应用笔记

ACM32H5 系列芯片

OSPI 应用笔记

版本：V1.0

日期：2025-3-20

上海航芯电子科技股份有限公司

1. 概述

本应用手册适用于 ACM32H5 系列芯片 OSPI 模块。它描述了与 OSPI 模块相关的设置和功能使用方法，以便在应用程序中进行优化设计。

本应用说明应与相关的用户手册、数据手册一同阅读。

2. OSPI 模块初始化

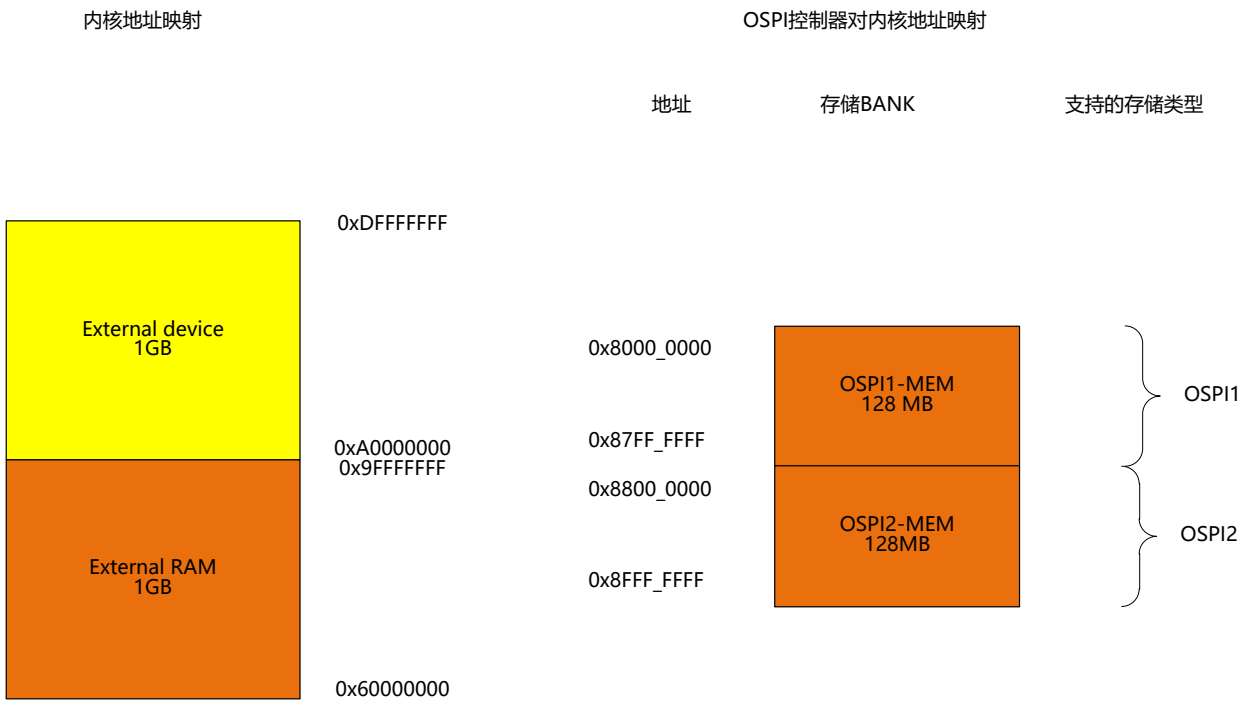
2.1. 主要特性

- 功能模式：间接模式和内存映射模式
  - 间接模式(FIFO 模式)：使用 OSPI 寄存器执行全部操作
  - 内存映射模式(Memory/XIP 模式)：外部存储器映射到微控制器地址空间，从而系统将其视作内部存储器
- 在内存映射模式下支持读写操作
- 可通过两级分频因子来配置宽范围波特率
- 支持 Mode0/1/2 /3 四种传输协议（除八线 DTR 模式）
- 支持 OSPI 一线、二线、四线、八线传输
- 支持 SDR 和 DTR 模式(仅八线)
- 支持 HyperBus™协议、 xOSPI 协议、Xccela 协议等
- 间接模式支持 DMA 访问
- 发送/接收数据 FIFO 大小为 16 个字节

2.2. 接口信号

OSPI 引脚名称	说明
OSPI_NCLK	差分时钟 根据 CLK 和 NCLK 信号的交叉输出命令、地址和数据信息
OSPI_CLK	单端时钟 不使用 NCLK，只使用单端 CLK。时钟不需要自由运行
OSPI_NCS	片选 总线事务以高到低的转换启动。总线事务以从低到高的转换终止
OSPI_DQS	读取期间的 DQ 选通时钟，写入期间的数据掩码 在 HyperBus™协议、 xOSPI 协议、Xccela 协议的命令/地址阶段，DQS 是一个从属输出，指示是否需要额外的初始延迟
OSPI_IOn(n = 0 to 7)	数据信号 在读写事务期间，命令、地址和数据信息在这些信号上传输

2.3. 内存地址映射



起始地址-结束地址	OSPI
0x80000000-0x87FFFFFF	OSPI1
0x88000000-0x8FFFFFFF	OSPI2

注意：External RAM 区可以直接执行代码，而 External device 区却不支持这个功能。

2.4. 结构体说明

2.4.1. OSPI_HandleTypeDef 结构体

```
typedef struct
{
    OSPI_TypeDef        *Instance;          /* 用于设置 OSPI 寄存器基地址 */
    OSPI_InitTypeDef    Init;              /* 用于设置 OSP 初始化的相关参数 */
    uint32_t            CSx;               /* 用于选择 CS 控制信号 */
    uint32_t            RxState;           /* 用于指示 OSPI 接收通信状态 */
    uint32_t            TxState;           /* 用于指示 OSPI 发送通信状态 */
    uint8_t             *Rx_Buffer;        /* 用于设置 OSPI 接收缓冲指针 */
    uint8_t             *Tx_Buffer;        /* 用于设置 OSPI 发送缓冲指针 */
    uint32_t            Rx_Size;           /* 用于设置接收数据量 */
    uint32_t            Tx_Size;           /* 用于设置发送数据量 */
    uint32_t            Rx_Count;          /* 用于设置已接收的数据量 */
    uint32_t            Tx_Count;          /* 用于设置已发送的数据量 */
```

```
    DMA_HandleTypeDef    *HDMA_Rx;        /* 接收 DMA 通道句柄指针 */
    DMA_HandleTypeDef    *HDMA_Tx;        /* 发送 DMA 通道句柄指针 */
}OSPI_HandleTypeDef;
```

2.4.2. 初始化结构体

```
typedef struct
{
    uint32_t    WorkMode;           /* 用于设置工作模式(时钟极性和时钟相位) */
    uint32_t    XMode;              /* 用于设置 1/2/4/8 线模式 */
    uint32_t    FirstBit;           /* 用于设置 LSB 或 MSB */
    uint32_t    BaudRatePrescaler;  /* 用于设置预分频系数 */
    uint32_t    SampleShifting;     /* 用于设置采样移位 */
    uint32_t    FWMode;             /* 用于设置 FIFO 模式(字节/半字/字) */
    uint32_t    FRMode;             /* 用于设置 FIFO 读模式(字节/半字/字) */
}OSPI_InitTypeDef;
```

2.4.3. 八线模式结构体

```
typedef struct
{
    uint32_t    DTRMode;            /* 单/双倍(STR/DTR)传输速率模式下传输 */
    uint32_t    DQSMode;            /* DQS 线输出使能或禁止 */
    uint32_t    MemoryType;         /* 存储器类型(Xccela/APM/Hyperbus/xSPI) */
    uint32_t    OutDelay;           /* 输出延迟 */
    uint32_t    DQSSample;          /* 使用 DQS 作为时钟采样 */
}OSPI_OctalInitTypeDef;
```

2.4.4. 内存映射模式结构体

```
typedef struct
{
    uint8_t     ReadCmd;            /* 读指令 */
    uint8_t     WriteCmd;           /* 写指令 */
    uint32_t    AlterByte;          /* 交替字节 */
    uint32_t    CsTimeoutVal        /* CS 低超时拉高时间 */
    uint32_t    HyperBurstType;     /* 突发模式(Linear 或 Wrapped) */
    uint32_t    DataMode;           /* 数据模式(1/2/4 线) */
    uint32_t    AlterByteMode;      /* 交替字节模式(1/2/4 线) */
    uint32_t    AddrMode;           /* 地址模式(1/2/4 线) */
```

```
    uint32_t    InstrMode;              /* 指令模式(1/2/4 线) */
    uint32_t    AddrWidth;              /* 地址位数(8/16/32 bit)
    uint32_t    DummyCycleSize;         /* 空指令周期长度 */
    uint32_t    ReadDummyByteEnable;    /* 读操作空指令使能 */
    uint32_t    WriteummyByteEnable;    /* 写操作空指令使能 */
    uint32_t    AlterByteSize;          /* 交替字节长度 */
    uint32_t    ReadAlterByteEnable;    /* 读操作交替字节使能 */
    uint32_t    WriteAlterByteEnable;   /* 写操作交替字节使能 */
    uint32_t    SendInstrOnce;          /* 仅发送一次指令 */
    uint32_t    ContinuousModeEnable;   /* 连续模式使能 */
    uint32_t    CsTimeoutEnable;        /* CS 低超时拉高使能 */
    uint32_t    WrapSize;               /* 回卷大小 */
    uint32_t    BurstLen;               /* 突发长度 */
    uint32_t    HyperXspiLC1;           /* Hyperbus/xSPI 模式下 RWDS=1 时 Latency */
    uint32_t    HyperXspiLc0;           /* Hyperbus/xSPI 模式下 RWDS=0 时 Latency */
}OSPI_MemoryInitTypeDef;
```

2.5. OSPI 初始化配置

利用 HAL 库的 OSPI_HandleTypeDef 结构体、八线模式结构体以及内存映射结构体可以很方便地写入参数。下面将描述具体的配置方法和过程。

首先定义一个 OSPI 的总结构体变量，例如：

```
OSPI_HandleTypeDef HYPER_Handle;
```

定义一个 OSPI 八线模式结构体变量，例如：

```
OSPI_OctalInitTypeDef HYPER_OSPI_Octal;
```

以及定义一个 OSPI 内存映射结构体变量，例如：

```
OSPI_MemoryInitTypeDef Hyper_Memeory_Handle;
```

2.5.1. 基本配置

- Instance：

```
HYPER_Handle.Instance                = OSPI1;    //OSPI 寄存器基地址，这里选择 OSPI1
```

- WorkMode：

```
HYPER_Handle.Init.WorkMode           = OSPI_WORK_MODE_0;  //OPSI 工作模式，时钟极性:0，时钟相位:0
```

- XMode：

```
HYPER_Handle.Init.XMode              = OSPI_8X_MODE; //1 线、2 线、4 线、8 线选择
```

- FirstBit：

```
HYPER_Handle.Init.FirstBit           = OSPI_FIRSTBIT_MSB;    //OSPI 总线传输中 MSB 或 LSB 在前
```

- BaudRatePrescaler：

```
HYPER_Handle.Init.BaudRatePrescaler  = OSPI_BAUDRATE_PRESCALER_2;    //分频系数
```

- SampleShifting：

```
HYPER_Handle.Init.SampleShifting     = OSPI_SAMPLE_SHIFT_2_5HCLK;    //采样延时，主机延迟 2.5hclk 周期开始采集数据
```

- FWMode：

```
HYPER_Handle.Init.FWMode             = OSPI_FIFO_HALFWORD;    //FIFO 写模式(字/半字/字节)
```

- FRMode：

```
HYPER_Handle.Init.FRMode             = OSPI_FIFO_HALFWORD;    //FIFO 读模式(字/半字/字节)
```

- CSx：

```
HYPER_Handle.CSx                    = OSPI_CS_0;     //CS 控制信号，CS0 控制信号(OSPI_CS)
```

2.5.2. 八线模式配置

- DTRMode：

```
HYPER_OSPI_Octal.DTRMode = OSPI_DTRM_DTR;  //双或单倍传输速率模式下传输(DTR/SDR)
```

- DQSMode：

```
HYPER_OSPI_Octal.DQSMode = OSPI_DQSOE_ENABLE;  //数据选通输出控制位
```

- MemoryType：

```
HYPER_OSPI_Octal.MemoryType = OSPI_MEM_HYPERBUS;    //存储器类型(Hyperbus、xOSPI、Xccela)
```

- OutDelay：

```
HYPER_OSPI_Octal.OutDelay = OSPI_TX_OUT_DELAY_HALF_HCLK;    //DTR 通信输出延迟，延迟 0.5HCLK 周期
```

- DQSSample：

```
HYPER_OSPI_Octal.DQSSample = OSPI_DQS_SAMPLE_DISABLE;  //DQS 采样控制位，使能后，接收数据时，数据将会使用 DQS 作为时钟对输入数据进行采样
```

2.5.3. 内存映射模式配置

- ReadCmd：

```
Hyper_Memeory_Handle.WriteCmd = 0x00; //写指令
```

- WriteCmd：

```
Hyper_Memeory_Handle.ReadCmd = 0x00;  //读指令
```

- AlterByte：

```
Hyper_Memeory_Handle.AlterByte = 0x00;  //交替字节
```

- CsTimeoutVal：

```
Hyper_Memeory_Handle.CsTimeoutVal = 100;   //CS 拉低后到强制拉高 CS 的时间值
```

- HyperBurstType：

```
Hyper_Memeory_Handle.HyperBurstType = MEMOACC1_BURST_WRAPPED;  //突发是 Linear 的还是 Wrapped 的
```

- DataMode：

```
Hyper_Memeory_Handle.DataMode = MEMOACC1_DATA_MODE_x1;   //数据模式，八线模式时，此位无效
```

- AlterByteMode：

```
Hyper_Memeory_Handle.AlterByteMode = MEMOACC1_ALTER_BYTE_MODE_x1;   //交替字节模式，八线模式时，此位无效
```

- AddrMode：

```
Hyper_Memeory_Handle.AddrMode = MEMOACC1_ADDR_MODE_x1; //地址模式，八线模式时，此位无效
```

- InstrMode：

```
Hyper_Memeory_Handle.InstrMode = MEMOACC1_INSTR_MODE_x1;  //指令模式，八线模式时，此位无效
```

- AddrWidth：

```
Hyper_Memeory_Handle.AddrWidth = MEMOACC1_ADDR_WIDTH_32;  //地址长度（Hyper 必须为 32 bit）
```

- DummyCycleSize：

```
Hyper_Memeory_Handle.DummyCycleSize = MEMOACC1_DUMMY_CYCLE_1;   // dummy 周期长度（Hyper/xSPI 无效）
```

- ReadDummyByteEnable：

```
Hyper_Memeory_Handle.ReadDummyByteEnable = MEMOACC1_READ_DUMMY_ENABLE;  //读操作空指令字节使能（Hyper 必须为 1）， dummy 使能位，有 dummy 时必须置 1
```

- WriteummyByteEnable：

```
Hyper_Memeory_Handle.WriteummyByteEnable = MEMOACC1_WRITE_DUMMY_ENABLE;   //写操作空指令字节使能（Hyper 必须为 1），dummy 使能位，有 dummy 时必须置 1
```

- AlterByteSize：

```
Hyper_Memeory_Handle.AlterByteSize = MEMOACC1_ALTER_BYTE_SIZE_8;  //交替字节长度 8、16、24、32，八线模式时，此位无效
```

- ReadAlterByteEnable：

```
Hyper_Memeory_Handle.ReadAlterByteEnable = MEMOACC1_READ_ALTER_DISABLE;    //读操作交替字节使能位，八线模式时，此位无效
```

- WriteAlterByteEnable：

```
Hyper_Memeory_Handle.WriteAlterByteEnable = MEMOACC1_WRITE_ALTER_DISABLE;   //写操作交替字节使能位，八线模式时，此位无效
```

- SendInstrOnce：

```
Hyper_Memeory_Handle.SendInstrOnce = MEMOACC1_CMD_SEND_ONCE_DISABLE;    //仅发送一次指令控制位，该位使能后，只会发送一次读写指令
```

- ContinuousModeEnable：

```
Hyper_Memeory_Handle.ContinuousModeEnable = MEMOACC1_CON_ENABLE;    //连续读使能控制位
```

- CsTimeoutEnable：

```
Hyper_Memeory_Handle.CsTimeoutEnable = MEMOACC1_CS_TIMEOUT_DISABLE;  //连续模式下 CS 拉高等待功能使能控制位
```

- WrapSize：

```
Hyper_Memeory_Handle.WrapSize = 0;   //回卷大小，该配置大小对应设置的存储器回卷长度
```

- BurstLen：

```
Hyper_Memeory_Handle.BurstLen = MEMOACC2_BURST_LEN_32;   //突发长度（Burst Length）
```

- HyperXspiLC1：

```
Hyper_Memeory_Handle.HyperXspiLC1 = MEMOACC2_LATENCY1_13;  // Hyperbus 或 xSPI 模式下，DQS 为 1 时的 latency 周期数
```

- HyperXspiLc0：

```
Hyper_Memeory_Handle.HyperXspiLc0 = MEMOACC2_LATENCY0_6;  // Hyperbus 或 xSPI 模式下，DQS 为 0 时的 latency 周期数
```

3. OSPI 命令序列

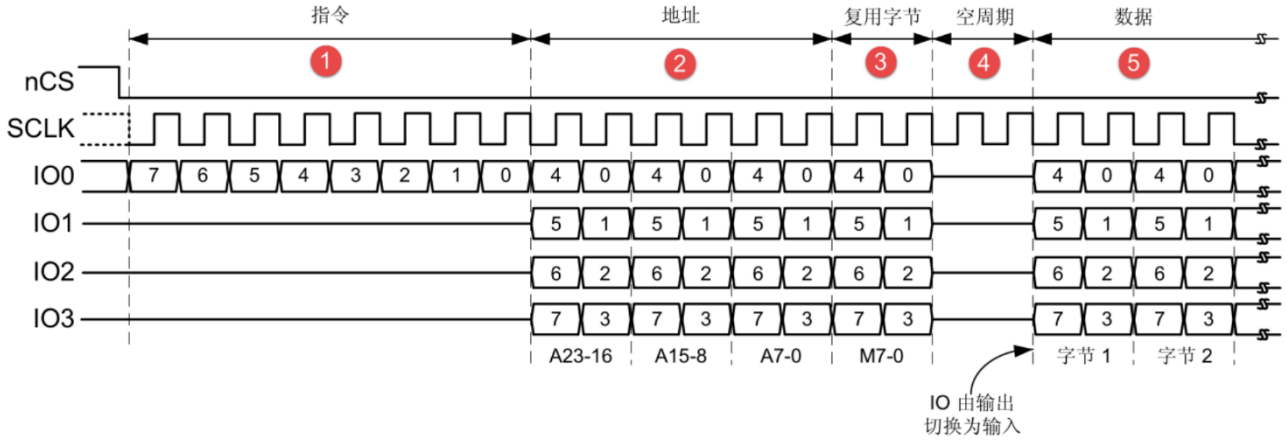
3.1. OSPI 常规命令协议

当使用常规命令协议时，OCTOPI 与外部设备通信使用命令。每个命令可以包括以下阶段：

1. 指令阶段：将一条指令发送到外部存储器，指定待执行操作的类型
2. 地址阶段：将 1-4 字节发送到外部存储器，指示操作地址
3. 交替字节阶段：将 1-4 字节发送到外部存储器，一般用于控制操作模式
4. 空指令周期阶段：给定的周期内不发送或接收任何数据，目的是给外部存储器留出准备数据阶段的时间，或插入内部刷新的时间。
5. 数据阶段：可从外部存储器接收或向其发送任意数量的字节。

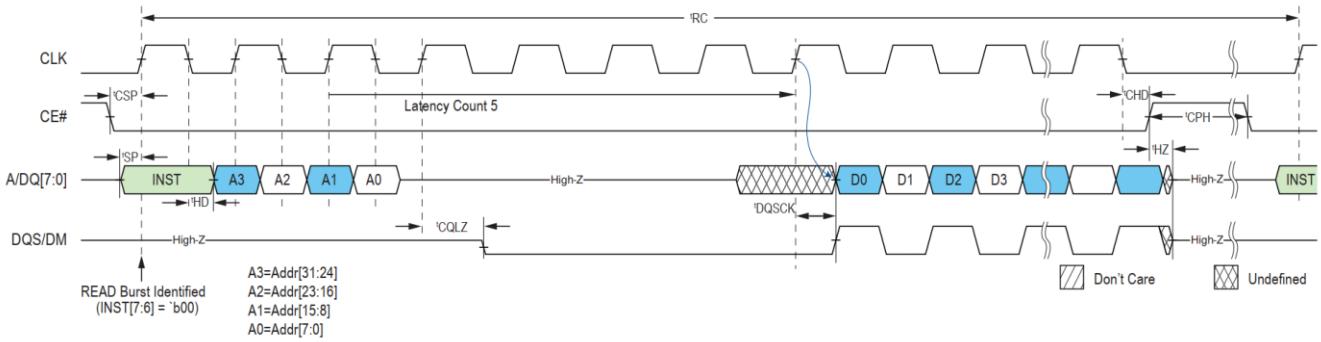
这些阶段中的任何一个都可以被配置为跳过，但是，在单阶段命令的情况下，唯一支持的用例是指令阶段。

NCS 在每个命令开始前下降，在每个命令结束后再次上升。



3.2. Xccela OPI 协议

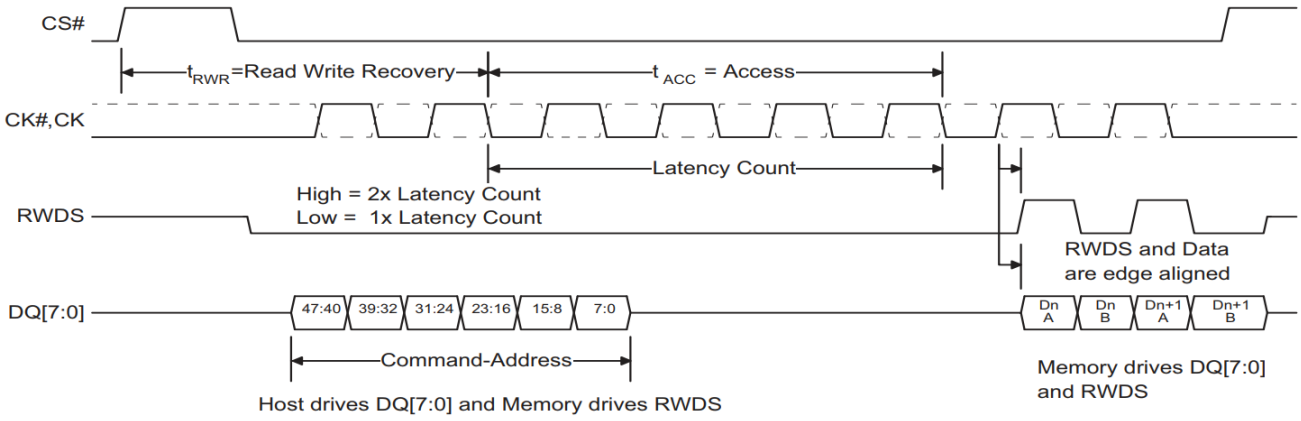
- 指令阶段：2 字节
- 地址阶段：4 字节（A3、A2、A1、A0）
- 空指令周期阶段：Latency 是从发地址 A1 开始计数
- 数据阶段



3.3. HyperBus 协议

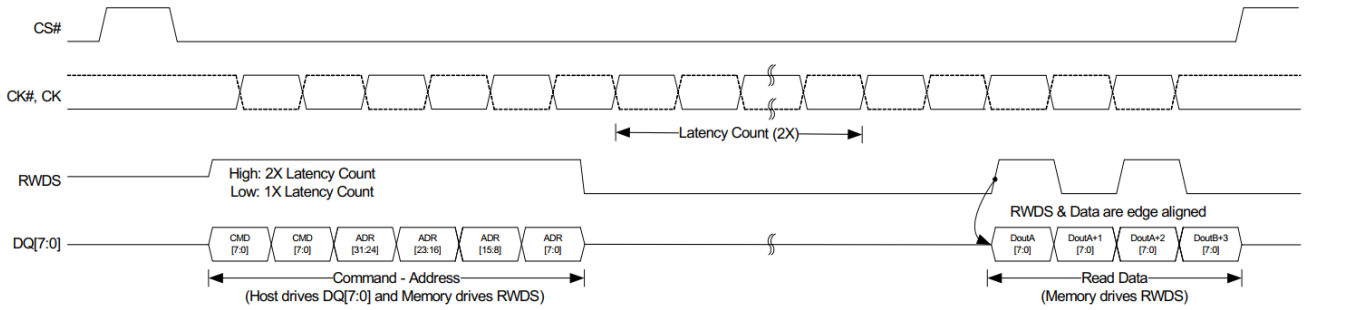
- 指令/地址阶段：6 字节（CA[47:40]、CA[39:2]、CA[31:24]、CA[23:16]、CA[15:8]、CA[7:0]）

- 空指令周期阶段：Latency 是从发地址 CA2[15:8]开始计数。如果 DQS（RWDS）在 CA 循环期间为 LOW，插入一个延迟计数。如果在 CA 循环期间 DQS（RWDS）为 HIGH，则插入了额外的延迟计数。
- 数据阶段



3.4. xSPI 协议

- 指令阶段：2 字节（CMD[7:0]、CMD[7:0]）
- 地址阶段：4 字节（ADDR[31:24]、ADDR[23:16]、ADDR[15:8]、ADDR[7:0]）
- 空指令周期阶段：Latency 是从发完地址（ADDR[7:0]）之后开始计数。如果 DQS（RWDS）在 CA 循环期间为 LOW，插入一个延迟计数。如果在 CA 循环期间 DQS（RWDS）为 HIGH，则插入了额外的延迟计数。
- 数据阶段



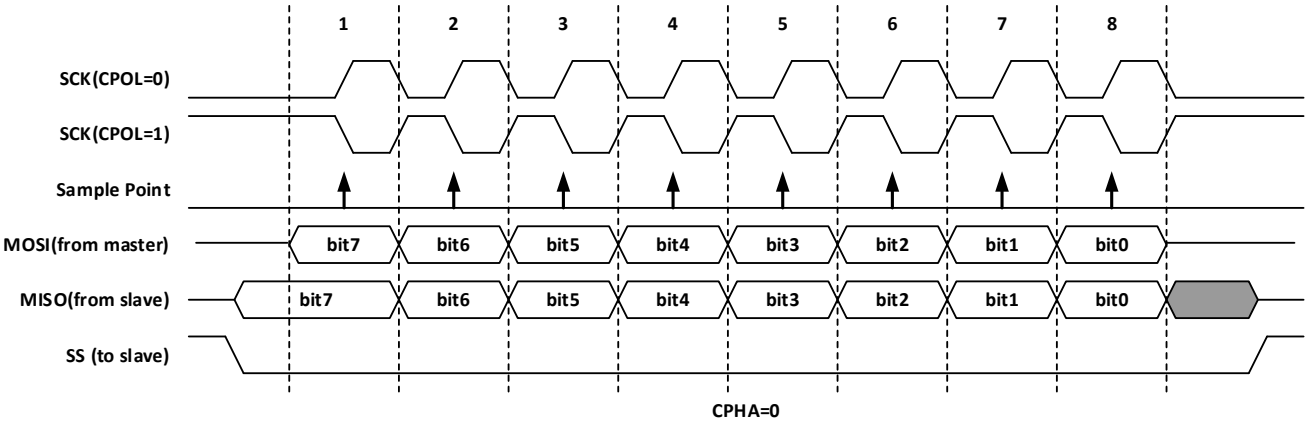
4. OSPI 多线模式

4.1. 一线 SPI 模式

传统 SPI 模式允许串行发送/接收单独的 1 位。在此模式下，数据通过 SO 信号（其 I/O 与 IO1 共享）发送到外部存储器。从外部存储器接收到的数据通过 SI（其 I/O 与 IO0 共享）送达。通过将（OSPI_CTL[6:5]）X_MODE 字段设置为 00 来选择一线模式。在每个已配置为一线模式的阶段中：

- IO0 (MOSI) 处于输出模式
- IO1 (MISO) 处于输入模式（高阻抗）
- IO2 (WP) 处于输出模式并强制置“0”（以禁止“写保护”功能）
- IO3 (HOLD) 处于输出模式并强制置“1”（以禁止“保持”功能）

下图为 OSPI 一线模式时序图（MSB）：

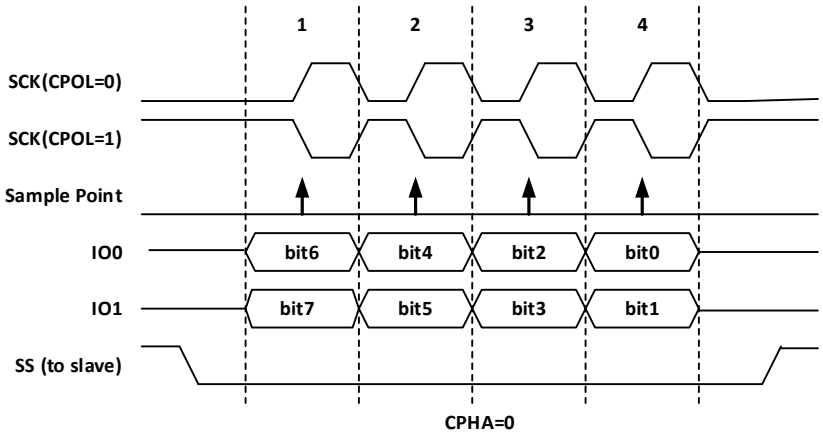


4.2. 二线 SPI 模式

在二线模式下，通过 IO0/IO1 信号同时发送/接收两位。通过将（OSPI_CTL[6:5]）X_MODE 字段设置为 01 来选择二线模式。在每个已配置为二线模式的阶段中：

- IO0/IO1 在数据阶段进行读取操作时处于高阻态（输入），在其他情况下为输出
- IO2 处于输出模式并强制置“0”
- IO3 处于输出模式并强制置“1”

下图为 OSPI 二线模式时序图（MSB）：

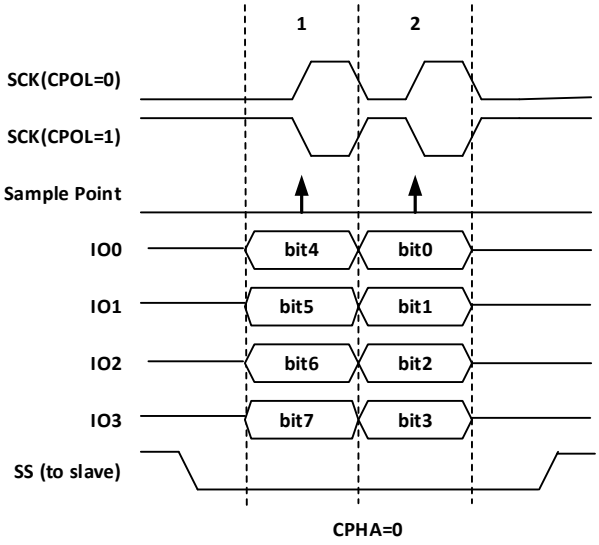


4.3. 四线 SPI 模式

在四线模式下，通过 IO0/IO1/IO2/IO3 信号同时发送/接收四位。OSPI_CTL[6:5]。在每个已配置为四线模式的阶段中：

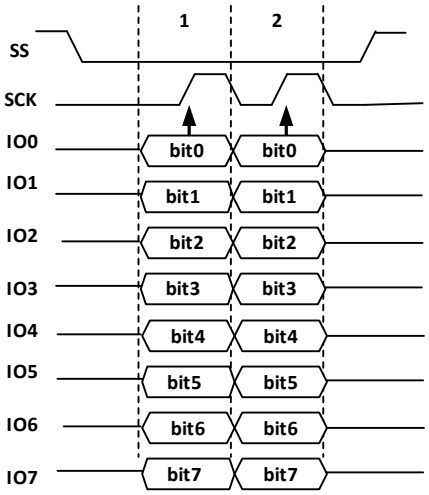
- IO0/IO1/IO2/IO3 在数据阶段进行读取操作时均处于高阻态（输入），在其他情况下为输出

下图为 OSPI 四线模式时序图（MSB）：



4.4. 八线 SDR 模式

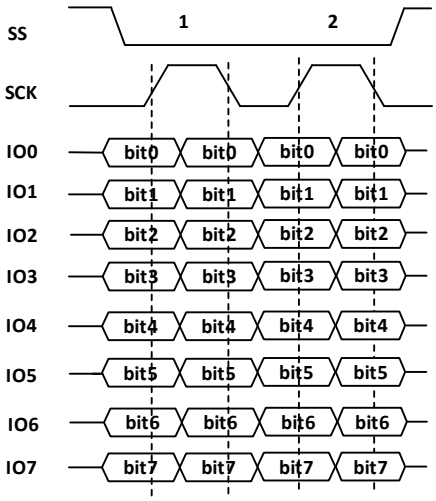
通过将（OSPI_CTL[6:5]）X_MODE 字段设置为 11 来选择八线模式。默认情况下 DTRM 位（OSPI_CTL[8]）为 0，OSPI 在单倍数据速率（SDR）模式下工作。在 SDR 模式下，当 OSPI 驱动 IO0、 IO1、 IO2、 IO3、IO4、IO5、IO6、IO7 信号时，这些信号仅在 CLK 的上降沿发生转变。在 SDR 模式下接收数据时，默认情况下（SSHIFT = 0 时），将使用 CLK 后续的上升沿对信号进行采样。下图为 OSPI 八线 SDR 模式时序图（MSB）：



4.5. 8 线 DTR 模式

通过将（OSPI_CTL[6:5]）X_MODE 字段设置为 11 来选择八线模式。通过将 DTRM 位（OSPI_CTL[8]）置 1，使 OSPI 在双倍数据速率（DTR）模式下工作。在 DTR 模式下，当 OSPI 驱动 IO0、 IO1、 IO2、 IO3、IO4、IO5、IO6、IO7 信号时，将在 CLK 的每个上升沿和下降沿发送 1 位。在 DTR 模式下接收数据时，默认情况下（SSHIFT = 0 时），将使用 CLK 后续的上升沿和下降沿对信号进行采样。下图为 OSPI 八线 DTR 模式时

序图（MSB）：


SS
1
2
SCK
IO0
IO1
IO2
IO3
IO4
IO5
IO6
IO7
bit0
bit1
bit2
bit3
bit4
bit5
bit6
bit7

5. 常见问题汇总

5.1. 采样延迟

通过设置（OSPI_RX_CTL[31:28]）SSHIFT 来控制是否延迟采集数据。最大支持推迟 7 个 HCLK 周期采集数据。

例如：

当 SSHIFT=0 时，即主机不延迟采集数据。

当 SSHIFT=1 时，即主机推迟 1 个 HCLK 周期开始采集数据。

当 SSHIFT=13 时，即主机推迟 7 个 HCLK 周期开始采集数据。

5.2. 输出延迟

仅在八线 DTR 模式下有效。通过设置（OSPI_TX_CTL[17:16]）OUTDLY 来控制是否延迟输出。最大支持延迟 2 个 HCLK 周期输出。

例如：

当 OUTDLY=0 时，即主机输出无延迟。

当 OUTDLY=1 时，即主机延迟 0.5 个 HCLK 周期输出。

当 OUTDLY=3 时，即主机延迟 2 个 HCLK 周期输出。

5.3. 叠封 PSRAM 的 IO 映射

注意只有部分型号叠封了 PSRAM，请仔细对照数据手册。

OSPI2	引脚名称	复用
OSPI_IO0	PP6	AF12
OSPI_IO1	PP7	AF12
OSPI_IO2	PP8	AF12
OSPI_IO3	P89	AF12
OSPI_IO4	PQ10	AF12
OSPI_IO5	PQ11	AF12
OSPI_IO6	PQ12	AF12
OSPI_IO7	PQ13	AF12
OSPI_NCS	PQ9	AF12
OSPI_CLK	PM14	AF12
OSPI_DQS	PQ15	AF12
OSPI_NRST	PN0	GPIO

6. 版本历史

版本	日期	作者	描述
V1.0	2025-3-20	Aisinochip	初始版

联系我们

公司：上海航芯电子科技股份有限公司

地址：上海市闵行区合川路 2570 号科技绿洲三期 2 号楼 702 室

邮编：200241

电话：+86-21-6125 9080

传真：+86-21-6125 9080-830

Email： service@AisinoChip.com

Website： www.AisinoChip.com