Rockchip RT-Thread SPI

文件标识：RK-KF-YF-093

发布版本：V2.3.0

日期：2024-11-12

文件密级：□绝密 □秘密 □内部资料 ■公开

免责声明

本文档按“现状”提供，瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文主要描述了 ROCKCHIP RT-Thread SPI 驱动的使用方法。

产品版本

芯片名称	内核版本
所有使用 RK RT-Thread SDK 的芯片产品	RT-Thread

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	赵仪峰	2019-07-13	初始发布
V1.0.1	赵仪峰	2020-05-27	修订格式
V2.0.0	林鼎强	2024-03-04	更新 RK2118 支持
V2.1.0	林鼎强	2024-04-26	添加 SPI slave 须知、常见问题
V2.2.0	林鼎强	2024-06-26	添加延时采样时钟配置方案
V2.3.0	林旭辉	2024-11-12	更新 RK3506 支持

目录

1. Rockchip SPI 功能特点

1.1 SPI 控制器特性

SPI（Serial Peripheral Interface）

- SPI master 支持 SPI mode 0/1/2/3，SPI slave 支持 SPI mode 0/1/2/3，SPI slave 推荐使用 SPI mode 0 方案（支持最高速率水准）
- 通常支持 2 个片选
- 支持 8 bits 和 16 bits 传输
- 支持中断传输模式和 DMA 传输模式
- 支持 FIFO 深度 32 级或 64 级
- 支持 SPI master mode 模式下的数据采样时钟 RSD 可配
- 支持 slave mode，部分芯片支持 slave mode 外部时钟采样
- 支持 lsb/msb 可配
- 支持大小端可配
- 部分芯片支持 cs inactive 特性，检测 CS 释放信号，产生中断
- 支持 MOSI/MOSI 四线传输，master mode MOSI 输出/MISO 输入，slave mode MISO 输出/MOSI 输入

1.2 SPI 接口特性

SOC	Master Mode 接口最高速率	Slave Mode 接口最高速率	TX/RX fifo 长度	Rx Sample Delay 单位
RK2108	50MHz	16MHz	64	\
RK2118	50MHz	50MHz	64	5ns
RK3506	50MHz	50MHz	64	5ns

说明：

- 接口最高速率为理论速率，受设备走线 PCB 质量影响，以实测为准
- 部分平台由于 PLL 策略原因无法准确分频到上限值，实际以最大分频值为准
- Rx Sample Delay 详细参考 “延时采样时钟配置方案”

1.3 SPI slave 须知

1.3.1 SPI2APB 说明

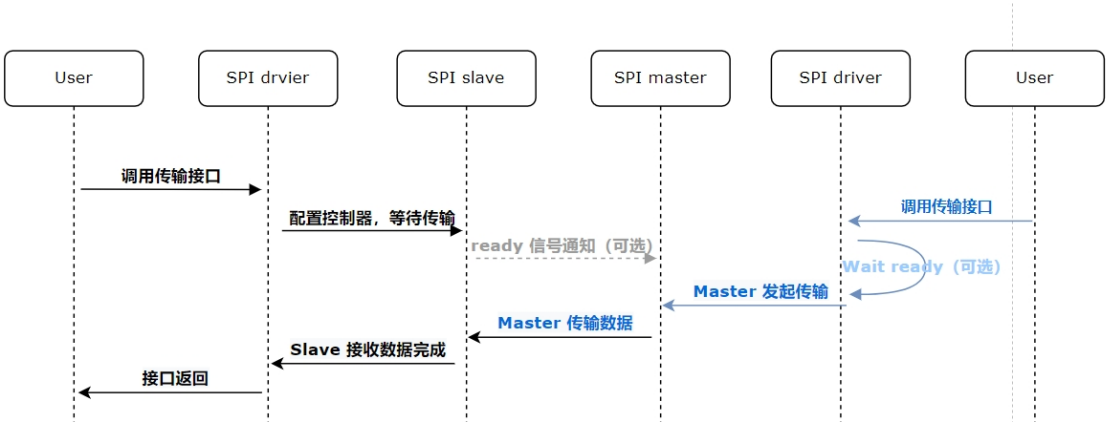
该文档为 RK SPI 模块专用文档，和 RK SPI2APB 为不同 IP，简单差别如下：

- RK SPI 支持 master/slave mode，且 slave mode 无数据传输格式要求
- RK SPI2APB 仅支持 slave mode，且 slave mode 要求固定传输格式

所以 SPI2APB 请参考 TRM SPI2APB 章节及 RK SPI2APB 开发文档《Rockchip_Developer_Guide_RT-Thread_SPI2APB》，例如 RK2118 SPI0（SPI2APB）

1.3.2 传输流程说明

RK SPI slave mode 基础传输流程为：



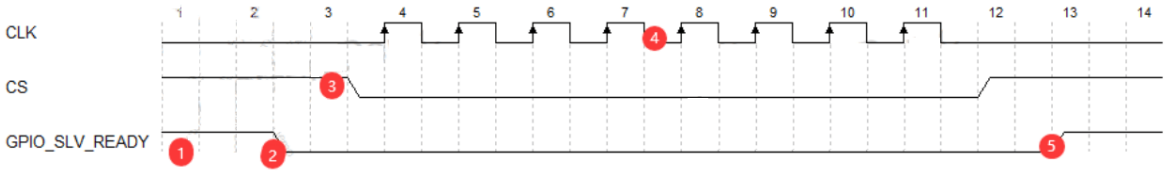
Slave 端：

- 用户层调用传输接口：
  - 支持单工只读、只写，或者全双工传输，接口要求指定传输长度
- 等待传输（主端应匹配此行为）：
  - Tgap（max）延时，SPI 控制器完成寄存器配置后进入等待远端 master device 发起传输状态，且每次 SPI Slave 传输都要重新调用传输接口，两笔传输之间结合 RT-Thead 系统需关注间隔时间最大值 Tgap（max），且要求 master 传输至少延时该时长 Tgap（max），由于客户实际业务负载不同，无法从理论上提供推荐值，以实测为准
  - ready 信号方案（可选），可以参考“常见问题” 章节中的 “增加 SPI slave ready 信号” 说明，增加实时通知信号
- 接收数据：
  - 传输方式支持：
    - DMA 传输：当传输长度超过 TX/RX fifo 一半水准，为了避免传输抖动，则软件自动启用 DMA 传输方案
    - IRQ 传输：当传输长度小于等于 TX/RX fifo 一半水准，软件使用中断传输方案
- 接口返回：
  - 传输长度限制：
    - 定长传输（默认），过滤 cs 释放信号，仅在接收到接口设定的传输长度对应量的数据后才返回，否则阻塞
    - 不定长传输，开启后检测 cs 释放信号，当 cs 提前释放或在接收到接口设定的传输长度对应量的数据后才返回，否则阻塞，配置参考 “不定长传输” 章节说明

Master 端：

- 用户层调用传输接口
- 等待 ready 信号有效（可选）：
  - Tgap（max）延时，匹配从端对应行为，确保 Tgap 内不会有两笔 SPI 传输发起
  - ready 信号方案（可选），根据 slave 支持情况决定是否添加，软件应匹配设定为上升沿触发中断，置位 ready 信号有效，传输完成后，再置位无效
- 控制器发起传输

1.3.3 传输时序简介



说明：

- step 1：用户层调用传输接口
- step 2：ready 信号（可选）
- step 3：等待传输
- step 4：数据传输
- step 5：接口返回

1.3.4 不定长传输

定长传输简介

RK SDK 默认配置 SPI slave 为定长传输，即 slave 端传输接口设定传输长度后，只有在接收到对应长度的数据后接口才会返回，否则会一直阻塞等待传输完成。

不定长传输简介

RK SDK 支持配置 SPI slave 为不定长传输，即 slave 端传输接口设定传输长度后，在接收到对应长度的数据或未接收全数据但出现 cs 释放的情况下后，接口返回已收发的数据量。

不定长传输基本原理为 SPI slave 支持并开启 cs inactive 特性，该特性支持硬件检测 cs 释放信号。

不定长传输支持情况

SOC	**cs inactive** 支持	**Flexible Length** 传输方案适配情况
RK2108	支持	未适配
RK2118	支持	已适配，BURST_SIZE=1 支持速率上限 25MHz，BURST_SIZE=2 支持速率上限 50MHz（最高）
RK3506	支持	未适配

不定长传输配置

参考 “控制器配置” 章节说明，除了开关外，还需关注 burst size 配置：

- burst size 越小，SPI 与总线的传输效率越低，可能会导致 slave 传输丢包，但对于 master 的传输 size 限制越小
    - 例如 RK2118，burst size 1，仅支持 master sclk 25MHz 以内，超过 25MHz 则有可能丢包，但对于 master 传输 size 无大小限定
- burst size 越大，SPI 与总线的传输效率越高，能支持更高的 IO 传输速率，但对于 master 的传输 size 限制越大
    - 例如 RK2118，burst size 2，支持 master sclk 50MHz，要求 master 传输 2 bytes aligned

2. 软件

2.1 代码路径

框架代码：

```
components/drivers/include/drivers/spi.h
components/drivers/spi/spi_core.c
components/drivers/spi/spi_dev.c
components/drivers/spi/qspi_core.c
```

驱动适配层：

```
bsp/rockchip-common/drivers/drv_spi.c
bsp/rockchip-common/drivers/drv_spi.h
```

测试命令，用户程序完全可以参照以下驱动：

```
bsp/rockchip-common/tests/spi_test.c
```

2.2 配置

2.2.1 控制器配置

打开配置，同时会生成/dev/spi0..2设备。

```
RT-Thread bsp drivers  --->
    RT-Thread rockchip "project" drivers  --->
        [*] Enable SPI
        [ ]   Enable SPI0 (SPI2APB)
        [*]   Enable SPI1
        [*]   Enable SPI2
        [*] Support stop SPI slave transmission when cs inactive
              Enable SPI Slave flexible length  --->
```

说明：

- RT_USING_SPI_SLAVE_FLEXIBLE_LENGTH：不定长传输开关，详细参考 “不定长传输” 章节说明
- RT_USING_SLAVE_DMA_RX_BURST_SIZE：不定长传输 DMA burst size，配置DMA突发大小，原理参考 “不定长传输” 章节说明，由于不定长传输默认使用 DMA 传输，所以要求上层传输时 buffer/length 符合 cache line size 对齐
- SPI0（SPI2APB）：SPI 控制器后标注 SPI2APB 字样的，为 SPI2APB 控制器，不同于 RK SPI 控制器，详细参考 《Rockchip_Developer_Guide_RT-Thread_SPI2APB》 文档

2.2.2 板级配置

确认已添加 SPI iomux 板级配置。

2.3 SPI 使用配置

参数说明

SPI 控制器作为 MASTER 时可以支持 0-50MHz（个别平台可以配置更高频率），作为 SLAVE 时固定支持 0-50MHz，且应配置为最高速率，能兼容不同速率MASTER 设备。

框架提供的配置函数 rt_spi_configure() 可以配置频率、模式和传输位宽等。

RT-Thread 标准框架支持可配参数：

```
#define RT_SPI_CPHA     (1<<0)                             /* bit[0]:CPHA, clock
phase */
#define RT_SPI_CPOL     (1<<1)                             /* bit[1]:CPOL, clock
polarity */

#define RT_SPI_LSB      (0<<2)                             /* bit[2]: 0-LSB */
#define RT_SPI_MSB      (1<<2)                             /* bit[2]: 1-MSB */

#define RT_SPI_MASTER   (0<<3)                             /* SPI master device
*/
#define RT_SPI_SLAVE    (1<<3)                             /* SPI slave device */

#define RT_SPI_MODE_0       (0 | 0)                        /* CPOL = 0, CPHA = 0
*/
#define RT_SPI_MODE_1       (0 | RT_SPI_CPHA)              /* CPOL = 0, CPHA = 1
*/
#define RT_SPI_MODE_2       (RT_SPI_CPOL | 0)              /* CPOL = 1, CPHA = 0
*/
#define RT_SPI_MODE_3       (RT_SPI_CPOL | RT_SPI_CPHA)    /* CPOL = 1, CPHA = 1
*/
```

RK 扩展 RT-Thread 框架支持可配参数：

```
#define RK_SPI_RESERVED_RSD_0     (0)
#define RK_SPI_RESERVED_RSD_1     (1)
#define RK_SPI_RESERVED_RSD_2     (2)
#define RK_SPI_RESERVED_RSD_3     (3)
#define RK_SPI_RESERVED_RSD_MASK  (3)
```

说明：

- RSD 配置详细参考 "延时采样时钟配置方案" 章节说明
- 要求索引 drv_spi.h 文件

配置代码示例

```
#include <drivers/spi.h>
#include "drv_spi.h"
```

```
struct rt_spi_configuration cfg;
struct rt_spi_device *spi_device = RT_NULL;

spi_device = (struct rt_spi_device *)rt_device_find("spi2_0");
if (spi_device ==  RT_NULL)
{
    rt_kprintf("Did not find device\n");
    return;
}

cfg.data_width = 8; /* 配置8bits传输模式 */
cfg.mode = RT_SPI_MASTER | RT_SPI_MSB | RT_SPI_MODE_0;
cfg.max_hz = 20 * 1000 * 1000; /* 配置频率 20Mhz */
spi_device->config.reserved = RT_SPI_RESERVED_RSD_1; /* 配置 rx sample delay 1 级
*/
rt_spi_configure(spi_device, &cfg); /* 配置 SPI*/
```

3. SPI测试

3.1 配置

```
RT-Thread bsp test case  --->
    [*] RT-Thread Common Test case  --->
        [*] Enable BSP Common TEST
        [*]  Enable BSP Common SPI TEST
```

3.2 测试命令

```
msh >spi_test
1. spi_test config <spi_device> <is_slave> <spi_mode> <is_msb> <speed>
<data_width>
2. spi_test read   <spi_device> <loops> <size>
3. spi_test write  <spi_device> <loops> <size>
4. spi_test duplex <spi_device> <loops> <size>
5. spi_test cs     <spi_device> <cs_state:1-take, 0-release>
6. spi_test bus    <spi_device> <bus_state:1-take, 0-release>
7. spi_test rate   <spi_device>
8. spi_test loop_thread   <spi_device> <loops> <size> <gap_ms> <random_pattern>
like:
        spi_test config spi1_0 0 3 0 24000000
        spi_test read   spi1_0 1 256
        spi_test write  spi1_0 1 256
        spi_test duplex spi1_0 1 256
        spi_test cs     spi1_0 1
        spi_test bus    spi1_0 1
        spi_test rate   spi1_0
        spi_test loop_thread   spi1_0 1 256 10 1
```

说明：

- random_pattern：0-递增序列，1-随机序列
- loops：相同操作循环次数

3.3 测试实例

3.3.1 回环通路

短接 MOSI/MISO 后使用全双工传输，配置 spi1_0、master_mode、spi_mode_3、lsb、24MHz

```
spi_test config spi1_0 0 3 0 24000000 8
spi_test duplex spi1_0 1 256
```

说明：

- 测试结果：回环测试失败会有 `test fail` 打印，原理为 MISO 没有接收到有效的 MOSI 数据，比对失败；如无异常打印，则测试成功
- 测试用途：建议使用该测试验证 SPI 控制器 master mode 工作正常

3.3.2 测速

配置 spi1_0、master_mode、spi_mode_3、lsb、50MHz

```
spi_test config spi1_0 0 3 0 50000000 8
spi_test rate   spi1_0
```

3.3.3 设备互联

spi 从设备守护进程

- 配置 spi2_0、slave_mode、spi_mode_3、lsb、50MHz
- 传输 10 loops、每次传输 256B、每次传输之间无延时、递增序列（要求对端也是递增序列）

```
spi_test config spi2_0 1 3 0 50000000 8
spi_test duplex_thread spi2_0 10 256 0 0
```

spi 主设备测试进程

- 配置 spi1_0、master_mode、spi_mode_3、lsb、50MHz
- 传输 10 loops、每次传输 256B、每次传输延时 10 ms 等待 slave ready、递增序列（要求对端也是递增序列）

```
spi_test config spi1_0 0 3 0 50000000 8
spi_test duplex_thread spi1_0 10 256 10 0
```

3.3.4 信号测试

测试要求：

- 互联 SPI 信号线，其中 MOSI 接 MOSI
- 共地
- 测试命令验证按照以下次序，先从后主

测试命令：

- spi 从设备守护进程
    - 配置 spi2_0、slave_mode、spi_mode_3、lsb、50MHz
    - 传输 100000000 loops、每次传输 4096B、每次传输之间无延时、随机序列（不做校验）

```
spi_test config spi2_0 1 3 0 50000000 8
spi_test duplex_thread spi2_0 100000000 4096 0 1
```

- spi 主设备测试进程
    - 配置 spi1_0、master_mode、spi_mode_3、lsb、50MHz
    - 传输 100000000 loops、每次传输 4096B、每次传输之间 10 ms 延时、随机序列（不做校验）

```
spi_test config spi1_0 0 3 0 50000000 8
spi_test duplex_thread spi1_0 100000000 4096 10 1
```

测试判断测试命令成功

- MSH 确认是否有两个 duplex 进程：

```
msh >list_thread
thread   pri  status      sp     stack size max used left tick  error
-------- ---  ------- ---------- ----------  ------  ---------- ---
spi_dupl   5  suspend 0x000000c4 0x00000400    50%   0x00000009 OK
spi_dupl   5  suspend 0x00000120 0x00000400    38%   0x00000009 OK
```

- 信号上确认 MOSI/MISO 同时有输出

4. 常见问题

4.1 SPI master mode 传输失败

建议参考 "SPI测试" 章节中的 “回环通路” 测试，如果测试失败，通常为 iomux 没有正确配置

4.2 SPI slave mode 传输失败

建议参考 “SPI测试”，配置控制器为 SPI mode 0，8 bits 传输 256Bytes，命令如下：

```
# slave
spi_test config spi2_0 1 0 0 50000000 8
spi_test read spi2_0 1 256

# master（根据实际平台做验证）
spi_test config spi2_0 0 0 0 50000000 8
spi_test write spi2_0 1 256
```

说明：

- master 发起传输前，正常 slave 传输会阻塞进入等待传输， RT-Thread 命令行阻塞，如果没有阻塞，则配置存在异常
- master 发起传输后，slave 依旧保持阻塞，常规调试流程：
  - 如果使用默认定长传输，未开启 “不定长传输” 配置，请确认 master 传输是否送出足够数据量，是否匹配 slave 端设定的传输长度
  - 互联设备是否已经共地
  - 示波器 master 设备测量 csn/clock 是否有有效信号
  - slave 硬件线路是否连接正确，线路上是否有 0Ω 电阻未贴片，可以直接通过参考 master 方式发起传输，同样使用示波器测量 csn/clock 信号，来确认接线是否正确
  - 如果没有测量到正确信号，通常为对应设备的 iomux 没有配置正确，请确认所配置的 iomux 和实际使用的 io 是否一致
    - 如果使用 IO 矩阵请确认是否存在多组 IO 同时配置为 SPI function IO，例如 RK2118 SPI1/2 RMIO
- master 发起传输后，slave 或 master 接收到错误的数据，常规调试流程：
  - 降低 master IO 速率看是否有优化，如果有优化，则高速下的信号质量无法满足要求
  - 确认数据是否存在规律：
    - 是否匹配 LSB/MSB 或者大小端配置问题
    - 是否仅接收到部分有效数据，怀疑两笔数据传输之间间隔过短，slave 一笔传输完成后还未成功重新配置，master 又发起一笔传输，建议增长两笔传输间隔，详细参考 “传输流程说明” 章节
    - master 端读数据异常，是否存在移位的现象，如果移位， 确认 master 端是否有采样延迟可调

4.3 增加 SPI slave ready 信号

参考以下代码自行增加 ready 信号控制逻辑：

```
diff --git a/bsp/rockchip/common/drivers/drv_spi.c
b/bsp/rockchip/common/drivers/drv_spi.c
index b73ada2fbf..c42b93890c 100644
--- a/bsp/rockchip/common/drivers/drv_spi.c
+++ b/bsp/rockchip/common/drivers/drv_spi.c
```

```
@@ -555,6 +555,10 @@ static rt_uint32_t rockchip_spi_xfer(struct rt_spi_device
*device, struct rt_spi
     }
     else
     {
+        HAL_GPIO_SetPinLevel(GPIOn, pin, GPIO_LOW);
+        HAL_DelayUs(1);
+        HAL_GPIO_SetPinLevel(GPIOn, pin, GPIO_HIGH);
+
         /* Use IT mode for slave while less fifo length. */
         if (spi->dma && HAL_SPI_CanDma(pSPI))
         {
@@ -585,6 +589,8 @@ static rt_uint32_t rockchip_spi_xfer(struct rt_spi_device
*device, struct rt_spi
                 rockchip_spi_wait_idle(spi, true);
             }
         }
+
+        HAL_GPIO_SetPinLevel(GPIOn, pin, GPIO_LOW);
     }

 complete:
```

4.4 SPI Debug 宏开关

SPI master mode debug 宏开关：

```
diff --git a/bsp/rockchip/common/drivers/drv_spi.c
b/bsp/rockchip/common/drivers/drv_spi.c
index 3562abffda..76e7268baf 100644
--- a/bsp/rockchip/common/drivers/drv_spi.c
+++ b/bsp/rockchip/common/drivers/drv_spi.c
@@ -39,7 +39,7 @@
 #include "dma.h"
 #include "hal_bsp.h"

-#define SPI_DEBUG 0
+#define SPI_DEBUG 1

 #if SPI_DEBUG
 #define spi_dbg(dev, fmt, ...) \
```

SPI master mode debug 宏开关：

```
diff --git a/bsp/rockchip/common/drivers/drv_spi.c
b/bsp/rockchip/common/drivers/drv_spi.c
index 3562abffda..76e7268baf 100644
--- a/bsp/rockchip/common/drivers/drv_spi.c
+++ b/bsp/rockchip/common/drivers/drv_spi.c
@@ -39,7 +39,7 @@
 #include "dma.h"
 #include "hal_bsp.h"

-#define SPI_DEBUG 0
+#define SPI_DEBUG 2

 #if SPI_DEBUG
 #define spi_dbg(dev, fmt, ...) \
```

4.5 SPI 传输模式说明

Master mode，当满足以下条件时使用 DMA 传输，否则使用 CPU 传输：

```
spi->dma # 支持 DMA
HAL_IS_CACHELINE_ALIGNED(pSPI->pRxBuffer)
HAL_IS_CACHELINE_ALIGNED(pSPI->pTxBuffer)
HAL_IS_CACHELINE_ALIGNED(pSPI->len)
pSPI->len > HAL_SPI_DMA_SIZE_MIN # HAL_SPI_DMA_SIZE_MIN 默认值为 512
```

Slave mode：

- 定长传输（默认），当满足以下条件时使用 DMA 传输，否则使用 IRQ 传输，而且建议 slave mode 使用 DMA 传输方案，且为了更好的 DMA burst size，建议传输长度 对齐 8Bytes 传输：

```
spi->dma # 支持 DMA
HAL_IS_CACHELINE_ALIGNED(pSPI->pRxBuffer)
HAL_IS_CACHELINE_ALIGNED(pSPI->pTxBuffer)
HAL_IS_CACHELINE_ALIGNED(pSPI->len)
pSPI->len > HAL_SPI_FIFO_LENGTH / 2 # HAL_SPI_FIFO_LENGTH 参考 “SPI 控制器特性”
```

- 不定长传输（宏开关配置），仅支持 DMA 传输，所以要求：

```
spi->dma # 支持 DMA
HAL_IS_CACHELINE_ALIGNED(pSPI->pRxBuffer)
HAL_IS_CACHELINE_ALIGNED(pSPI->pTxBuffer)
HAL_IS_CACHELINE_ALIGNED(pSPI->len) # HAL_SPI_DMA_SIZE_MIN 默认值为 512
```

4.6 延时采样时钟配置方案

对于 SPI io 速率较高的情形，正常 SPI mode 可能依旧无法匹配外接器件输出延时，RK SPI master read 可能无法采到有效数据，需要启用 SPI rsd 逻辑来延迟采样时钟。

RK SPI rsd（read sample delay）控制逻辑有以下特性：

- 可配值为 0，1，2，3
- 延时单位为 1 spi_clk 周期，即控制器工作时钟周期，不同芯片对应值参考 "SPI 控制器特性“ 章节记录