.5			8
CLK Low Level width		t_{CL}	2		2		2		2.5			8
Access Time from CLK	CL* = 2	t_{AC}		6		6		6		6		9
	CL* = 3			4.5		5		5		5.4		
Output Data Hold Time		t_{OH}	3		3		3		3			9
Output Data High Impedance Time	CL* = 2	t_{HZ}		6		6		6		6		7
	CL* = 3			4.5		5		5		5.4		

配置代码例程：

```
sdram_timing_struct.tmrd               = XMC_DELAY_CYCLE_2;
sdram_timing_struct.txsr               = XMC_DELAY_CYCLE_11;
sdram_timing_struct.tras               = XMC_DELAY_CYCLE_7;
```

```
sdram_timing_struct.trc                    = XMC_DELAY_CYCLE_9;
sdram_timing_struct.twr                    = XMC_DELAY_CYCLE_2;
sdram_timing_struct.trp                    = XMC_DELAY_CYCLE_3;
sdram_timing_struct.trcd                   = XMC_DELAY_CYCLE_3;
```

2.6 SDRAM 启动序列配置

- Clock enable 时钟使能

```
sdram_cmd_struct.cmd                       = XMC_CMD_CLK;
sdram_cmd_struct.auto_refresh              = 1;
sdram_cmd_struct.cmd_banks                 = cmd_bank;
sdram_cmd_struct.data                      = 0;
xmc_sdram_cmd(&sdram_cmd_struct);

while((xmc_flag_status_get(XMC_BANK5_6_SDRAM, XMC_BUSY_FLAG) != RESET) && (timeout > 0))
{
        timeout --;
}
 delay_ms(100);
```

- 预充电

```
sdram_cmd_struct.cmd                       = XMC_CMD_PRECHARG_ALL;
sdram_cmd_struct.auto_refresh              = 1;
sdram_cmd_struct.cmd_banks                 = cmd_bank;
sdram_cmd_struct.data                      = 0;
xmc_sdram_cmd(&sdram_cmd_struct);

timeout = 0xFFFF;
while((xmc_flag_status_get(XMC_BANK5_6_SDRAM, XMC_BUSY_FLAG) != RESET) && (timeout > 0))
{
        timeout --;
}
```

- 设置刷新计数器

2. FEATURES

- 3.3V ± 0.3V Power Supply
- Up to 200 MHz Clock Frequency
- 4,194,304 Words × 4 Banks × 16 Bits Organization
- Self Refresh Mode: Standard and Low Power
- CAS Latency: 2 and 3
- Burst Length: 1, 2, 4, 8 and Full Page
- Burst Read, Single Writes Mode
- Byte Data Controlled by LDQM, UDQM
- Power Down Mode
- Auto-precharge and Controlled Precharge
- 8K Refresh Cycles/64 mS

计算方法：counter = （SDRAM resfresh period / number of rows）- 20;

刷新速率 = 64ms / 8K = 7.8125us;

counter = 7.8125us * 144MHz – 20 = 1105;

```
/* counter = (refresh_count * 1000 * SDCLK) / row - 20 */
/* counter = (64ms * 1000 * 144MHz ) / 2^13 - 20 */
xmc_sdram_refresh_counter_set(1105);
```

- 自动刷新

```
sdram_cmd_struct.cmd                    = XMC_CMD_AUTO_REFRESH;
sdram_cmd_struct.auto_refresh           = 8;
sdram_cmd_struct.cmd_banks              = cmd_bank;
sdram_cmd_struct.data                   = 0;
xmc_sdram_cmd(&sdram_cmd_struct);

timeout = 0xFFFF;
while((xmc_flag_status_get(XMC_BANK5_6_SDRAM, XMC_BUSY_FLAG) != RESET) && (timeout > 0))
{
  timeout --;
}
```

- 加载模式寄存器

```
sdram_cmd_struct.cmd                    = XMC_CMD_LOAD_MODE;
sdram_cmd_struct.auto_refresh           = 1;
sdram_cmd_struct.cmd_banks              = cmd_bank;
sdram_cmd_struct.data                   = (uint32_t)SDRAM_BURST_LEN_1 |
                                          SDRAM_BURST_SEQUENTIAL |
                                          SDRAM_CAS_LATENCY_3 |
                                          SDRAM_OPERATING_MODE_STANDARD |
                                          SDRAM_WR_BURST_SINGLE;
xmc_sdram_cmd(&sdram_cmd_struct);

timeout = 0xFFFF;
while((xmc_flag_status_get(XMC_BANK5_6_SDRAM, XMC_BUSY_FLAG) != RESET) && (timeout > 0))
{
        timeout --;
}
```

2.7 SDRAM 例程

2.7.1 SDRAM Basic

此例程配置 SDRAM 设备之后，对 SDRAM 设备进行读写操作，并判断读写数据是否正确，包括如下步骤：

- GPIO 初始化
- SDRAM 配置

```
xmc_sdram_default_para_init(&sdram_init_struct, &sdram_timing_struct);
```

```
sdram_init_struct.bank                          = XMC_SDRAM_BANK1;
sdram_init_struct.internel_banks                = XMC_INBK_4;
sdram_init_struct.clkdiv                        = XMC_CLKDIV_2;
sdram_init_struct.write_protection              = FALSE;
sdram_init_struct.burst_read                    = FALSE;
sdram_init_struct.read_delay                    = XMC_READ_DELAY_1;
sdram_init_struct.column_address                = XMC_COLUMN_9;
sdram_init_struct.row_address                   = XMC_ROW_13;
sdram_init_struct.cas                           = XMC_CAS_3;
sdram_init_struct.width                         = XMC_MEM_WIDTH_16;

sdram_timing_struct.tmrd                        = XMC_DELAY_CYCLE_2;
sdram_timing_struct.txsr                        = XMC_DELAY_CYCLE_11;
sdram_timing_struct.tras                        = XMC_DELAY_CYCLE_7;
sdram_timing_struct.trc                         = XMC_DELAY_CYCLE_9;
sdram_timing_struct.twr                         = XMC_DELAY_CYCLE_2;
sdram_timing_struct.trp                         = XMC_DELAY_CYCLE_3;
sdram_timing_struct.trcd                        = XMC_DELAY_CYCLE_3;
xmc_sdram_init(&sdram_init_struct, &sdram_timing_struct);
```

- SDRAM 启动序列

```
void sdram_init_sequence(xmc_cmd_bank1_2_type cmd_bank)
{
  xmc_sdram_cmd_type sdram_cmd_struct;
  uint32_t timeout = 0xFFFF;

  sdram_cmd_struct.cmd                          = XMC_CMD_CLK;
  sdram_cmd_struct.auto_refresh                 = 1;
  sdram_cmd_struct.cmd_banks                    = cmd_bank;
  sdram_cmd_struct.data                         = 0;
  xmc_sdram_cmd(&sdram_cmd_struct);

  while((xmc_flag_status_get(XMC_BANK5_6_SDRAM, XMC_BUSY_FLAG) != RESET) && (timeout > 0))
  {
    timeout --;
  }
  delay_ms(100);

  sdram_cmd_struct.cmd                          = XMC_CMD_PRECHARG_ALL;
  sdram_cmd_struct.auto_refresh                 = 1;
  sdram_cmd_struct.cmd_banks                    = cmd_bank;
  sdram_cmd_struct.data                         = 0;
  xmc_sdram_cmd(&sdram_cmd_struct);

  timeout = 0xFFFF;
```

```
  while((xmc_flag_status_get(XMC_BANK5_6_SDRAM, XMC_BUSY_FLAG) != RESET) && (timeout > 0))
      {
      timeout --;
   }

  /* counter = (refresh_count * 1000 * SDCLK) / row - 20 */
  /* counter = (64ms * 1000 * 144MHz) / 2^13 - 20 */
  xmc_sdram_refresh_counter_set(1105);

  sdram_cmd_struct.cmd                  = XMC_CMD_AUTO_REFRESH;
  sdram_cmd_struct.auto_refresh         = 8;
  sdram_cmd_struct.cmd_banks            = cmd_bank;
  sdram_cmd_struct.data                 = 0;
  xmc_sdram_cmd(&sdram_cmd_struct);

  timeout = 0xFFFF;
  while((xmc_flag_status_get(XMC_BANK5_6_SDRAM, XMC_BUSY_FLAG) != RESET) && (timeout > 0))
  {
    timeout --;
  }

  sdram_cmd_struct.cmd                  = XMC_CMD_LOAD_MODE;
  sdram_cmd_struct.auto_refresh         = 1;
  sdram_cmd_struct.cmd_banks            = cmd_bank;
  sdram_cmd_struct.data    = (uint32_t)SDRAM_BURST_LEN_1 |
                             SDRAM_BURST_SEQUENTIAL |
                             SDRAM_CAS_LATENCY_3 |
                             SDRAM_OPERATING_MODE_STANDARD |
                             SDRAM_WR_BURST_SINGLE;
  xmc_sdram_cmd(&sdram_cmd_struct);

  timeout = 0xFFFF;
  while((xmc_flag_status_get(XMC_BANK5_6_SDRAM, XMC_BUSY_FLAG) != RESET) && (timeout > 0))
  {
    timeout --;
  }
}
```

- SDRAM 读写访问

```
/**
  * @brief    writes a half-word buffer to the sdram memory.
  * @param    pbuffer : pointer to buffer.
  * @param    writeaddr : sdram memory internal address from which the data will be
  *             written.
  * @param    numhalfwordtowrite : number of half-words to write.
  * @retval none
```

```
  */
void sdram_writebuffer(uint16_t* pbuffer, uint32_t writeaddr, uint32_t numhalfwordtowrite)
{
  for(; numhalfwordtowrite != 0; numhalfwordtowrite--) /*!< while there is data to write */
  {
    /*!< Transfer data to the memory */
    *(uint16_t *) (SDRAM_BANK1_ADDR + writeaddr) = *pbuffer++;

    /*!< increment the address*/
    writeaddr += 2;
  }
}

/**
  * @brief   reads a block of data from the sdram.
  * @param   pbuffer : pointer to the buffer that receives the data read from the
  *                sdram memory.
  * @param   readaddr : sdram memory internal address to read from.
  * @param   numhalfwordtoread : number of half-words to read.
  * @retval none
  */
void sdram_readbuffer(uint16_t* pbuffer, uint32_t readaddr, uint32_t numhalfwordtoread)
{
  for(; numhalfwordtoread != 0; numhalfwordtoread--) /*!< while there is data to read */
  {
    /*!< read a half-word from the memory */
    *pbuffer++ = *(__IO uint16_t*) (SDRAM_BANK1_ADDR + readaddr);

    /*!< increment the address*/
    readaddr += 2;
  }
}
```

2.7.2 SDRAM DMA

此例程配置 SDRAM 设备之后，使用对 SDRAM 设备进行读写操作，并判断读写数据是否正确，包括如下步骤：

- GPIO 初始化（同 SDRAM Basic）
- SDRAM 配置（同 SDRAM Basic）
- SDRAM 启动序列（同 SDRAM Basic）
- SDRAN DMA 读写

```
void sdram_copy_to_sram_dma(uint16_t *buffer, uint32_t length)
{
  crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);
  /* dma1 channel1 configuration */
```

```
  dma_reset(DMA1_CHANNEL1);
  dma_init_struct.buffer_size = length;
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_MEMORY;
  dma_init_struct.memory_base_addr = (uint32_t)buffer;
  dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_HALFWORD;
  dma_init_struct.memory_inc_enable = TRUE;
  dma_init_struct.peripheral_base_addr = (uint32_t)SDRAM_BANK1_ADDR;
  dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_HALFWORD;
  dma_init_struct.peripheral_inc_enable = TRUE;
  dma_init_struct.priority = DMA_PRIORITY_MEDIUM;
  dma_init_struct.loop_mode_enable = FALSE;
  dma_init(DMA1_CHANNEL1, &dma_init_struct);
  dma_channel_enable(DMA1_CHANNEL1, TRUE);
  while(dma_flag_get(DMA1_FDT1_FLAG) == RESET)
  {
  }
  dma_channel_enable(DMA1_CHANNEL1, FALSE);
}

void sram_copy_to_sdram_dma(uint16_t *buffer, uint32_t length)
{
  crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);
  /* dma1 channel1 configuration */
  dma_reset(DMA1_CHANNEL1);
  dma_init_struct.buffer_size = length;
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_MEMORY;
  dma_init_struct.memory_base_addr = (uint32_t)SDRAM_BANK1_ADDR;
  dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_HALFWORD;
  dma_init_struct.memory_inc_enable = TRUE;
  dma_init_struct.peripheral_base_addr = (uint32_t)buffer;
  dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_HALFWORD;
  dma_init_struct.peripheral_inc_enable = TRUE;
  dma_init_struct.priority = DMA_PRIORITY_MEDIUM;
  dma_init_struct.loop_mode_enable = FALSE;
  dma_init(DMA1_CHANNEL1, &dma_init_struct);
  dma_channel_enable(DMA1_CHANNEL1, TRUE);
  while(dma_flag_get(DMA1_FDT1_FLAG) == RESET)
  {
  }
  dma_channel_enable(DMA1_CHANNEL1, FALSE);
}
```

3 文档版本历史

表 3. 文档版本历史

日期	版本	变更
2021.11.5	2.0.0	最初版本

重要通知 - 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途（及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：（A）对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）航天应用或航天环境；（D）武器，且/或（E）其他可能导致人身伤害、死亡及财产损害的应用。如果采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险及法律责任仍将由采购商单独承担，且采购商应独力负责在前述应用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。