Peripheral Development Guide

This document provide some frequent Q&A for peripheral usage

Table of Contents

1 GPIO

1.1 Function Description

1.1.1 I/O characteristic

Characteristic of each GPIO can be found in datasheet version later than xRTL81XXX_DataSheet_vXXrXX_201610XX. The table of “GPIO Characteristic” provides information including

- Whether this GPIO accepts external signal as interrupt sources.
- Whether this GPIO supports Schmitt trigger. The external signal can be debounced to remove any spurious glitches that are less than one period of the external debouncing clock.
- Driving Current
- Default State (PH = Pull-High, HI = High-impedance)
- Configurable input or output state during Sleep or Shutdown mode.

Symbol	Interrupt	Schmitt trigger	Driving[1] (mA)	**Default state**	**Sleep Mode, shutdown**
GPIOX_X	Y	N	8/16 (Configurable)	**PH**	**Input/Output; PU, PD or HI (Configurable)**

1.1.2 I/O pin multiplexer

Please refer to Pin Function Group Table in datasheet to check system requirement. Take for an example, the following table is Pin Function Table for 8195A.

PIN name	JTAG	SDD	SDH	MII	UART Group	I2C Group	SPI Group	I2S Group	PCM Group	WL_LED	PWM	ETE	WKDT
GPIOA_0		D2	D2	RX_CK	UART2_IN		SPI1_MISO						
GPIOA_1		D3	D3	RXD0	UART2_CTS		SPI1_MOSI						
GPIOA_2		CMD	CMD	RXD1	UART2_RTS		SPI1_CLK						
GPIOA_3		CLK	CLK	RXD2	UART0_RTS								
GPIOA_4		D0	D0	RXD3	UART2_OUT		SPI1_CS						
GPIOA_5		D1	D1	RXDV	UART0_CTS								D_SBY0
GPIOA_6		INT	CD	RXERR	UART0_IN								
GPIOA_7			WP	COL	UART0_OUT								
GPIOB_0					LOG_OUT							ETE0	D_SLP0
GPIOB_1					LOG_IN					WL_LED0		ETE1	
GPIOB_2						I2C3_SCL						ETE2	
GPIOB_3						I2C3_SDA						ETE3	
GPIOB_4										WL_LED0	PWM0		
GPIOB_5										WL_LED0	PWM1		
GPIOC_0				TXD2	UART0_IN		SPI0_CS0	I2S1_WS	PCM1_SYNC		PWM0	ETE0	
GPIOC_1				TXD1	UART0_CTS		SPI0_CLK	I2S1_CLK	PCM1_CLK		PWM1	ETE1	
GPIOC_2				TXD0	UART0_RTS		SPI0_MOSI	I2S1_SD_TX	PCM1_OUT		PWM2	ETE2	
GPIOC_3				TX_CK	UART0_OUT		SPI0_MISO	I2S1_MCK	PCM1_IN		PWM3	ETE3	
GPIOC_4				TXD3		I2C1_SDA	SPI0_CS1	I2S1_SD_RX					
GPIOC_5				TXEN		I2C1_SCL	SPI0_CS2						
GPIOD_4				MDC	UART2_IN	I2C0_SDA	SPI1_CS		PCM1_SYNC		PWM0	ETE0	
GPIOD_5				MDIO	UART2_CTS	I2C0_SCL	SPI1_CLK		PCM1_CLK		PWM1	ETE1	D_SBY2
GPIOD_6					UART2_RTS	I2C1_SCL	SPI1_MOSI	I2S0_SD_RX	PCM1_OUT		PWM2	ETE2	
GPIOD_7					UART2_OUT	I2C1_SDA	SPI1_MISO		PCM1_IN		PWM3	ETE3	
GPIOE_0	TRST				UART0_OUT	I2C2_SCL	SPI0_CS0	I2S0_WS	PCM0_SYNC		PWM0		
GPIOE_1	TDI				UART0_RTS	I2C2_SDA	SPI0_CLK	I2S0_CLK	PCM0_CLK		PWM1		
GPIOE_2	TDO				UART0_CTS	I2C3_SCL	SPI0_MOSI	I2S0_SD_TX	PCM0_OUT		PWM2		
GPIOE_3	TMS				UART0_IN	I2C3_SDA	SPI0_MISO	I2S0_MCK	PCM0_IN		PWM3		D_SBY3
GPIOE_4	CLK					I2C3_SCL	SPI0_CS1						
GPIOE_5						I2C3_SDA	SPI0_CS2						

Note 1: Function Pin is enabled via entire group, the un-used pin cannot be disabled separately.
Ex: if using debugger in SWD mode, the pin: JTAG_TRST, JTAG_TDI and JTAG_TDO cannot be used as GPIO at the same time.
Note 2: Only SPI0 CS0/CS1 can be separately configured as GPIO even if SPI0 is used at same time. Therefore, user can employ SPI0 in PC_0 ~ PC_4(or PE_0 ~ PE_4), while configuring PC_4(PE_4) and PC_5(PE_5) as GPIO function.

1.1.3 Interrupt configuration

The type of interrupt is programmable with one of the following settings:

- ✓ Active-high and level
- ✓ Active-low and level
- ✓ Rising edge
- ✓ Falling edge

Please check example gpio_level_irq to check how to set interrupt type.

```
/* High Level Trigger */
gpio_irq_set(&gpiol, IRQ_HIGH, 1);

/* Low Level Trigger */
gpio_irq_set(&gpiol, IRQ_LOW, 1);
```

Please check example gpio_irq to check how to set interrupt type.

```
/* Rising Edge Trigger */
gpio_irq_set(&gpio, IRQ_RISE, 1);

/* Falling Edge Trigger */
gpio_irq_set(&gpio, IRQ_FALL, 1);
```

2 SPI

2.1 Function Description

2.1.1 Ameba defines the PHA

With the SPI, the clock polarity (SCPOL) configuration parameter determines whether the inactive state of the serial clock is high or low. To transmit data, both SPI peripherals must have identical serial clock phase (SCPH) and clock polarity (SCPOL) values.

CPOL(clock polarity)

- CPOL = 0 SCK is Low when SPI is inactive.
- CPOL = 1 SCK is High when SPI is inactive.

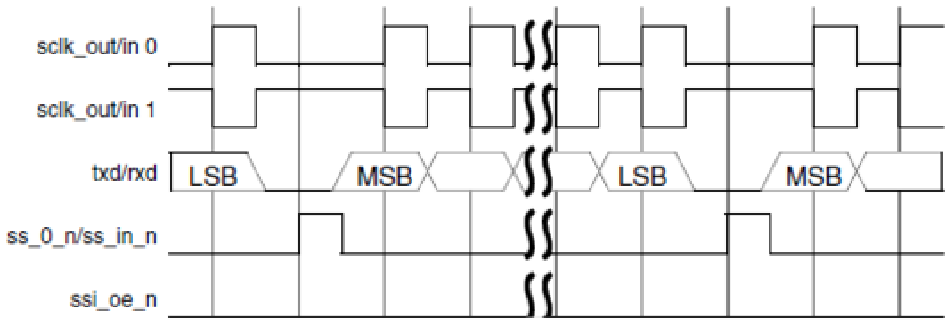
SCPH(clock phase)

- SCPH = 0 the SS toggle every frame. Indicates that the signal is received in first edge (Rising, when CPOL=0. Falling, when CPOL=1.).
- SCPH = 1 the SS toggle at start and stop. Indicates that the signal is received in second edge (Rising, when CPOL=1. Falling, when CPOL=0.).

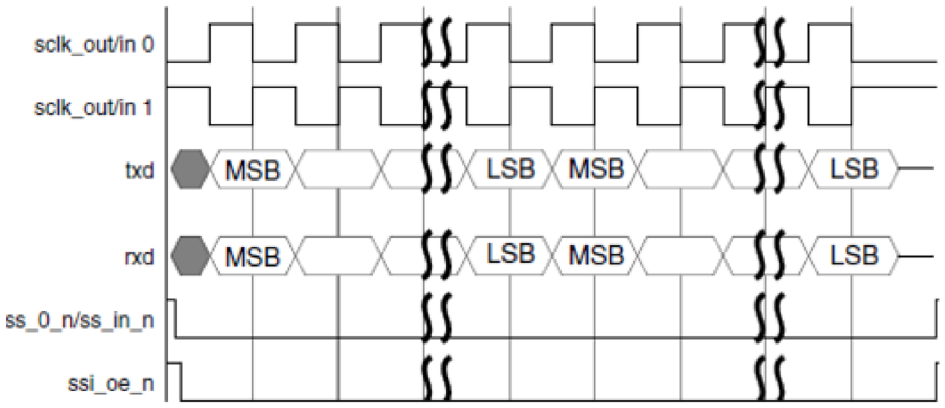
When SCPH = 0 the chip select toggle every frame. If other devices do not accept this setting, you can use another GPIO pin to control CS just toggle at start and stop.

Wave diagram:

Serial Format Continuous Transfers (SCPH = 0) when SSI_SCPH0_SSTOGGLE = 1



SPI Serial Format Continuous Transfer (SCPH = 1)



2.1.2 SPI_format

There are four possible transfer modes on the DW_apb_ssi for performing SPI serial transactions. For transmit and receive transfers (transfer mode field of the Control Register = 2'b00), data transmitted from the DW_apb_ssi to the external serial device is written into the transmit FIFO. Data received from the external serial device into the DW_apb_ssi is pushed into the receive FIFO.

mode	POL	PHA
0	0	0
1	0	1
2	1	0
3	1	1

*Default is mode 3.

Please check example spi to check how to set SCPOL_INACTIVE & SCPH_TOGGLES.

```
/* SCPOL_INACTIVE_IS_LOW & SCPH_TOGGLES_IN_MIDDLE */
spi_format(&spi_master, 8, 0, 0);

/* SCPOL_INACTIVE_IS_LOW & SCPH_TOGGLES_AT_START */
spi_format(&spi_master, 8, 1, 0);

/* SCPOL_INACTIVE_IS_HIGH & SCPH_TOGGLES_IN_MIDDLE */
spi_format(&spi_master, 8, 2, 0);

/* SCPOL_INACTIVE_IS_HIGH & SCPH_TOGGLES_AT_START */
spi_format(&spi_master, 8, 3, 0);
```

2.1.3 Interrupt mode and DMA mode

- Interrupt mode

Please check example **spi** & **spi_multislave** to check how to set interrupt mode.

```
/* send & recv one frame*/
spi_master_write(&spi_master, TestData);
spi_slave_read(&spi_slave);

/* send & recv target length data use interrupt mode */
spi_master_write_stream(&spi_master, TestBuf, TEST_BUF_SIZE);
spi_slave_read_stream(&spi_slave, TestBuf, TEST_BUF_SIZE);
```

- DMA mode

If using high-speed SPI, it is recommended to use DMA mode instead of interrupt mode to reduce the consumption of CPU resources.

Please check example **spi_stream_twoboard** to check how to set DMA mode.

```
/* send & recv target length data use DMA mode */
spi_master_write_stream_dma(&spi_master, TestBuf, TEST_BUF_SIZE);
spi_slave_read_stream_dma(&spi_slave, TestBuf, TEST_BUF_SIZE);
```

3 UART

3.1 Function Description

3.1.1 Features of UART

- High Speed UART (max baud rate 4MHz and DMA mode) and low speed UART (IO mode)
- UART (RS232 Standard) Serial Data Format
- Transmit and Receive Data FIFO
- Programmable Asynchronous Clock Support
- Auto Flow Control
- Programmable Receive Data FIFO Trigger Level
- DMA data moving support to save CPU loading

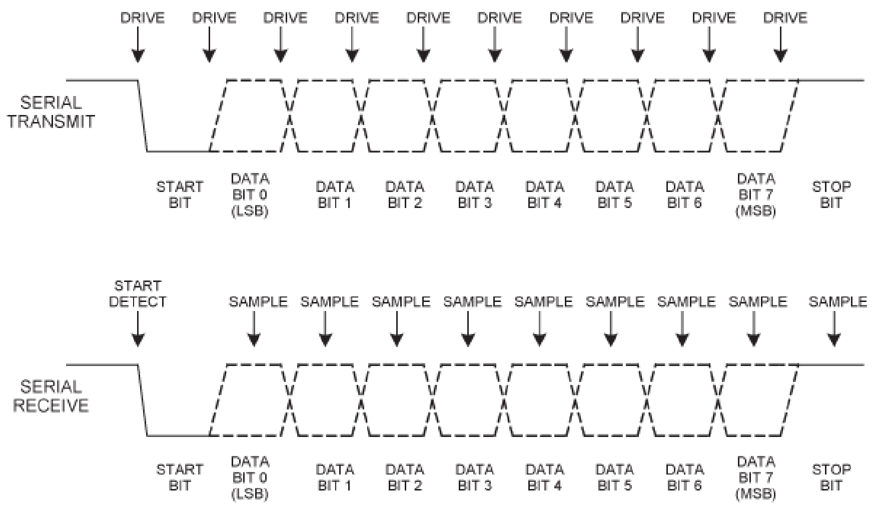


Figure UART Interface Waveform

3.1.2 Interrupt mode and DMA mode

3.1.2.1 Interrupt mode

Please check example **uart** & **uart_stream_irq** to check how to set interrupt mode.

```
/* send & recv one byte*/
serial_putc(sobj, *(pstr+i));
serial_getc(&sobj);

/* send & recv target length data use interrupt mode */
serial_send_stream(sobj, pstr, _strlen(pstr));
serial_recv_stream(&sobj, rx_buf, 8);
```

3.1.2.2 DMA mode

If using high-speed UART, it is recommended to use DMA mode instead of interrupt mode to reduce the consumption of CPU resources.

Please check example **uart_stream_dma** to check how to set DMA mode.

```
/* send & recv target length data use DMA mode */
serial_send_stream_dma(sobj, pstr, _strlen(pstr));
serial_recv_stream_dma(&sobj, rx_buf, 13);
```

3.1.2.3 Interrupt + DMA mode for UART RX

For UART RX, the first byte data can be received by UART interrupt mode then the other data can be received by UART DMA mode.

Please check example code as below.

```
void main(void)
{
    char temp[128];
    char temp2[8];
    int actual_len=0;
    rx_semaphore = xSemaphoreCreateCounting(0xFFFFFFFF, 0);
    DiagPrintf("===Receive start===\r\n");
    int32_t received;
    while(1)
    {
        serial_init(&serial, PA_7, PA_6);
        serial_baud(&serial, 115200);
        serial_format(&serial, 8, ParityEven, 1);
        serial_set_flow_control(&serial, FlowControlRTSCTS, 0, 0);
/*recv data use interrupt mode */
        received = serial_recv_blocked(&serial, temp2, 1, 500);
        if(received>0)
        {
            memcpy(temp+actual_len,temp2,received);
            actual_len += received;
            while(1)
            {
                /*recv data use DMA mode */
                received = uart_recv(temp2,1,30);
                if(received>0)
                {
                    memcpy(temp+actual_len,temp2,received);
                    actual_len += received;
                    if(actual_len==16)
                    {
                        for(int i=0;i<actual_len;i++)
                        {
                            DiagPrintf("%02x ", temp[i]);
                        }
                        DiagPrintf("\r\n");
                        actual_len = 0;
                        break;
                    }
                }
            }
        }
        serial_free(&serial);
    }
    for(;;){/*halt*/}
}
```

4 Reset

4.1 sys_reset()

sys_reset() is software reset, only CPU restart, the peripheral register setting is not be reset.

5 Debug

Hardware auto-detect whether the debugger adapter is JTAG or SWD, but if debugger is SWD, the pin used by JTAG (but not used in SWD mode) cannot be configured as general purpose I/O. If there is such requirement to configure those pin as GPIO, it is needed to disable jtag at first. Please refer to detail in chapter 5.3.

5.1 SWD and JTAG selection algorithm

To switch SWJ-DP from JTAG to SWD operation, debugger host send the following sequence:

1. Send more than 50 SWCLKTCK cycles with SWDIOTMS=1 to ensure that both SWD and JTAG are in their reset states.
2. Send the 16-bit JTAG-to-SWD select sequence 0b0011110011100111(MSB first) on SWDIOTMS.
3. Send more than 50 SWCLKTCK cycles with SWDIOTMS=1.

5.2 Disable JTAG

It is able to call *sys_jtag_off()* to turn of jtag. For system security consideration, turn on jtag is not supported.

5.3 Share SWD/JTAG with GPIO

Some UART/I2C/I2S/SPI may map to the same pin with SWD/JTAG, and to share the same pin, the following procedure can be considered.

- System boot up with JTAG on by default.
- The code jump to main function.
- Use one external GPIO to check whether JTAG will to be disabled or not. If JTAG is to be disabled, (system is in application mode), then disable jtag and configure the pin mux. Otherwise, if JTAG is not to be disabled (system is running on debugger mode), then do not disable jtag.

➢ Peripheral example example_sources\gpio_jtag\ demonstrates the scenarios.