# PRBMD02 Application Note

Peripheral application note

## Disclaimer

# 1. Watchdog

## 1.1. Watchdog description

In a microsystem composed of single-chip microcomputers, when the code writing is not robust, or the operating environment is interfered by external hardware, resulting in confusion of data in various registers and memory, the program will run away or the system will fall into an infinite loop. At this time, the normal operation of the program is interrupted, and the normal logic cannot continue to be executed, resulting in the stagnation of the entire system and unforeseen consequences. In order to solve the above problems, the watchdog came into being. Watchdog, also called WATCHDOG, is essentially a timer circuit, generally has one input and one output, the input is called feed dog, and the output is generally connected to the reset terminal of another part. The function of the watchdog is to regularly check the internal conditions of the chip, and once an error occurs, it will send a restart signal to the chip.

## 1.2. Watchdog hardware

- WATCHDOG clock is 32.768Khz, you can choose RC 32k or XTAL 32k.
- WATCHDOG dog feeding cycle can choose 2s, 4s, 8s, 16s, 32s, 64s, 128s, 256s.
- WATCHDOG usage can poll and interrupt. When the polling mode is selected, the system will reset if the dog is not fed after the dog-feeding period. When the interrupt mode is selected, when the dog is not fed after the dog-feeding period, the system will generate a WATCHDOG interrupt when the first dog-feeding period arrives. The dog can be fed in this interrupt function. If the dog is not fed, the system will feed the dog in the second feeding period. A reset occurs when a dog cycle is reached.
- When the system sleeps, the WATCHDOG information will be lost and needs to be reconfigured after waking up.
- PRBMD02 does not support interrupt feeding the dog.

### 1.2.1. Watchdog sample code

Refers to SDK

# 2. Timer

## 2.1. Timer description

TIMER is standard in the microsystem and provides an accurate timing mechanism for the application.

## 2.2. Timer hardware

- The system has a total of 6 hardware TIMERs, 4 of which have been used by software resources such as the protocol stack and OSAL scheduler, and the remaining 2 are used by applications.
- The clock source is fixed at 4MHz and cannot be divided by hardware. In the driver, for the convenience of calculation, the software divides it by 4.
- The bit width is 32bit, that is, the maximum count value is 0xFFFFFFFF.
- Both interrupt mode and non-interrupt mode are supported.
- Support **free-runningmode** and **user-definedcountmode**. After the former is reduced to 0, 0xFFFFFFFF is automatically loaded. After the latter is reduced to 0, the user-preconfigured value is automatically loaded. The latter is used in the driver.
- After the Timer interrupt is generated, it will not stop automatically, but will automatically load data for the next count, and so on. To stop it, turn it off manually.
- When the system sleeps, the TIMER information will be lost and needs to be reconfigured after waking up.

## 2.3. Timer sample code

refers to SDK

## 3. PWM

### 3.1. PWM description

PWM stands for Pulse Width Modulation and it is a series of pulses that will appear as square waves. That is, at any given point in time, the waveform is either high or low.

PWM has two parameters:

- Duty Cycle = On Time / (On Time + Off Time)
- Frequency = 1 / Total Duration, Total Duration = On Time + Off Time

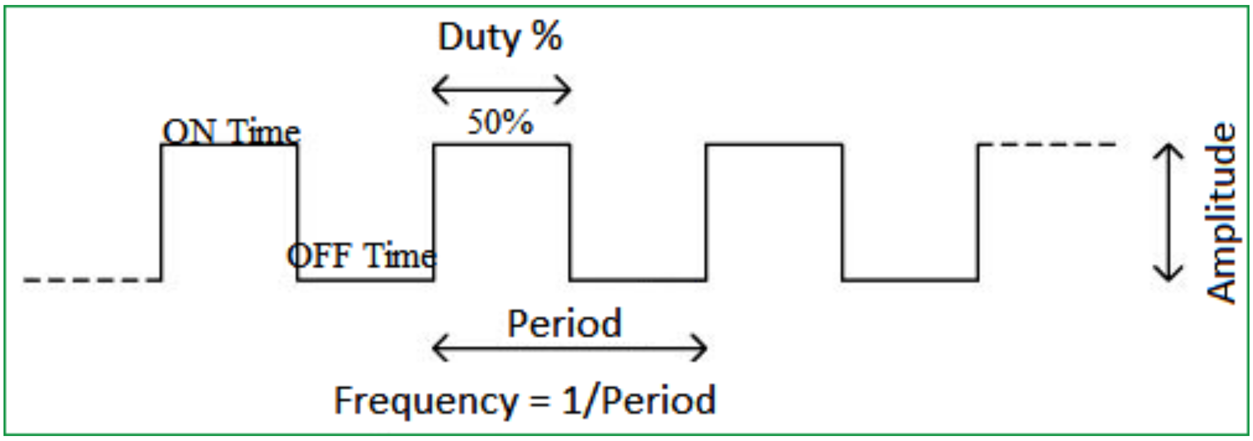

Fig 1: PWM schematic

### 3.2. PWM hardware

- Supports 6 channels PWM
- The clock source is 16Mhz, and the frequency divisions supported by each PWM are 1, 2, 4, 8, 16, 32, 64, and 128.
- When the system sleeps, the PWM information is lost and needs to be reconfigured after waking up.
- All FMUX-capable IOs can be multiplexed as PWM.
- Support UP mode and UPANDDOWN mode. The former supports a duty cycle of 0~100%; the latter does not support a duty cycle of 0% and 100%, and requires GPIO output levels to assist in implementation.

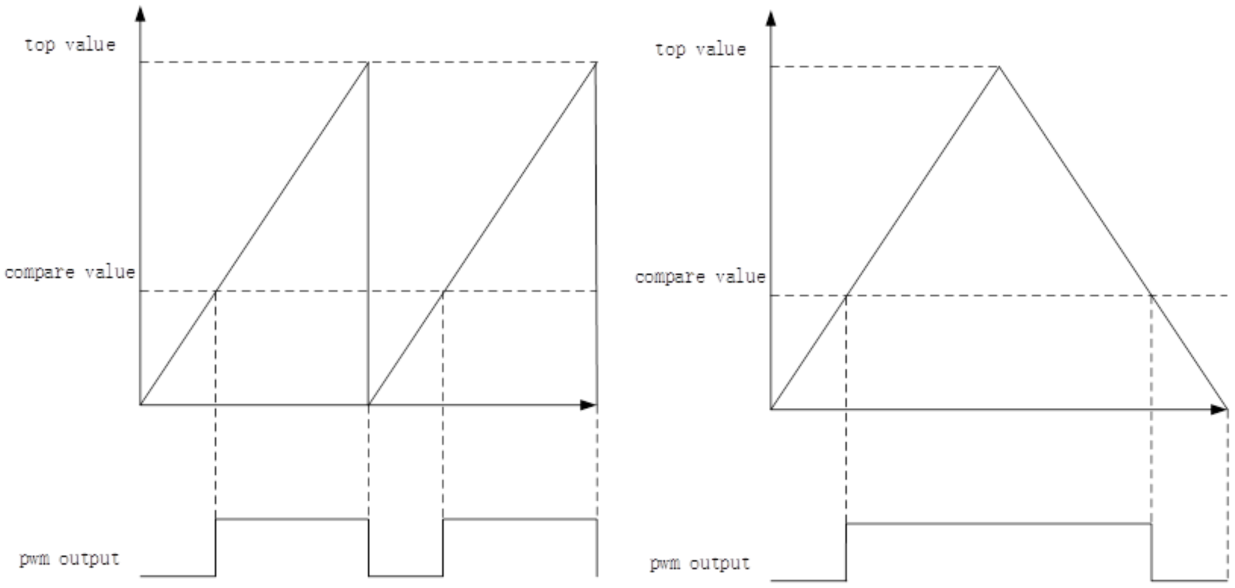

Fig 2:UP MODE, UP AND DOWN MODE

PWM duty-cycle calculation

| | 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) |

PWM Duty Cycle Notes:

- UPMODE supports duty cycle range: 0%~100%, including 0% and 100%. Taking POLARITY_FALLING as an example, when CMP_VAL=0, the duty cycle is 0%; when CMP_VAL=TOP_VAL, the duty cycle is 100%.
- UP AND DOWN MODE supports duty cycle range: (0%~100%), excluding 0% and 100%.

PWM frequency calculation:

| | Divide by N (N=1,2,4,8,16,32,64,128) |
|---|---|
| UP Mode | 16/N/(TOP_VAL+1) |
| UP and DOWN mode | 8/N/TOP_VAL |

PWM Notes:

- UPMODE supports frequency range: 62.5KHz~8MHz. Supported resolutions: 0 and 2/65536~65536/65536.
- UP AND DOWN MODE supports frequency range: 31.25KHz~4MHz. Supported resolutions: 0/65535~65534/65535.

### 3.3. PWM sample code

Refers to SDK

## 4. UART

### 4.1. UART description

Universal Asynchronous Receiver/Transmitter, commonly referred to as a UART. In UART communication, two UARTs communicate directly with each other. The transmit UART converts parallel data from a control device such as a CPU into serial form and sends it serially to the receive UART, which then converts the serial data back to parallel data for the receive device. Only two wires are required to transfer data between the two UARTs.

The UART transmits data asynchronously, which means that there is no clock signal to synchronize the bit output of the transmitting UART with the bit sampling of the receiving UART. Instead of clocking the sending UART, it adds start and stop bits to the packet being transmitted. These bits define the start and end of the packet, so the receiving UART knows when to start reading bits.

When the receiving UART detects the start bit, it starts reading the input bits at a specific frequency called the baud rate. Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps). Both UARTs must run at approximately the same baud rate. Data transmitted by the UART is organized into packets. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART), optional parity bits, and 1 or 2 stop bits.

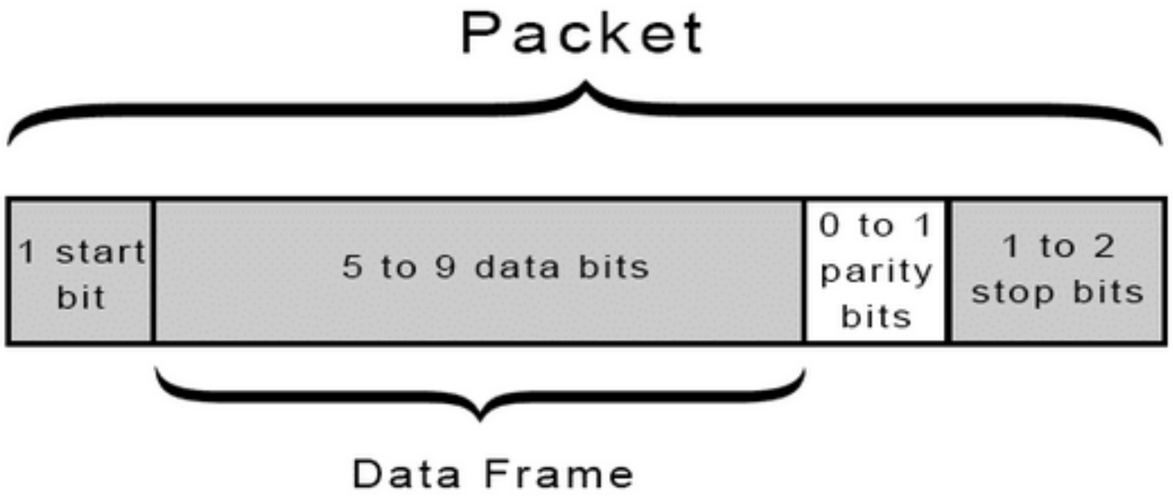

Fig 3: UART Frame Format

### 4.2. UART hardware

- 2 channels UART
- The hardware TXFIFO depth is 16 bytes, and the hardware RXFIFO depth is 16 bytes.
- The PCLK clock is equal to HCLK and can be divided by frequency, but frequency division is not recommended.
- When the system sleeps, the UART information is lost and needs to be reconfigured after waking up.
- All FMUX-capable IOs can be multiplexed as UART.
- System log printing uses UART0 (P9, P10) by default, which can be turned off or turned on through DEBUG_INFO configuration.
- Assuming that the current system main frequency is hclk and does not divide the frequency, the required baud rate is baud, and the actual hardware configuration register is divisor = (hclk) / (16 * baud). When the missing fractional part is greater than 2%, this baud rate does not support garbled characters. For example: the system clock is 48M, the baud rates are 115200, 921600, 1000000, the registers to be configured are 26.04166666666667, 3.255208333333333, 3, and the errors are 0.16%, 7.84%, and 0%, so 115200 and 1000000 are supported at this time. **921600 is not supported.**

### 4.3. UART sample code

refers to SDK

## 5. SPI

### 5.1. SPI description

SPI, stands for Serial Peripheral interface, as the name suggest is the serial peripheral interface. It is a synchronous, full-duplex, master-slave interface. Data from the master or slave is synchronized on the rising or falling edge of the clock. The master and slave can transmit data at the same time.

Only four lines are occupied on the pins of the chip, which saves the pins of the chip, and at the same time saves space and provides convenience for the layout of the PCB. It is precisely because of this easy-to-use feature that more and more chips are now This communication protocol is integrated.

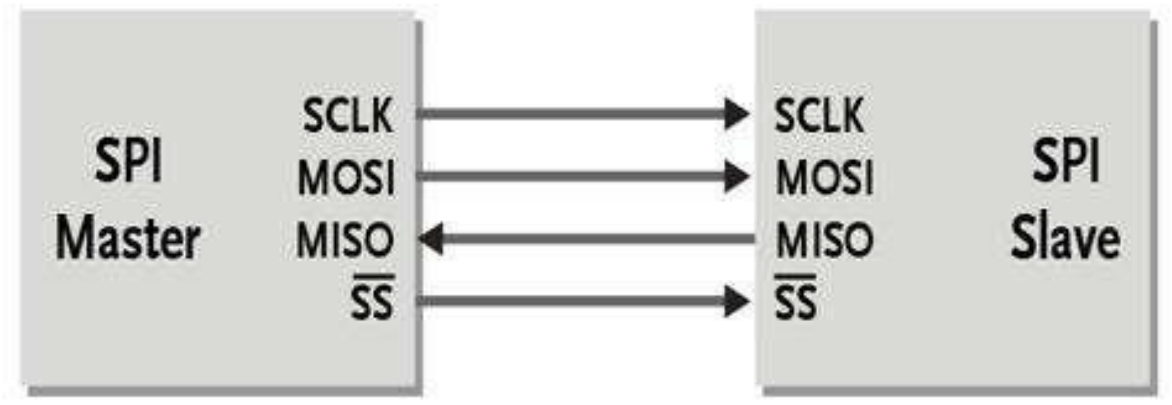

Fig 4: SPI connection

The device that generates the clock signal is called the master. The data transferred between the master and the slave is synchronized with the clock generated by the master. 4-wire SPI devices have four signals:

- Clock (SPI CLK, SCLK)
- Chip select (CS)
- Master Output, Slave Input (MOSI)
- Master Input, Slave Output (MISO)

In order to exchange data with the peripherals, the SPI module can configure the polarity and phase of the output serial synchronous clock according to the working requirements of the peripherals. The clock polarity (CPOL) has no significant impact on the transmission protocol.

- CPOL: Clock polarity selection, when it is 0, the SPI bus is idle and is low, and when it is 1, the SPI bus is idle and high.
- CPHA: Clock phase selection, when it is 0, it is sampled at the first transition edge of SCK, and when it is 1, it is sampled at the second transition edge of SCK.

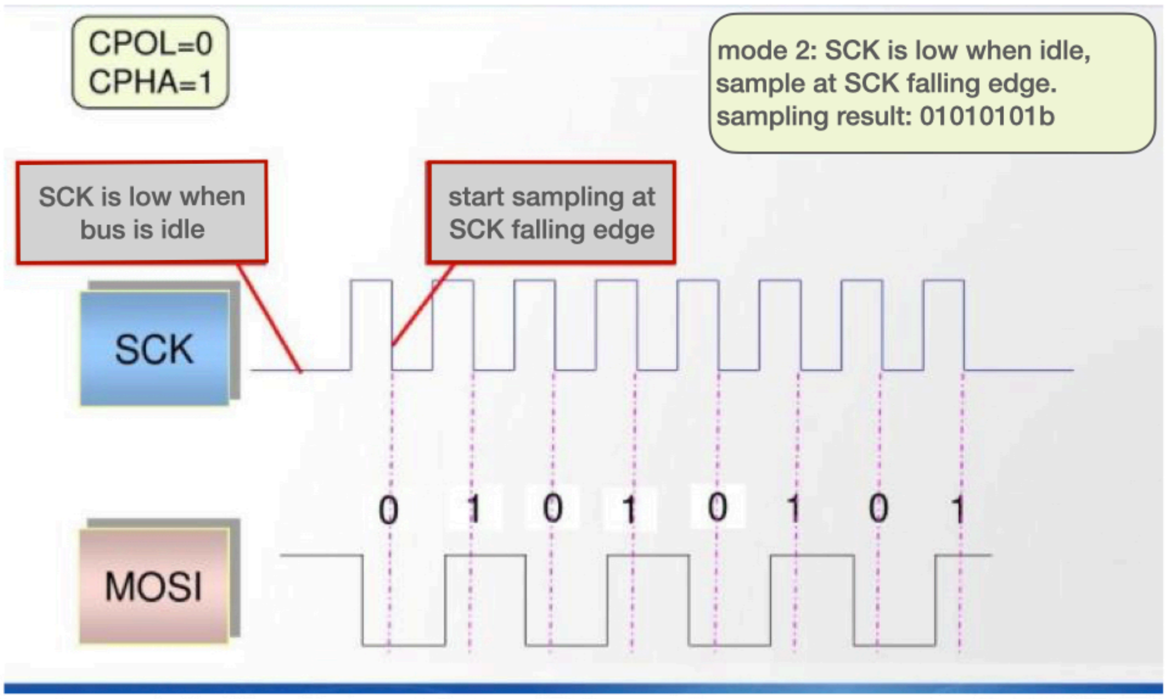
CPOL=0
CPHA=1
mode 2: SCK is low when idle, sample at SCK falling edge. sampling result: 01010101b
SCK is low when bus is idle
start sampling at SCK falling edge
SCK
0 1 0 1 0 1 0 1
MOSI

Fig 5: CPOL=0; CPHA=0

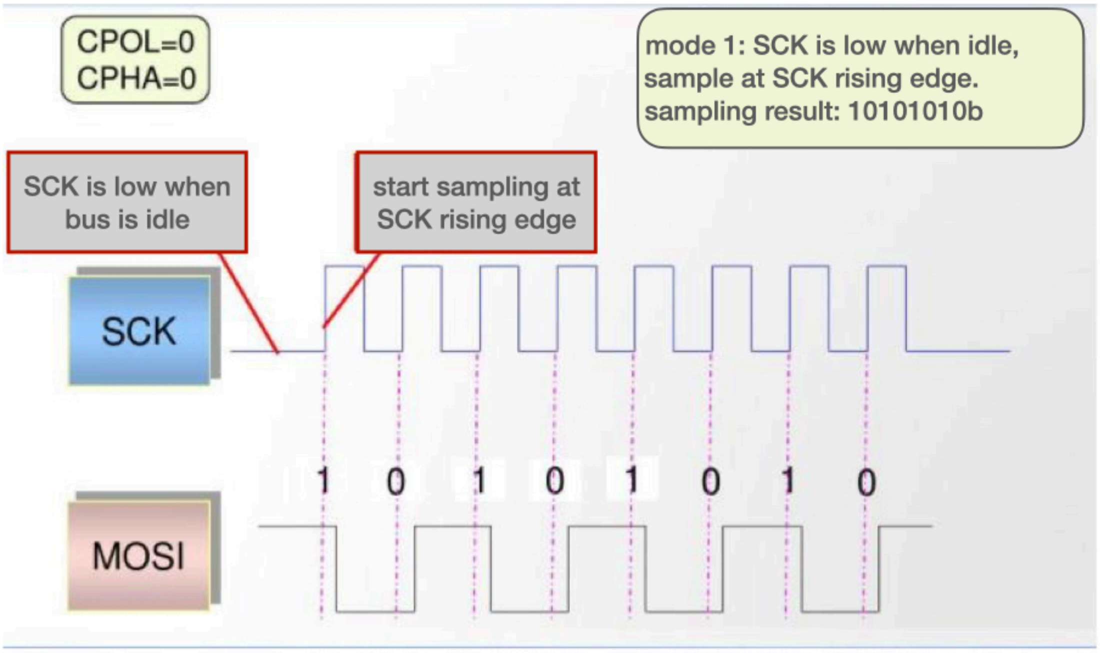
CPOL=0
CPHA=0
mode 1: SCK is low when idle, sample at SCK rising edge. sampling result: 10101010b
SCK is low when bus is idle
start sampling at SCK rising edge
SCK
1 0 1 0 1 0 1 0
MOSI

Fig 6: CPOL=0; CPHA=1

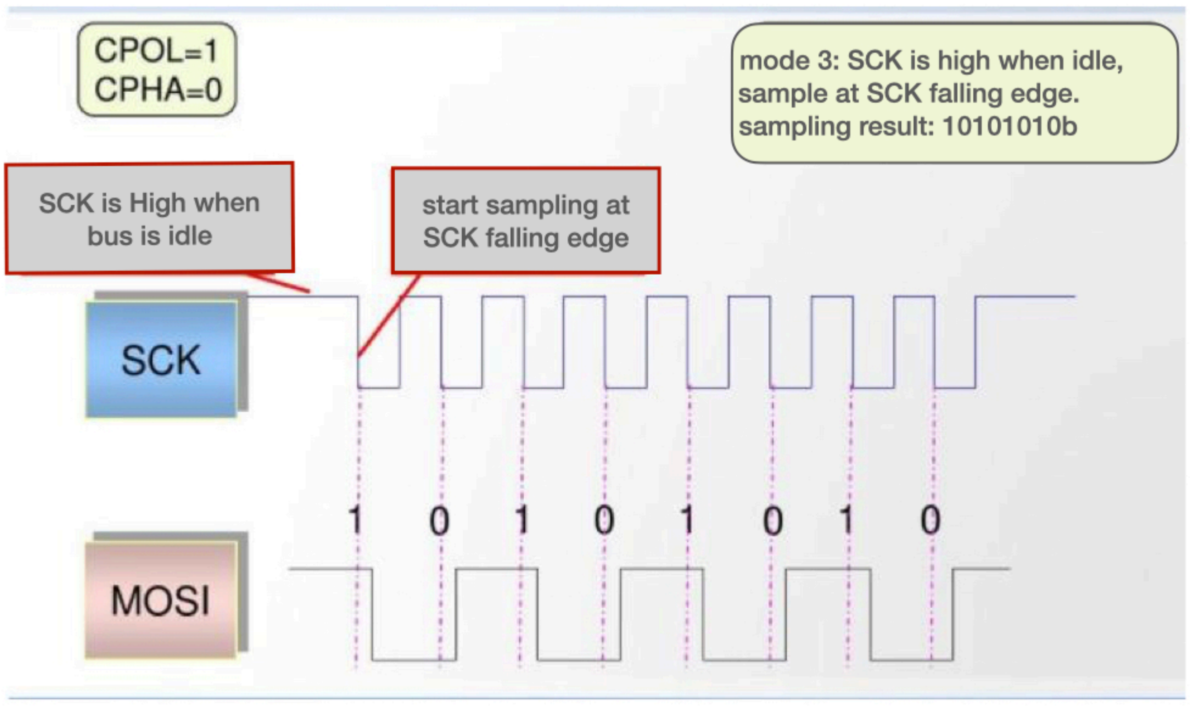
CPOL=1
CPHA=0
mode 3: SCK is high when idle, sample at SCK falling edge. sampling result: 10101010b
SCK is High when bus is idle
start sampling at SCK falling edge
SCK
1 0 1 0 1 0 1 0
MOSI

Fig 7: CPOL=1, CPHA=0

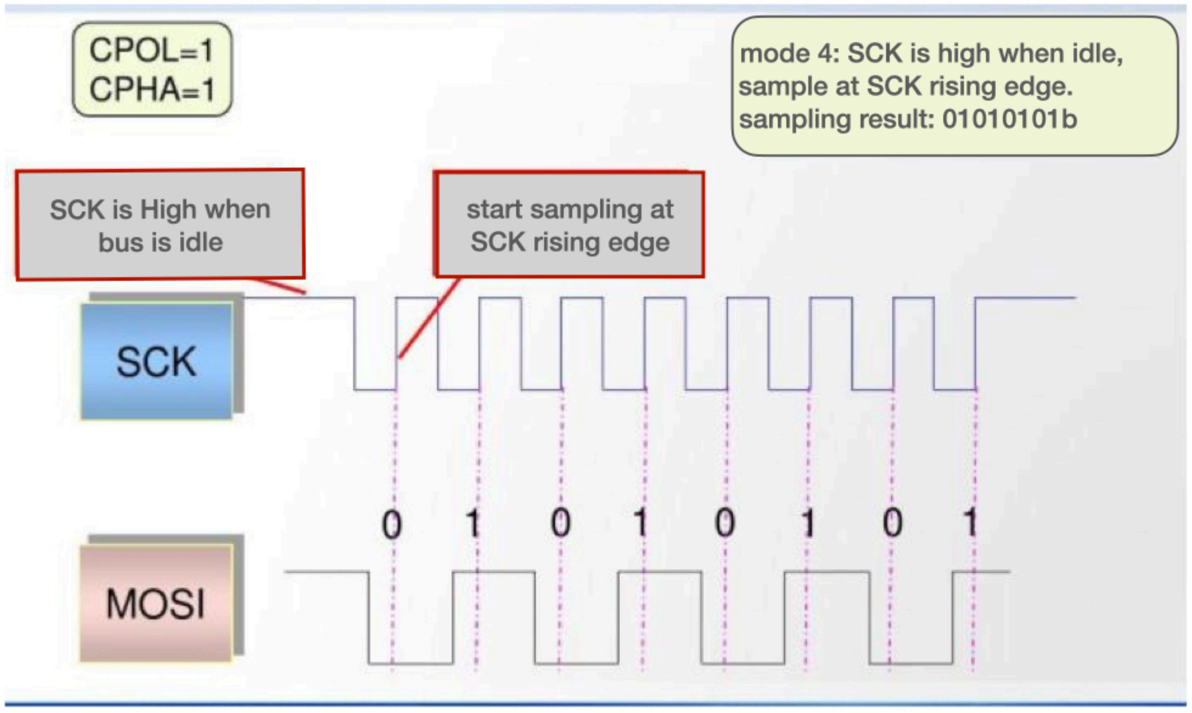
CPOL=1
CPHA=1
mode 4: SCK is high when idle, sample at SCK rising edge. sampling result: 01010101b
SCK is High when bus is idle
start sampling at SCK rising edge
SCK
0 1 0 1 0 1 0 1
MOSI

Fig 8: CPOL=1, CPHA=1

## 5.2.SPI hardware

Support 2 SPI channels, configurable to Master or Slave

The depth of hardware TXFIFO and RXFIFO is 8, the unit can be configured by software, and the range is 4bit~16bit

The clock is equal to HCLK and can be divided by frequency, but frequency division is not recommended.

When the system sleeps, the SPI information is lost and needs to be reconfigured after waking up.

All FMUX-capable IOs can be multiplexed as SPI.

When using SPI to send data, you can choose to control the level of CS automatically or manually, and you can choose whether to use interrupts.

In manual mode means setting IO as GPIO and outputting it high and low. Manual mode is selected when force_cs in spi_Cfg_t is TRUE, and interrupt mode is used when int_mode in spi_Cfg_t is TRUE.

### 5.3. Supported SPI speed

The SPI frequency has the following precautions, where F ssi_clk is pclk, and F sclk_in is spi clock.

- SPI Master: F ssi_clk >= 2 × (maximum F sclk_out )
- SPI Slave (receive only): F ssi_clk >= 6 × (maximum F sclk_in )
- SPI Slave: F ssi_clk >= 8 × (maximum F sclk_in)

### 5.4. SPI sample code

See SDK

## 6. I2C

### 6.1. I2C description

I2C Bus (Inter-Integrated Circuit Bus) was originally a two-wire serial bus developed by Philips Semiconductors (now acquired by NXP), which is often used for the connection between microcontrollers and peripherals.

I2C only needs two lines to support one-master multi-slave or multi-master connection. I2C uses two bidirectional open-drain lines, which are connected with pull-up resistors.

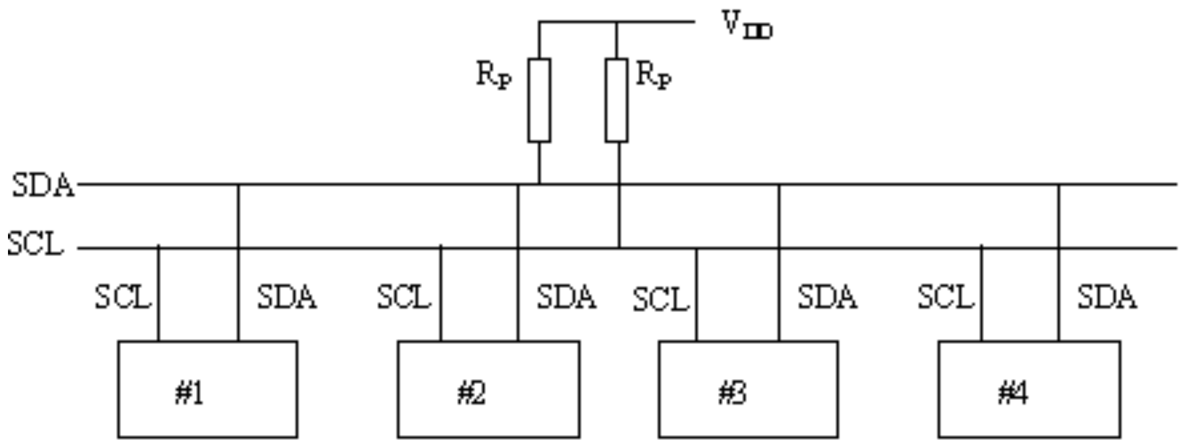

Fig 9: I2C diagram

### 6.2. I2C hardware

- Two channels I2C, which can be configured with MASTER or SLAVE.
- Hardware TXFIFO depth 8 bytes, hardware RXFIFO depth 8 bytes
- The PCLK clock is equal to HCLK and can be divided by frequency, but frequency division is not recommended.
- When the system sleeps, the I2C information is lost and needs to be reconfigured after waking up.
- All FMUX I/O can be configured as I2C.
- When using I2C, you need to connect a pull-up resistor, such as 2.2K or 4.7K.

### 6.3. I2C sample code

Refers to SDK

## 7. KSCAN

### 7.1.KSCAN description

KSCAN, also known as matrix keyboard, can be used when there are more buttons and less io. A kscan with M rows and N columns can implement M*N matrix keys.

KSCAN Branch and column, row input column output. When there is no button, each row on each column is in the idle state, and the key value is 0. When a button is pressed, the bit of the row corresponding to the column will be set to the key value of 1, and an interrupt will be triggered.

### 7.2.KSCAN hardware

11 io's can be configured as rows. 12 io can be configured as columns, P16P17 is not recommended. For the corresponding io and index, see kscan.h.

When the system sleeps, the KSCAN information is lost and needs to be reconfigured after waking up.

### 7.3.KSCAN sample code

Refers to SDK