MediaTek MT7688 SPI Slave Programming Guide

Version: 1.0

Release date: 10 May 2017

Document Revision History

Revision	Date	Description
1.0	10 May 2017	Initial release

Table of Contents

Lists of Tables and Figures

1. SPIS

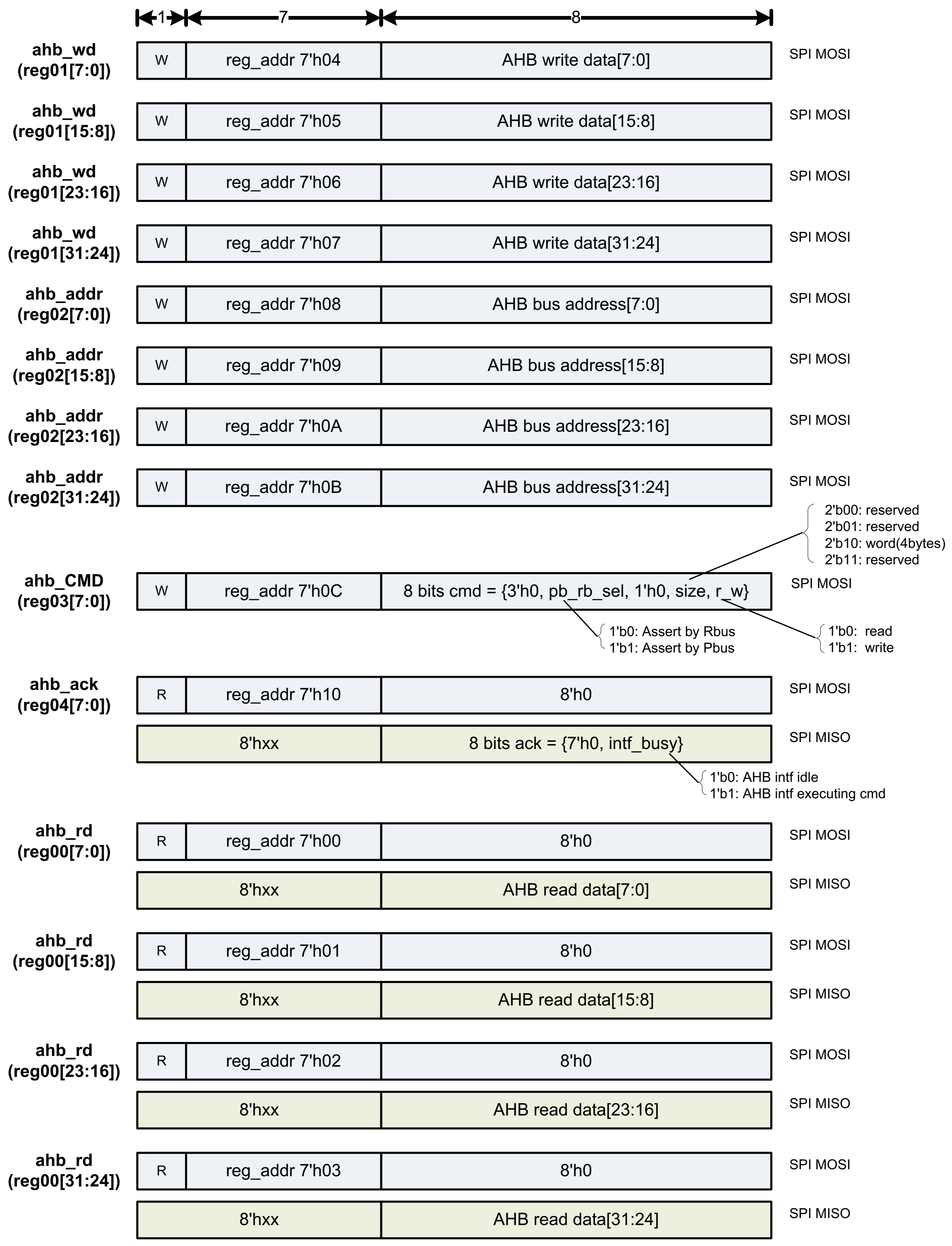
The SPI slave module translates the 16bits SPI serial protocol to create either Rbus or Pbus master transaction, for accessing DRAM data or configuring registers.

1.1. Overview

There is no software required to run on MediaTek MT7688 SPI slave. The only requirement is to setup pin-control for SPI slave, which means to disable Ethernet ports 1 to 4 (to set in IoT mode).

For SPI master to connect to MT7688 SPI slave, there is a protocol to follow (see section 1.2, “Protocol”), registers REG0 to REG4 in section 1.6.1, “Register of SPI Slave Interface”   are used to get DRAM(Rbus) data from or configuring registers(Pbus) on MT7688.

1.2. Protocol



There are 5 registers in this module. Reg00 is the read data from AHB. Reg01 is the write data that programmers want to write to AHB. Reg02 is the address that programmers want to write/read to/from AHB. The configured value must be a **physical address**. Reg03 is the command that applies to AHB protocol. Reg04 is the status for polling to make sure AHB bus is idle or busy.

Before programming AHB/APB registers, programmers should check reg04 bit0 to see if AHB is idle. Programmers can set reg03 (cmd register) to kick SPIS2AHB module to write/read one byte/halfword/word/dword to/from AHB/APB.

Before SPI write/read to/from AHB, programmers should guarantee AHB bus is non-busy by checking spitoahb_spi.reg04[0] to see if it equals to 1’b0.

1.3. Programming Sequence

1.3.1. Standard mode

1) Example 1: Write 0x0123_4567 data to the register at address 0x1013_0004

Step1.

Check if SPIS is idle. SPI master asserts following data on SPI_MOSI:

```
{1’b0, 7’h10, 8’hxx} // Read SPIS status reg04
```

Wait until the SPI_MISO returned data bit[0] = 0, which indicates the Rbus/Pbus interface of SPIS is idle and ready to execute a new access command.

Step2.

Prepare commands for bus accessing. SPI master asserts following data on SPI_MOSI respectively:

```
{1’b1, 7’h04, 8’h67} // put bus write data [7:0] into SPIS reg01[7:0]
{1’b1, 7’h05, 8’h45} // put bus write data [15:8] into SPIS reg01[15:8]
{1’b1, 7’h06, 8’h23} // put bus write data [23:16] into SPIS
reg01[23:16]
{1’b1, 7’h07, 8’h01} // put bus write data [31:24] into SPIS
reg01[31:24]
{1’b1, 7’h08, 8’h04} // put bus address [7:0] into SPIS reg02[7:0]
{1’b1, 7’h09, 8’h00} // put bus address [15:8] into SPIS reg02[15:8]
{1’b1, 7’h0A, 8’h13} // put bus address [23:16] into SPIS reg02[23:16]
{1’b1, 7’h0B, 8’h10} // put bus address [31:24] into SPIS reg02[31:24]
{1’b1, 7’h0C, {3’b0, 1’b0, 1’b0, 2’b10, 1’b1}} // Start the bus write
access via Rbus master interface
```

Step3.

Wait for the bus accessing to be done. SPI master asserts following data on SPI_MOSI:

```
{1’b0, 7’h10, 8’hxx} // Read SPIS bus interface status
```

Wait until the SPI_MISO returned data bit[0] = 0, and make sure that either Rbus or Pbus finishes the bus access.

2) Example 2: Read 0x0123_4567 data from the register at address 0x1013_0004

Step1.

Check if SPIS is idle. SPI master asserts following data on SPI_MOSI:

```
{1’b0, 7’h10, 8’hxx} // Read SPIS status reg04
```

Wait until the SPI_MISO returned data bit[0] = 0, which indicates the Rbus/Pbus interface of SPIS is idle and ready to execute a new access command.

Step2.

Prepare commands for bus accessing. SPI master asserts following data on SPI_MOSI respectively:

```
{1'b1, 7'h08, 8'h04} // put bus address [7:0] into SPIS reg02[7:0]
{1'b1, 7'h09, 8'h00} // put bus address [15:8] into SPIS reg02[15:8]
{1'b1, 7'h0A, 8'h13} // put bus address [23:16] into SPIS reg02[23:16]
{1'b1, 7'h0B, 8'h10} // put bus address [31:24] into SPIS reg02[31:24]
{1'b1, 7'h0C, {3'b0, 1'b0, 1'b1, 2'b10, 1'b0}} // Start the bus read
access via Pbus master interface
```

Step3.

Wait for the bus accessing to be done. SPI master asserts following data on SPI_MOSI:

```
{1'b0, 7'h10, 8'hxx} // Read SPIS status
```

Wait until the SPI_MISO returned data bit[0] = 0, and make sure that either Rbus or Pbus finishes the bus access

Step4.

Get read data. SPI master asserts following data on SPI_MOSI respectively:

```
{1'b0, 7'h00, 8'hxx} // get bus read data [7:0] from SPIS reg00[7:0]
```

SPIS_MISO return data 16'hxx_67

```
{1'b0, 7'h01, 8'hxx} // get bus read data [15:8] from SPIS reg00[15:8]
```

SPIS_MISO return data 16'hxx_45

```
{1'b0, 7'h02, 8'hxx} // get bus read data [23:16] from SPIS reg00[23:16]
```

SPIS_MISO return data 16'hxx_23

```
{1'b0, 7'h03, 8'hxx} // get bus read data [31:24] from SPIS reg00[31:24]
```

SPIS_MISO return data 16'hxx_01

1.3.2. Sequential mode

1) Example 1: Write 0x0123_4567 data to the register at address 0x1013_0004

Step1.

Check if SPIS is idle. SPI master asserts following data on SPI_MOSI:

```
{1'b0, 7'h10, 8'h00} // Read SPIS bus interface status
```

Wait until the SPI_MISO returned data bit[0] = 0, which indicates the Rbus/Pbus interface of SPIS is idle and ready to execute a new access command.

Step2.

Prepare commands for bus accessing. SPI master asserts following data on SPI_MOSI.

```
{1'b1, 7'h04, 8'h67, 8'h45, 8'h23, 8'h01, 8'h04, 8'h00, 8'h13, 8'h10,
{3'b0, 1'b0, 1'b0, 2'b10, 1'b1}}
// put bus write data[31:0] into SPIS reg01[31:0], put bus address
[31:0] into SPIS reg02[31:0]
// Start the bus write access via Rbus master interface
```

Step3.

Wait for the bus accessing to be done. SPI master asserts following data on SPI_MOSI:

```
{1'b0, 7'h10, 8'h00} // Read SPIS bus interface status
```

Wait until the SPI_MISO returned data bit[0] = 0, make sure that either Rbus or Pbus finish the bus access.

2) Example 2: Read 0x0123_4567 data from the register at address 0x1013_0004

Step1.

Check if SPIS is idle. SPI master asserts following data on SPI_MOSI:

```
{1'b0, 7'h10, 8'hxx} // Read SPIS status reg04
```

Wait until the SPI_MISO returned data bit[0] = 0, which indicates the Rbus/Pbus interface of SPIS is idle and ready to execute a new access command.

Step2.

Prepare commands for bus accessing. SPI master asserts following data on SPI_MOSI:

```
{1'b1, 7'h08, 8'h04, 8'h00, 8'h13, 8'h10, {3'b0, 1'b0, 1'b1, 2'b10,
1'b0}}
// put bus address [31:0] into SPIS reg02[31:0]
// Start the bus read access via Pbus master interface
```

Step3.

Wait for the bus accessing to be done. SPI master asserts following data on SPI_MOSI

```
{1'b0, 7'h10, 8'hxx} // Read SPIS status reg04
```

Wait until the SPI_MISO returned data bit[0] = 0, make sure that either Rbus or Pbus finish the bus access.

Step4.

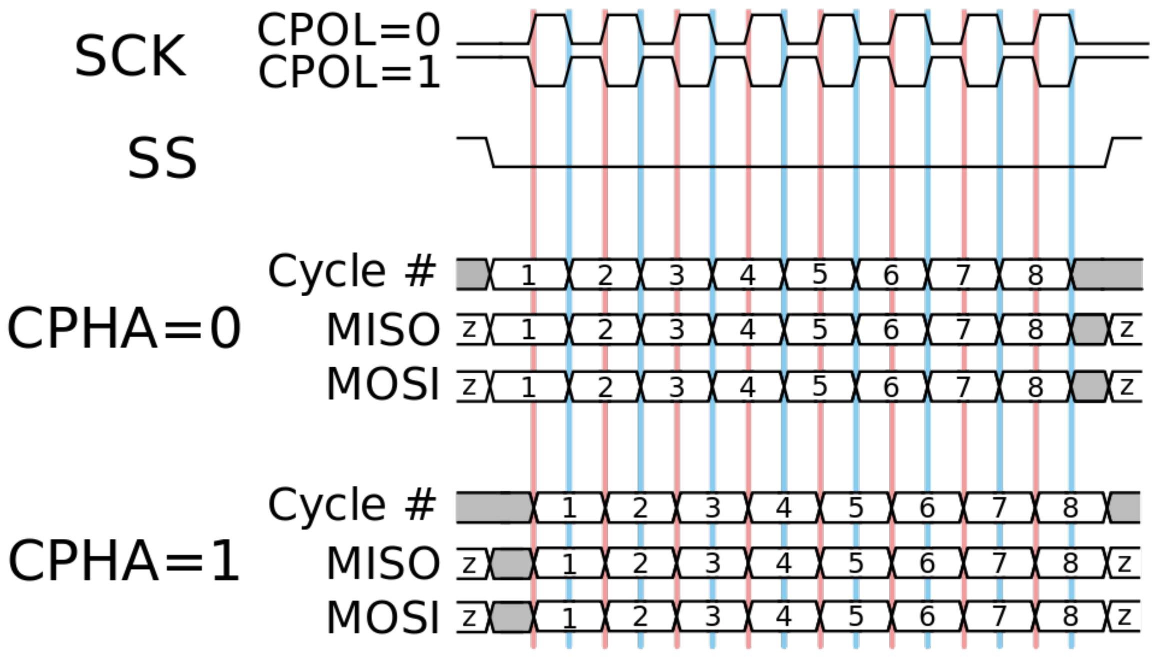
Get read data. SPI master asserts following data on SPI_MOSI:

```
{1'b0, 7'h00, 8'hxx, 8'hxx, 8'hxx, 8'hxx} // get bus read data from SPIS
reg00[31:0]
```

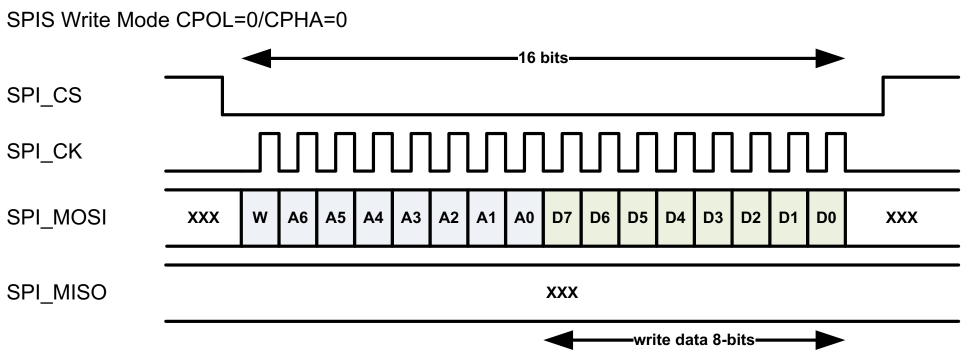
SPIS_MISO return data {8'hxx, 8'h67, 8'h45, 8'h23, 8'h01}

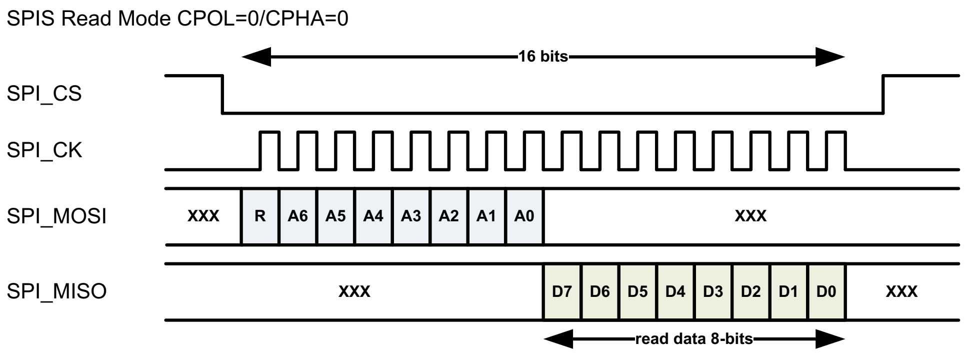
1.4. Protocol Timing

Limitation: Maximum clock frequency: 20MHz

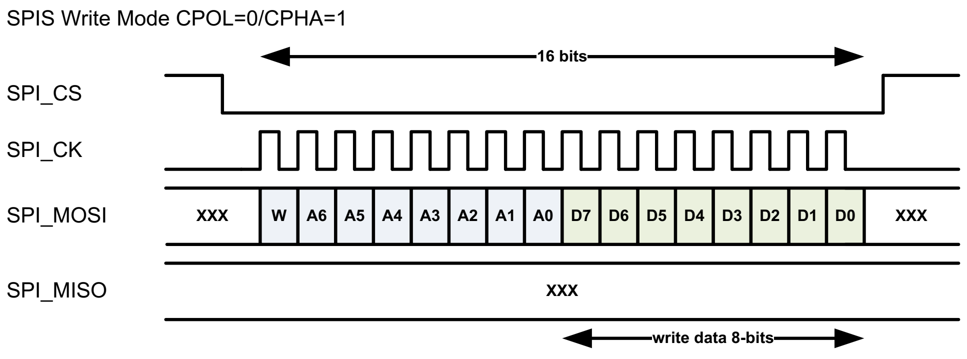


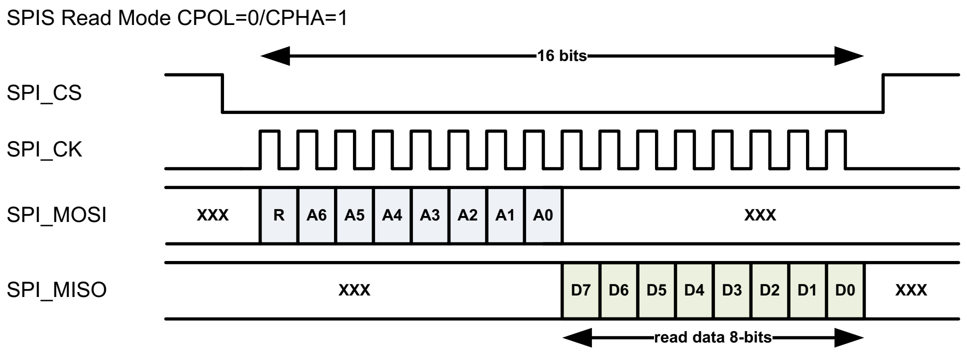
1.4.1. CPOL=0, CPHA=0


SPIS Write Mode CPOL=0/CPHA=0
16 bits
SPI_CS
SPI_CK
SPI_MOSI
XXX
W A6 A5 A4 A3 A2 A1 A0 D7 D6 D5 D4 D3 D2 D1 D0
XXX
SPI_MISO
XXX
write data 8-bits

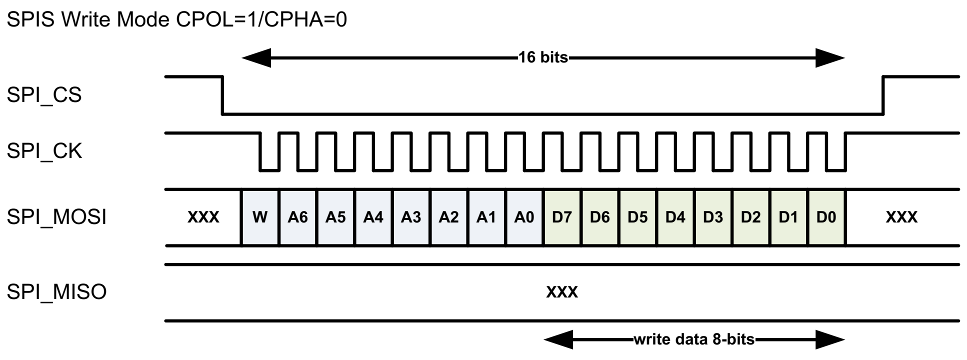


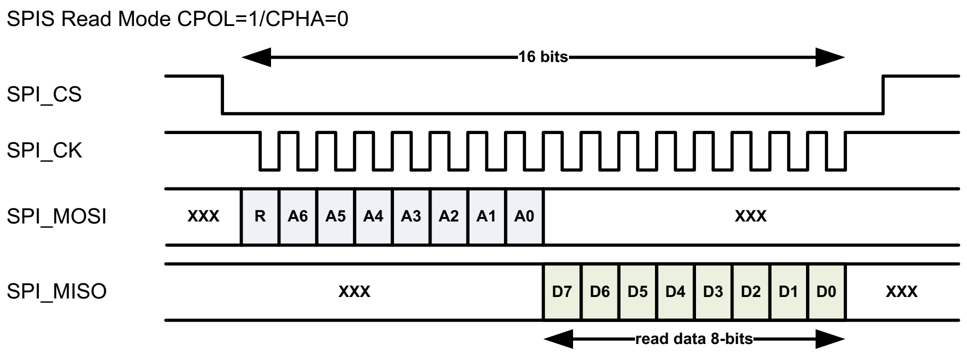
1.4.2. CPOL=0, CPHA=1



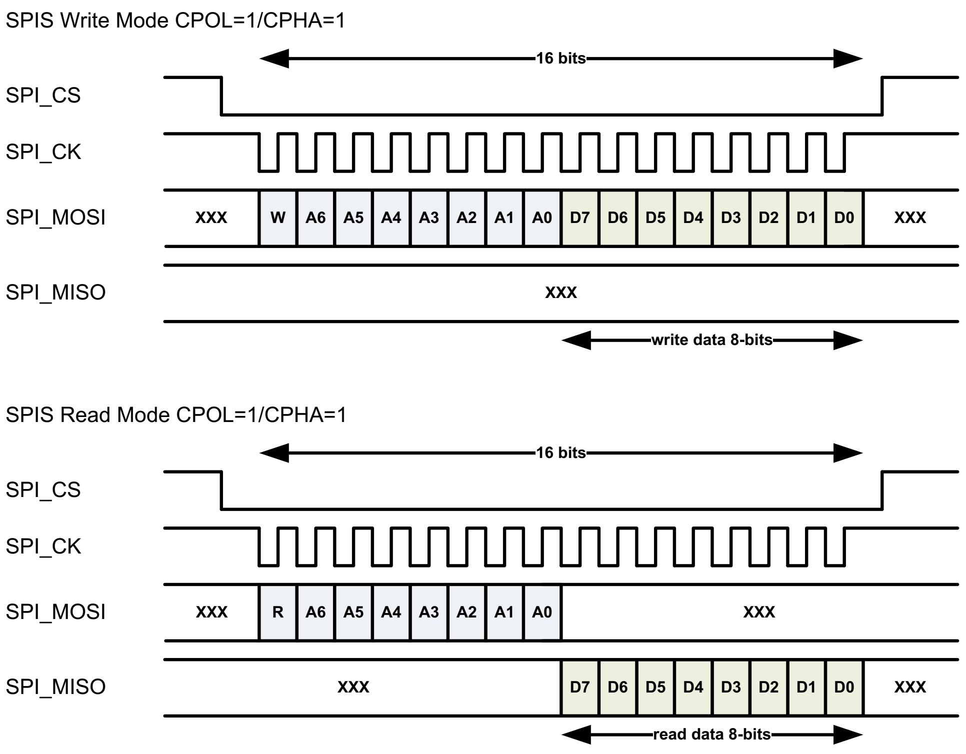


1.4.3. CPOL=1, CPHA=0





1.4.4. CPOL=1, CPHA=1



1.5. Interrupt

SPIS doesn’t have dedicated hardware interrupt indication signal. SPI master could configure a software interrupt source of interrupt controller to inform the CPU of SPI slave side.

1.6. Operation Register

1.6.1. Register of SPI Slave Interface

Module name: SPIS Base address: (+0h)

Address	Name	Width	Register Function
00000000	REG00	32	SPI Slave Register 00
00000004	REG01	32	SPI Slave Register 01
00000008	REG02	32	SPI Slave Register 02
0000000C	REG03	32	SPI Slave Register 03
00000010	REG04	32	SPI Slave Register 04

1.7. Register Descriptions

00000000 REG00 SPI Slave Register 00 00000000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	bus_read_data[31:16]															
Type	RO															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	bus_read_data[15:0]															
Type	RO															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit(s)	Name	Description
31:0	bus_read_data	SPI Slave Register 00 for bus read data

00000004 REG01 SPI Slave Register 01 00000000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	bus_write_data[31:16]															
Type	RW															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	bus_write_data[15:0]															
Type	RW															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit(s)	Name	Description
31:0	bus_write_data	SPI Slave Register 01 for bus write data

00000008 REG02 SPI Slave Register 02 00000000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	bus_address[31:16]															
Type	RW															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	bus_address[15:0]															
Type	RW															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit(s)	Name	Description
31:0	bus_address	SPI Slave Register 02 for bus address This address must be physical address

0000000C REG03 SPI Slave Register 03 00000000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	reg03_31_5[26:11]															
Type	RW															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	reg03_31_5[10:0]											bus_pb_rb_sel	reg03_3	bus_size		bus_r_w
Type	RW											RW	RW	RW		RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit(s)	Name	Description
31:5	reg03_31_5	reg03[31:5] reserved bit
4	bus_pb_rb_sel	Bus interface selection 0: Bus transaction is asserted by Rbus master interface, can access DRAM and peripheral registers 1: Bus transaction is asserted by Pbus master interface, can peripheral registers only.
3	reg03_3	reg03[3] reserved bit
2:1	bus_size	Bus access size 00: reserved 01: reserved 10: word (4bytes) 11: reserved
0	bus_r_w	Bus access type

		0: read 1: write

00000010 REG04 SPI Slave Register 04 00000000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name																
Type																
Reset																
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name																bus_busy
Type																RO
Reset																0

Bit(s)	Name	Description
0	bus_busy	Bus interface status 0: SPIS bus interface is idle for next access command 1: SPIS bus interface is busy

1.7.1. Register of Pbus Slave Interface

The CPU at SPIS side could access this Pbus slave interface to probe the SPIS internal registers reg00~reg04 status, and control the SPI polarity/phase. Users can't configure SPIS reg via this pbus slave interface, and SPIS reg must be configured by SPI master.

Module name: spis_pbslv Base address: (+10000700h)

Address	Name	Width	Register Function
10000700	SPIS_REG00	32	SPI Slave Register 00
10000704	SPIS_REG01	32	SPI Slave Register 01
10000708	SPIS_REG02	32	SPI Slave Register 02
1000070C	SPIS_REG03	32	SPI Slave Register 03
10000710	SPIS_REG04	32	SPI Slave Register 04
10000740	SPIS_CFG	32	SPI Slave Configuration

10000700 SPIS_REG00 SPI Slave Register 00 00000000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	spis_reg00[31:16]															
Type	RO															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	spis_reg00[15:0]															
Type	RO															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit(s)	Name	Description
31:0	spis_reg00	SPI Slave Register 00

10000704 SPIS_REG01 SPI Slave Register 01 00000000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	spis_reg01[31:16]															
Type	RO															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	spis_reg01[15:0]															
Type	RO															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit(s)	Name	Description
31:0	spis_reg01	SPI Slave Register 01

10000708 SPIS_REG02 SPI Slave Register 02 00000000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	spis_reg02[31:16]															
Type	RO															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	spis_reg02[15:0]															
Type	RO															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit(s)	Name	Description
31:0	spis_reg02	SPI Slave Register 02

1000070C SPIS_REG03 SPI Slave Register 03 00000000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	spis_reg03[31:16]															
Type	RO															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	spis_reg03[15:0]															
Type	RO															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit(s)	Name	Description
31:0	spis_reg03	SPI Slave Register 03

10000710 SPIS_REG04 SPI Slave Register 04 00000000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	spis_reg04[31:16]															
Type	RO															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	spis_reg04[15:0]															
Type	RO															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit(s)	Name	Description
31:0	spis_reg04	SPI Slave Register 04

10000740 SPIS_CFG SPI Slave Configuration 00000000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name																
Type																
Reset																
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name															spis_mode	
Type															RW	
Reset															0	0

Bit(s)	Name	Description
1:0	spis_mode	SPI slave clock polarity and phase configuration 2'b00: CPOL=0, CPHA=0 2'b01: CPOL=0, CPHA=1

2'b10: CPOL=1, CPHA=0
2'b11: CPOL=1, CPHA=1