



■ **PHY622X**

Peripheral Application

Note

Version 1.0

Author: phyplusinc

Security: Public

Date: 2021.01

PhyPlus

Copyright © 2020 Phyplus Microelectronics Limited All rights reserved.
Reproduction in whole or in part is prohibited without the prior written permission of the copyright holder.



Revision History

Revision	Author	Date	Description
V1.0		2021.01.25	This document is used for 6220/6250/6222/6252.

目录

1	WATCHDOG	1
1.1	WATCHDOG 概述	1
1.2	Watchdog 硬件	1
1.3	WATCHDOG 示例代码	1
2	TIMER	1
2.1	TIMER 概述	1
2.2	TIMER 硬件	1
2.3	TIMER 示例代码	2
3	PWM	2
3.1	PWM 概述	2
3.2	PWM 硬件	2
3.3	PWM 示例代码	3
4	UART	4
4.1	UART 概述	4
4.2	UART 硬件	4
4.3	UART 示例代码	4
5	SPI	5
5.1	SPI 概述	5
5.2	SPI 硬件	7
5.3	SPI 支持速率	8

5.4	SPI 示例代码	8
6	I2C	8
6.1	I2C 概述	8
6.2	I2C 硬件	8
6.3	I2C 示例代码	9
7	KSCAN	9
7.1	KSCAN 概述	9
7.2	KSCAN 硬件	9
7.3	KSCAN 示例代码	9

图表目录

图 1:	PWM 示意图	2
图 2:	UP MODE, UP AND DOWN MODE	3
表 1:	PWM 占空比计算	3
表 2:	PWM 频率计算	3
图 3:	UART 帧格式	4
图 4:	SPI 示意图	5
图 5:	CPOL=0 CPHA=0	6
图 6:	CPOL=0 CPHA=1	6
图 7:	CPOL=1 CPHA=0	7
图 8:	CPOL=1 CPHA=1	7
图 9:	I2C 示意图	8

1 WATCHDOG

1.1 WATCHDOG 概述

在单片机构成的微系统中，当代码书写不健壮，或者是运行环境受到外界硬件干扰，造成各种寄存器和内存的数据混乱，会导致程序跑飞或是系统陷入死循环。此时程序的正常运行被打断，正常逻辑无法继续执行，导致整个系统的陷入停滞状态，发生不可预料的后果。

为了解决上述问题，看门狗应运而生。看门狗，又叫 WATCHDOG，从本质上来讲就是一个定时器电路，一般有一个输入和一个输出，其中输入叫做喂狗，输出一般连接到另外一个部分的复位端。看门狗的功能是定期的查看芯片内部的情况，一旦发生错误就向芯片发出重启信号。

1.2 Watchdog 硬件

- WATCHDOG 时钟为 32.768Khz，可以选择 RC 32k 或 XTAL 32k。
- WATCHDOG 喂狗周期可选择 2S、4S、8S、16S、32S、64S、128S、256S。
- WATCHDOG 使用方式可以轮询和中断。当选择轮询方式时，超过喂狗周期未喂狗，系统将产生复位。当选择中断方式时，当超过喂狗周期未喂狗，系统会在第一个喂狗周期到达时产生 WATCHDOG 中断，此中断函数内可以喂狗，如果不喂狗，系统将在第二个喂狗周期达到时产生复位。
- 当系统休眠时，WATCHDOG 信息会丢失，唤醒后需要重新配置。
- 6220/6250 不支持中断方式喂狗。

1.3 WATCHDOG 示例代码

见 SDK。

2 TIMER

2.1 TIMER 概述

TIMER 是微系统中的标配，为应用提供精准的计时机制。

2.2 TIMER 硬件

- 系统共有 6 个硬件 TIMER，其中 4 个已经被协议栈、OSAL 调度器等软件资源所使用，其余 2 个供应用使用。
- 时钟源固定 4MHz，硬件不可分频。驱动中为了计算方便，软件将其 4 分频。
- 位宽为 32bit，即最大计数值为 0xFFFFFFFF。
- 支持中断方式和非中断方式。

- 支持 free-running mode 和 user-defined count mode。前者减到 0 后, 自动加载 0xFFFFFFFF。后者减到 0 后, 自动加载用户预先配置的值。驱动中用的是后者。
- Timer 中断产生后不会自动停止, 而是自动装载数据进行下一次计数, 如此反复。如需要将其停止, 可手动将其关闭。
- 当系统休眠时, TIMER 信息会丢失, 唤醒后需要重新配置。

2.3 TIMER 示例代码

见 SDK。

3 PWM

3.1 PWM 概述

PWM 表示脉冲宽度调制, 它是一系列脉冲, 这些脉冲将以方波的形式出现。也就是说, 在任何给定的时间点, 波型要么是高电平或者是低电平。

PWM 有两个参数:

- 占空比 = 接通时间 / (接通时间 + 断开时间)
- 频率 = 1 / 总持续时间, 总持续时间 = 接通时间 + 断开时间

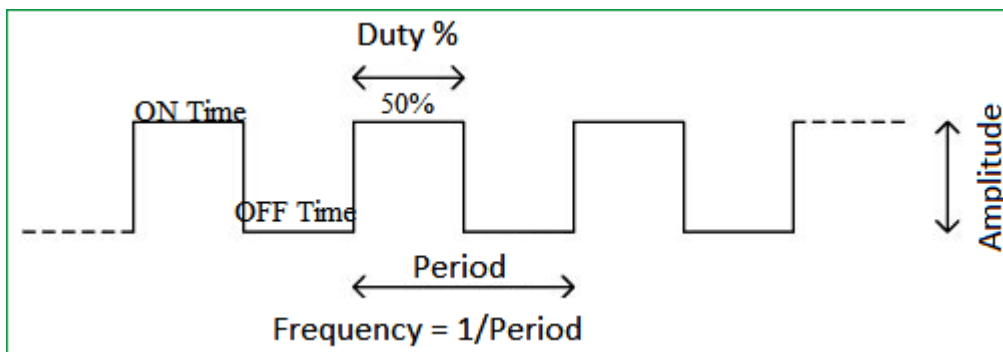


图 1: PWM 示意图

3.2 PWM 硬件

- 支持 6 路 PWM。
- 时钟源 16Mhz, 每路 PWM 支持的分频有 1、2、4、8、16、32、64、128。
- 当系统休眠时, PWM 信息会丢失, 唤醒后需要重新配置。
- 所有可 FMUX 的 IO 都可以复用为 PWM。
- 支持 UP 模式和 UP AND DOWN 模式。前者支持占空比 0~100%; 后者不支持占空比 0% 和 100%, 需要 GPIO 输出高低来辅助实现。

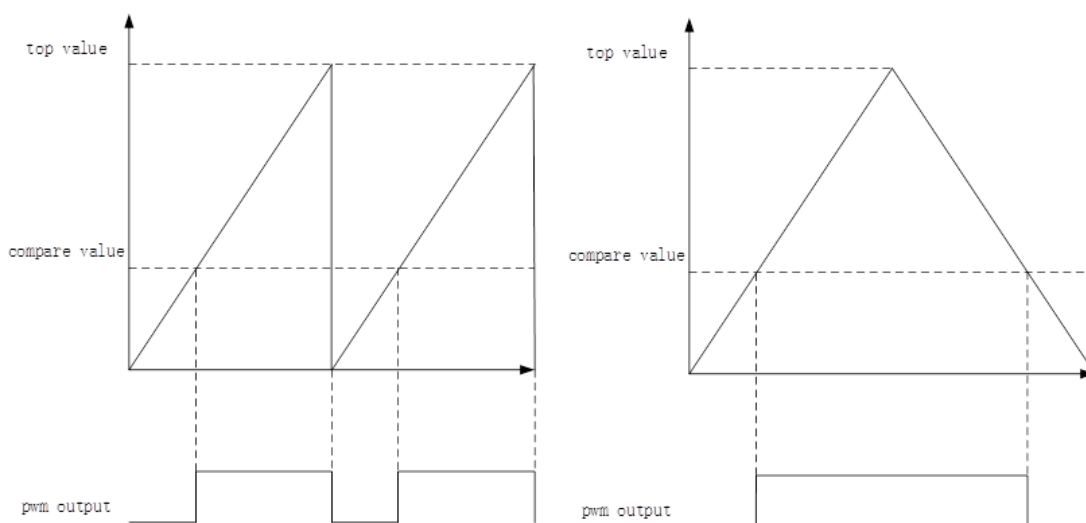


图 2: UP MODE, UP AND DOWN MODE

PWM 占空比的计算:

	POLARITY_FALLING	POLARITY_RISING
UP MODE	$(CMP_VAL+1)/(TOP_VAL+1)$	$1-((CMP_VAL+1)/(TOP_VAL+1))$
UP AND DOWN MODE	CMP_VAL/TOP_VAL	$1- (CMP_VAL/TOP_VAL)$

表 1: PWM 占空比计算

PWM 占空比注意事项:

- UP MODE 支持占空比范围: 0%~100%, 包括 0%和 100%。以 POLARITY_FALLING 为例, 当 $CMP_VAL=0$ 时, 占空比为 0%; 当 $CMP_VAL=TOP_VAL$ 时, 占空比为 100%。
- UP AND DOWN MODE 支持占空比范围: (0%~100%), 不包括 0%和 100%。

PWM 频率的计算:

	N 分频(N=1,2,4,8,16,32,64,128)
UP MODE	$16/N/(TOP_VAL+1)$
UP AND DOWN MODE	$8/N/TOP_VAL$

注意事项:

表 2: PWM 频率计算

PWM 注意事项:

- UP MODE 支持频率范围: 62.5KHz~8MHz。支持分辨率: 0 和 $2/65536\sim65536/65536$ 。
- UP AND DOWN MODE 支持频率范围: 31.25KHz~4MHz。支持分辨率: $0/65535\sim65534/65535$ 。

3.3 PWM 示例代码

见 SDK。

4 UART

4.1 UART 概述

通用异步收发传输器 (Universal Asynchronous Receiver/Transmitter)，通常称作 UART。在 UART 通信中，两个 UART 直接相互通信。发送 UART 将来自 CPU 等控制设备的并行数据转换为串行形式，并将其串行发送到接收 UART，接收 UART 然后将串行数据转换回接收设备的并行数据。在两个 UART 之间传输数据只需要两根线。

UART 以异步方式发送数据，这意味着没有时钟信号将发送 UART 的位输出与接收 UART 的位采样同步。发送 UART 不是时钟信号，而是将开始和停止位添加到正在传输的数据包中。这些位定义数据包的开始和结束，因此接收 UART 知道何时开始读取位。

当接收 UART 检测到起始位时，它开始以称为波特率的特定频率读取输入位。波特率是数据传输速度的度量，以每秒位数 (bps) 表示。两个 UART 必须以大致相同的波特率运行。

UART 传输的数据被组织成数据包。每个数据包包含 1 个起始位，5 到 9 个数据位（取决于 UART），可选的奇偶校验位以及 1 或 2 个停止位。

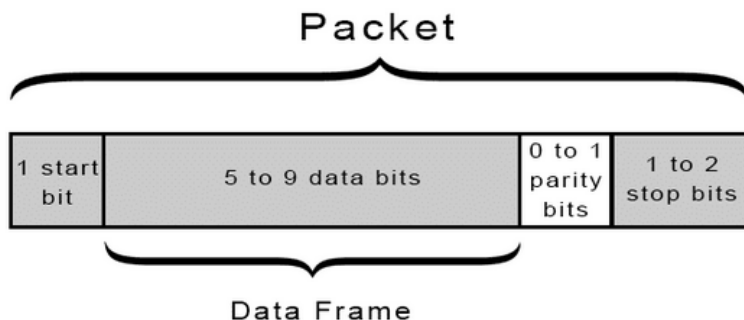


图 3: UART 帧格式

4.2 UART 硬件

- 支持 2 路 UART。
- 硬件 TX FIFO 深度 16 字节，硬件 RX FIFO 深度 16 字节。
- PCLK 时钟等于 HCLK，可分频，不建议分频。
- 当系统休眠时，UART 信息会丢失，唤醒后需要重新配置。
- 所有可 FMUX 的 IO 都可以复用为 UART。
- 系统日志打印默认使用 UART0(P9、P10)，可通过 DEBUG_INFO 配置关闭或打开。
- 假设当前系统主频为 hclk 且不分频，需要的波特率为 baud，实际硬件配置的寄存器为 divisor = (hclk) / (16 * baud)。当丢失的小数部分大于 2% 时，此波特率不支持会乱码。比如：系统时钟为 48M，波特率分别为 115200、921600、1000000，需要配置的寄存器分别是 26.041666666666667、3.2552083333333333、3，误差分别是 0.16%、7.84%、0%，因此此时支持 115200 和 1000000，不支持 921600。

4.3 UART 示例代码

见 SDK。

5 SPI

5.1 SPI 概述

SPI，是英语 **S**erial **P**eripheral **I**nterface 的缩写，顾名思义就是串行外围设备接口。是一种同步、全双工、主从式接口。来自主机或从机的数据在时钟上升沿或下降沿同步。主机和从机可以同时传输数据。

在芯片的管脚上只占用四根线，节约了芯片的管脚，同时为 PCB 的布局上节省空间，提供方便，正是出于这种简单易用的特性，现在越来越多的芯片集成了这种通信协议。

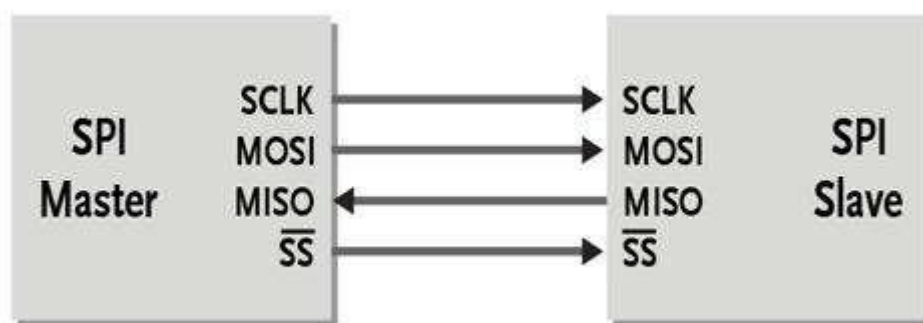


图 4: SPI 示意图

产生时钟信号的器件称为主机。主机和从机之间传输的数据与主机产生的时钟同步。4 线 SPI 器件有四个信号：

- 时钟(SPI CLK, SCLK)
- 片选(CS)
- 主机输出、从机输入(MOSI)
- 主机输入、从机输出(MISO)

SPI 模块为了和外设进行数据交换，根据外设工作要求，其输出串行同步时钟极性和相位可以进行配置，时钟极性（CPOL）对传输协议没有重大的影响。

- CPOL: 时钟极性选择，为 0 时 SPI 总线空闲为低电平，为 1 时 SPI 总线空闲为高电平。
- CPHA: 时钟相位选择，为 0 时在 SCK 第一个跳变沿采样，为 1 时在 SCK 第二个跳变沿采样。

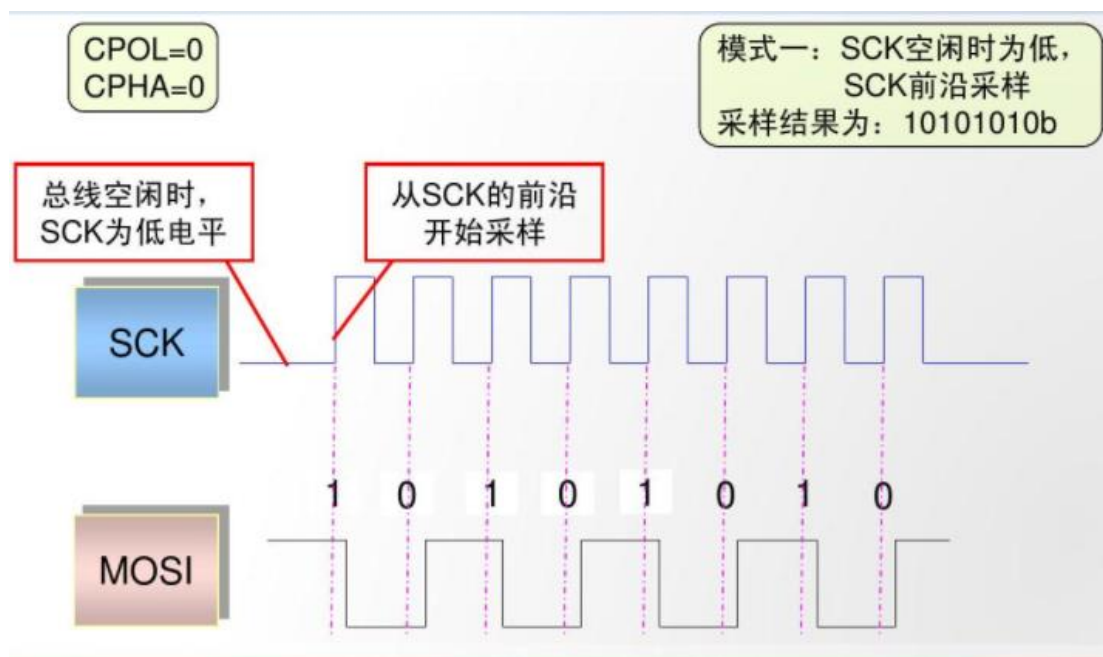


图 5: CPOL=0 CPHA=0

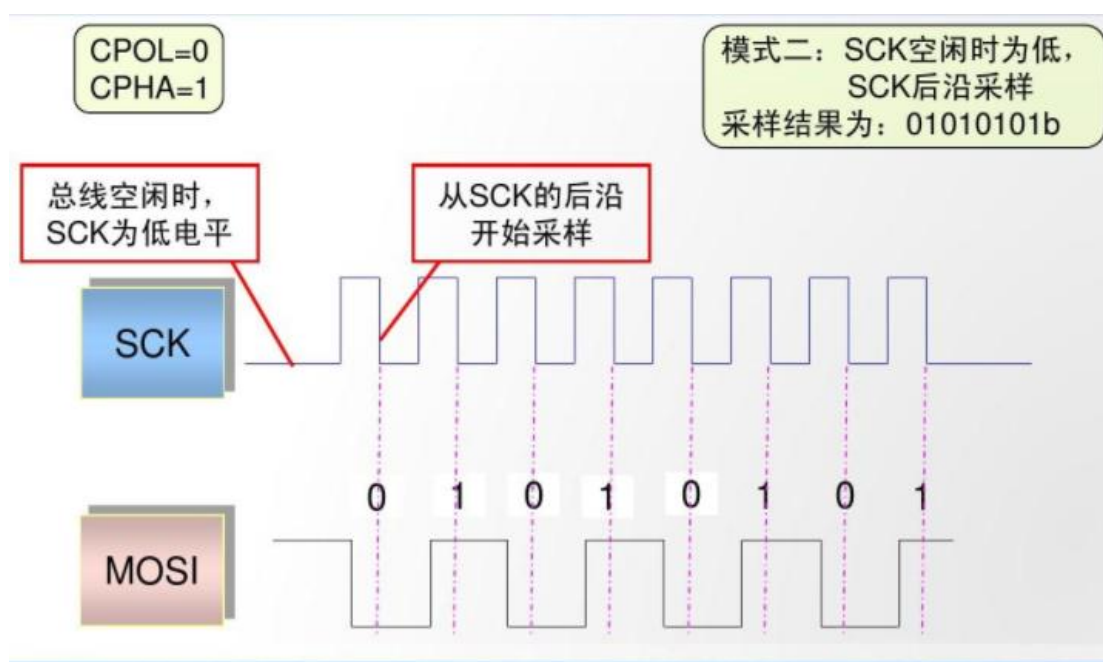


图 6: CPOL=0 CPHA=1

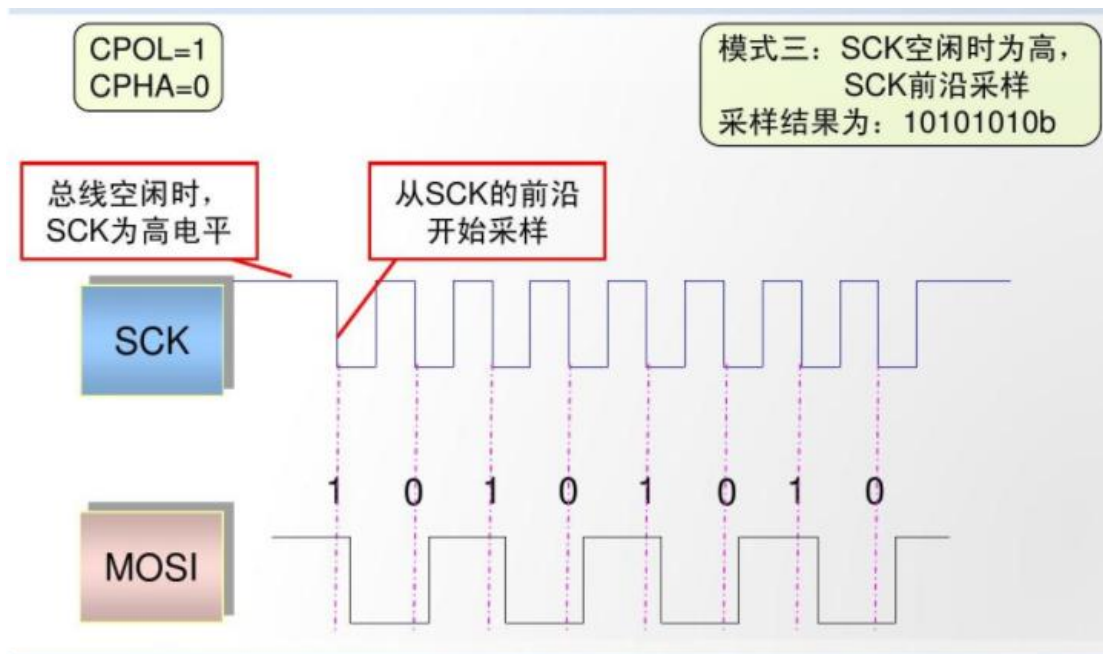


图 7：CPOL=1 CPHA=0

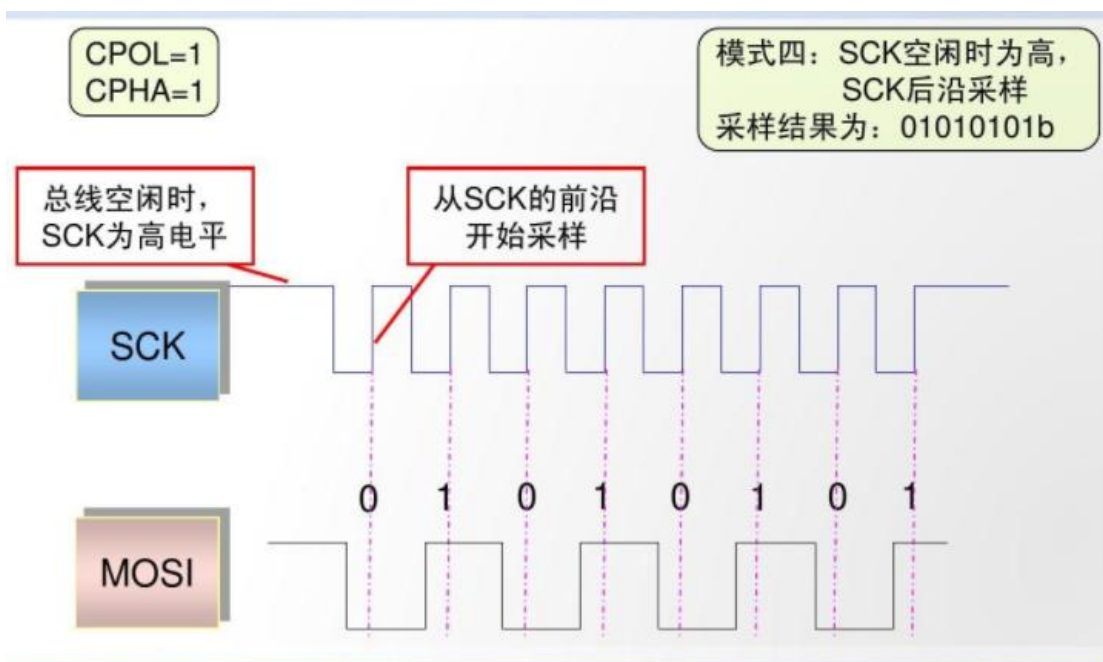


图 8：CPOL=1 CPHA=1

5.2 SPI 硬件

- 硬件支持 2 路 SPI，可配置 MASTER 或 SLAVE。
- 硬件 TX FIFO 和 RX FIFO 深度是 8，单位软件可配，范围为 4bit~16bit。
- 时钟等于 HCLK，可分频，不建议分频。

- 当系统休眠时，SPI 信息会丢失，唤醒后需要重新配置。
- 所有可 FMUX 的 IO 都可以复用为 SPI。
- 当使用 SPI 发送数据时，可以选择自动或手动控制 cs 的高低，可以选择是否使用中断。
所谓手动是通过将 IO 设置为 GPIO 并将其输出高低。spi_Cfg_t 中的 force_cs 为 TRUE 时选择手动模式，spi_Cfg_t 中的 int_mode 为 TRUE 时使用中断方式。

5.3 SPI 支持速率

SPI 频率有如下注意事项，其中 F_{ssi_clk} 为 pclk， F_{sclk_in} 为 spi clock。

- SPI Master: $F_{ssi_clk} \geq 2 \times (\text{maximum } F_{sclk_out})$
- SPI Slave (receive only): $F_{ssi_clk} \geq 6 \times (\text{maximum } F_{sclk_in})$
- SPI Slave: $F_{ssi_clk} \geq 8 \times (\text{maximum } F_{sclk_in})$

5.4 SPI 示例代码

见 SDK。

6 I2C

6.1 I2C 概述

I2C Bus(Inter-Integrated Circuit Bus) 最早是由 Philips 半导体（现被 NXP 收购）开发的两线时串行总线，常用于微控制器与外设之间的连接。

I2C 仅需两根线就可以支持一主多从或者多主连接，I2C 使用两个双向开漏线，配合上拉电阻进行连接。

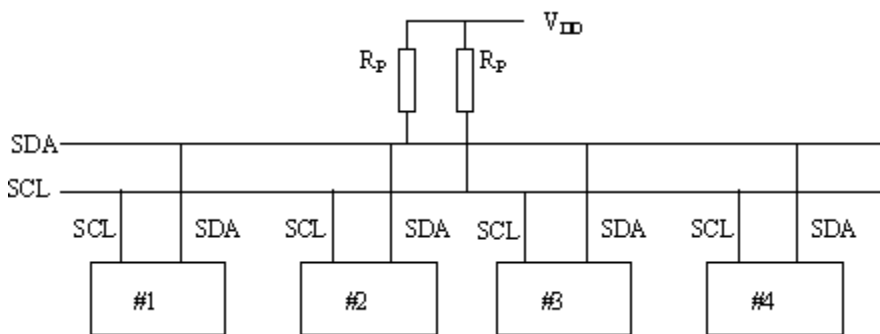


图 9: I2C 示意图

6.2 I2C 硬件

- 硬件支持 2 路 I2C，可配置 MASTER 或 SLAVE。
- 硬件 TX FIFO 深度 8 字节，硬件 RX FIFO 深度 8 字节。
- PCLK 时钟等于 HCLK，可分频，不建议分频。
- 当系统休眠时，I2C 信息会丢失，唤醒后需要重新配置。
- 所有可 fmux 的 io 都可以复用为 I2C。
- I2C 使用时需要接上拉电阻，比如 2.2K 或 4.7K。

6.3 I2C 示例代码

见 SDK。

7 KSCAN

7.1 KSCAN 概述

KSCAN，也称矩阵键盘，当应用按键较多而 io 较少时，可以使用 KSCAN。M 行 N 列的 kscan 可以实现 $M*N$ 个矩阵按键。

KSCAN 分行和列，行输入列输出。当无按键时，每列上的每行都处于 idle 状态键值为 0，当有按键按下时，该列对应的该行 bit 会置位键值为 1，同时会触发中断。

7.2 KSCAN 硬件

- 11 个 io 可以配置为行。12 个 io 可以配置为列，不建议使用 P16P17。对应的 io 和索引见 kscan.h。
- 当系统休眠时，KSCAN 信息会丢失，唤醒后需要重新配置。

7.3 KSCAN 示例代码

见 SDK。