Datasheet

CC-SPI-AXI Documentation

1.2

Scope

This document describes the CC-SPI-AXI IP core. Module features and configuration registers are described. The document contains integration guide that covers synthesis options and instantiation example for easy implementation in customer's environment.

Contents

1. SPI Controller

1.1 Functionality

- Master or Slave customizable operating modes,
- full duplex,
- customizable bits order during data transmission (starting from MSB or LSB)
- customizable data frame length (8, 16, 24 or 32 bits),
- programmable transmission speed in Master mode,
- customizable interrupt for data transfer stages signaling and for reporting events on nSS line (in Slave mode),
- four modes of data transfer.

1.2 Overview

Serial Peripheral Interface (SPI) is a synchronous communication interface using 3 or 4 lines. Data exchange is held between two devices where one of them acts as a Master and the other one as a Slave during communication. Master device initializes and controls data transfer. Figure 1.1 shows typical schematic of Master and Slave connection. In

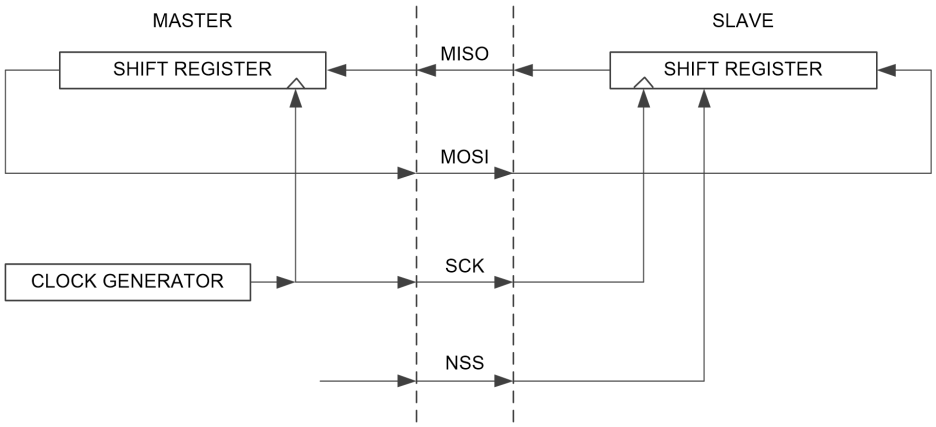


Figure 1.1. SPI Master and Slave connection.

general view, the system consists of two shift registers and SCK clock signal generator. The SPI Master initiates the data exchange by setting low state on the Slave Select (nSS) line of the selected SPI Slave. Both devices place data in their shift registers, afterwards the SPI Master generates the appropriate number of pulses on SCK line (equal to the number of transmitted bits). Data are transmitted from the Master to the Slave on the MOSI line (Master Out – Slave In) and from the Slave to the Master on the MISO line (Master In – Slave Out). After transmitting all data packages (of any length), the SPI Master device may synchronize the SPI Slave device by setting high state on the nSS line.

1.3 Block Diagram

The SPI controller is a serial communication device. The device enables two-way data transmission (full duplex) by three or four communication lines. The controller may act as a Master or Slave device. Transmission paths (i.e. sending and receiving paths) are buffered, thus eliminating delays between subsequent data frames. The controller provides interrupt signal, which may be used at program level for data transmission control. Buffer overflow can be signaled by raising an interrupt event and setting the corresponding status bit. Figure 1.2 shows block diagram of the SPI controller.

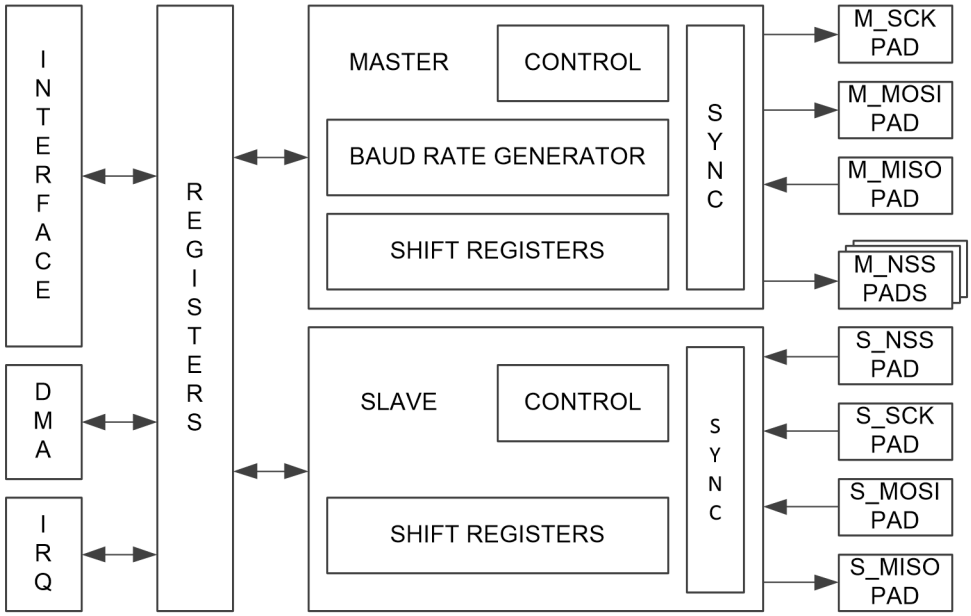


Figure 1.2. The SPI controller block diagram

1.4 Data Transmission Modes

The SPI controller supports 4 data transfer modes. The modes are defined by the combination of phase and polarity of the SCK signal with respect to the transmitted data. Table 1.1 and Figure 1.3 shows details of data transmission modes.

Mode	First SCK edge	Second SCK edge
MODE 0	raising - bit sampling	falling - bit exposure
MODE 1	raising - bit exposure	falling - bit sampling
MODE 2	falling - bit sampling	raising - bit exposure
MODE 3	falling - bit exposure	raising - bit sampling

Table 1.1. Data transmission modes

Data bits are sampled and exposed ('shifting out') on different SCK clock edges. This provides a sufficiently long time to stabilize the signals. Additionally, the controller enables to program the bits transmission order (from LSB to MSB and vice versa) and data frame length (8, 16, 24 or 32 bits).

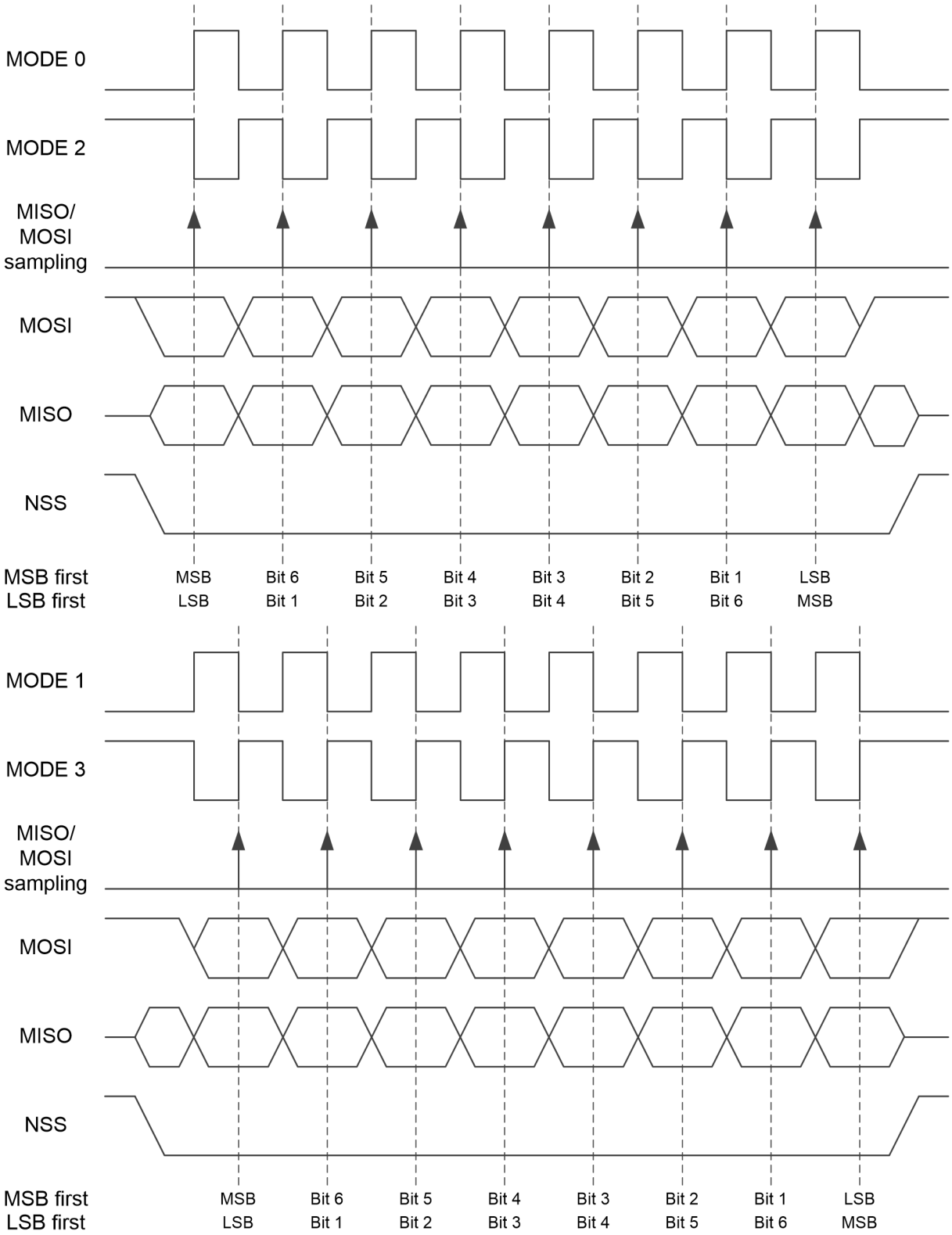


Figure 1.3. Data transmission modes.

1.5 SPI Master

1.5.1 Overview

In the Master mode, the controller does not automatically control the nSS line. The user application should use NSS register (if available) or custom GPIO line for this purpose. The module consists of three main components:

- SCK clock signal generator,
- sender's shift register with frame logic,
- receiver's shift register with frame logic.

The SCK clock signal generator is a frequency divider generating SPI interface clock signal. The maximum possible frequency to obtain on SCK line is half of the peripheral clock – $\frac{F_{PCLK}}{2}$. The SCK line simultaneously clocks data shifting from/into sender's/receiver's shift register. Length of a data frame is programmable and the frame transmission starts when data are available in the TDR register (1.8.4).

1.5.2 Data Transmission

The sender unit of SPI Master module contains a bits counter and a shift register. The bits counter based on the data transmission mode (the MODE bits – 1.8.3), the order of bits transfer (the MSB bit – 1.8.3) and the length of data frame (the FLEN bits – 1.8.3) controls sender's shift register and status lines.

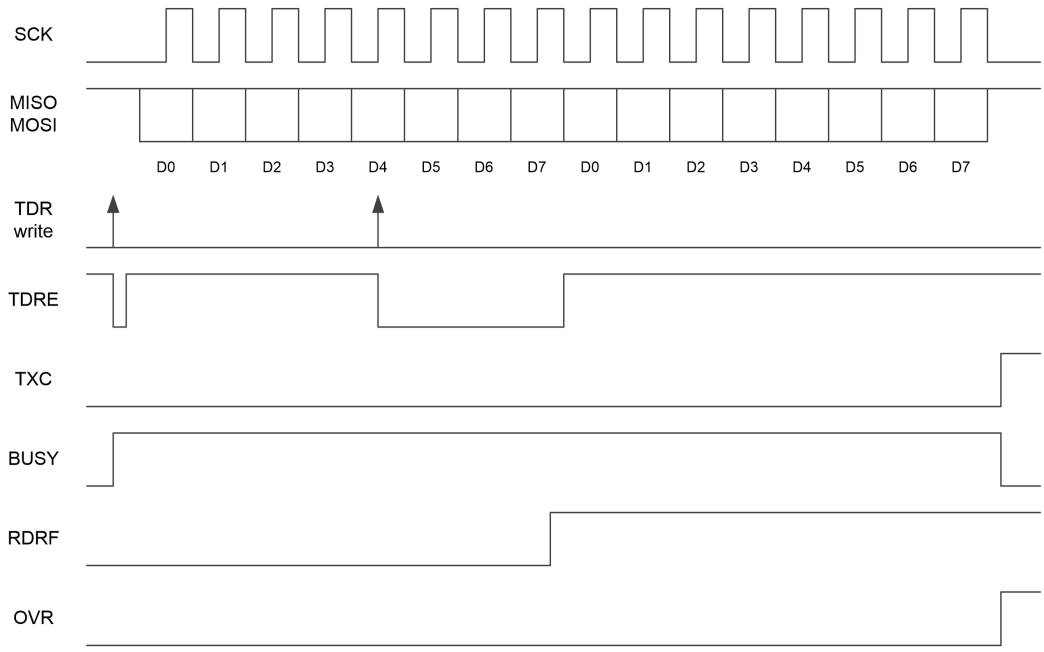


Figure 1.4. Data transmission in SPI Master mode (MODE 0).

Loading data into the TDR register (1.8.4) starts the SCK clock generation and begins the frame transmission (Figure 1.4). Data are locked in the shift register, what is signaled by the TDRE status bit (1.8.2). This indicates that the user may load a new data into the TDR register. The new data is going to be send immediately after finishing ongoing

transmission (Figure 1.4). If the TDR register doesn't have any new data to send and current transmission ends then the SPI controller finishes data sending. This is indicated by setting the high state of the TXC bit and the low state of the BUSY bit in the STATUS register (1.8.2).

1.5.3 Data Receiving

The receiver unit of SPI Master module contains a bits counter and a shift register. The implementation of SPI interface assumes simultaneous two-way communication (Figure 1.4). Therefore, data receiving is initialized by writing a data into the TDR register (1.8.4). The written data format depends on SPI Slave device (Figure 1.4). The SPI controller signals the data receiving by setting the high state of the RDRF bit in the STATUS register (1.8.2). The received data stored in the RDR register (1.8.5) will be lost if the register will not be read by the software before the receiving of current data frame is completed. The lost of the data is signaled (Figure 1.4) by the high state of the OVR bit in the STATUS register (1.8.2). When both upstream and downstream DMA channels are active, then the SPI transmit module will automatically delay the transmission of new data, if necessary, until RDRF bit is cleared. This prevents the overflow from occur. In other cases overflow event has to be handled in the user program.

1.5.4 Pad Output Enable

The figure below presents the behaviour of output enable signal that drives SCK and MOSI lines.

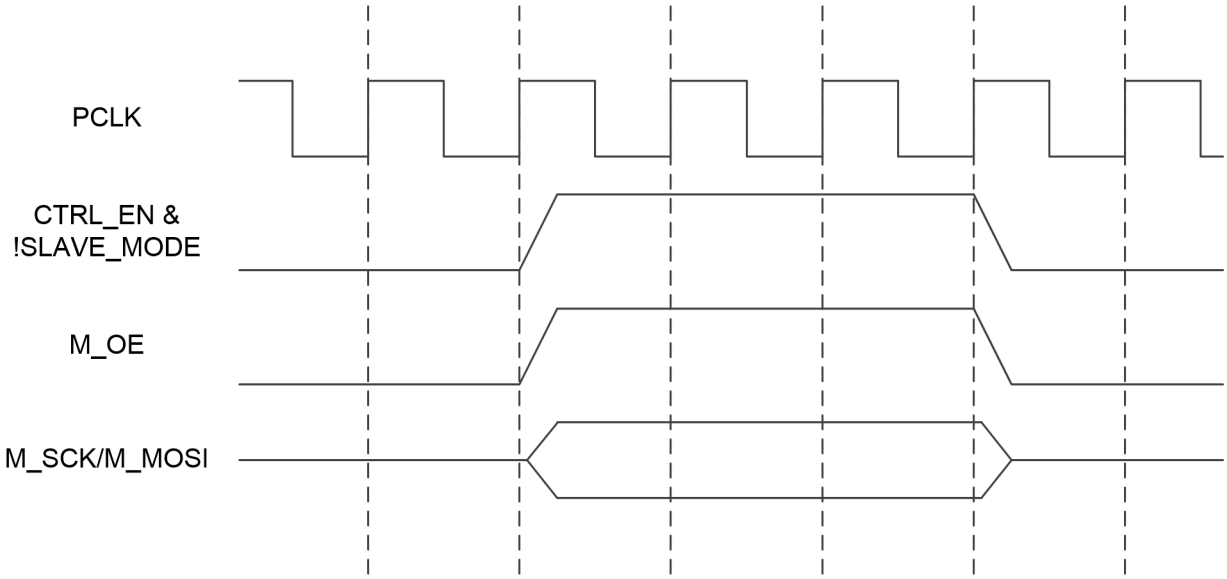


Figure 1.5. SPI Master pad output enable scheme.

1.6 SPI Slave

1.6.1 Overview

The input signals of the SPI interface are synchronized by a 2FF synchronizers clocked by the PCLK signal. Sampling of the MOSI input line (synchronized to the PCLK clock) and writing bits into the input shift register is done on proper edge of the SCK clock. Including the delay introduced by the clock synchronizers minimal time period of low and high states of the SCK line is three cycles of the PCLK clock. This constraints the maximum input SCK frequency of the slave to $\frac{F_{PCLK}}{8}$ or less. The controlling master must also allow the decreased setup time on the slave data out line.

1.6.2 Data Transmission

The sender unit of the SPI Slave module contains a bits counter and a shift register. Based on the transmission settings in the CTRL (1.8.3) register (i.e.: the operation mode – the MODE[1:0] bits; bits order – the MSB bit; data frame length – the FLEN[1:0] bits) and the nSS signal the bits counter monitors which bit is currently on the MISO line. The transmission status bits in the STATUS register (1.8.2) are set according to the currently transmitted bit.

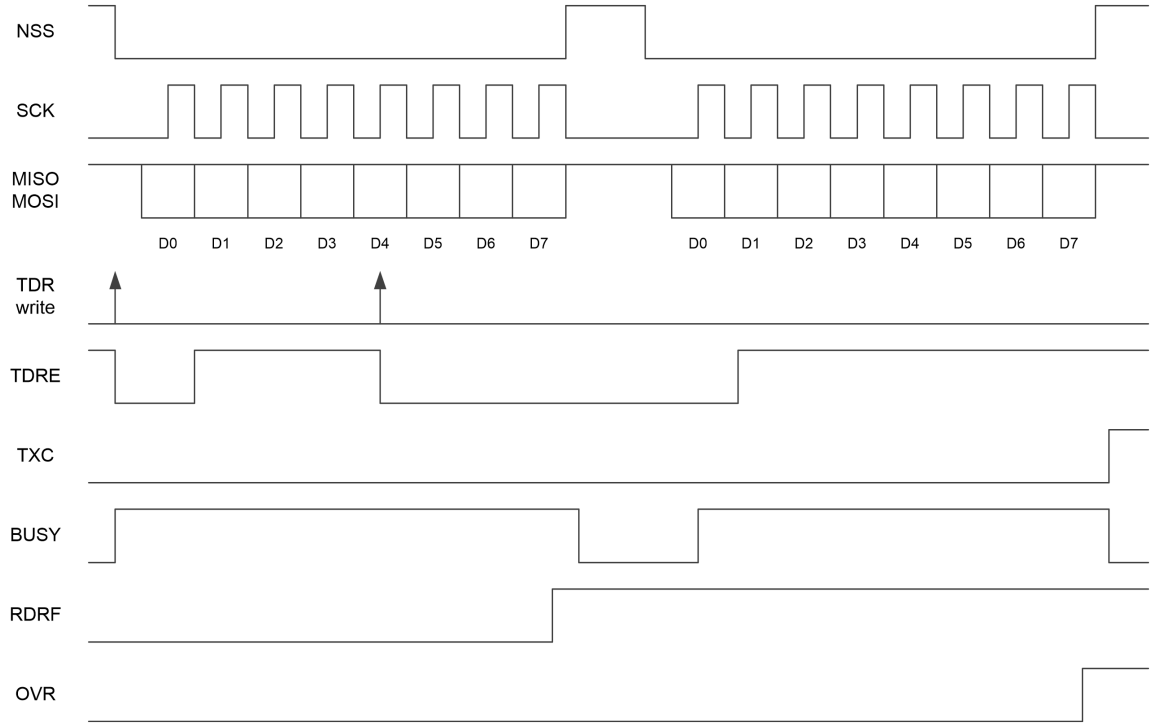


Figure 1.6. Transmission in the SPI Slave mode (MODE 0).

In the MODE 0 and the MODE 2 modes first transmitted bit is taken directly from the TDR register (1.8.4). It is due to the fact that in these modes the first edge is a sampling edge. Therefore, the state of this bit cannot change until the bit is transmitted. After that the MISO line is switched onto the sender's shift register (where TDR register data were previously latched). Accordingly the controller forces this behaviour through controlling the proper state of the TDRE bit (1.8.2) - Figure 1.6.

1.6.3 Data Receiving

The receiver unit of SPI Slave module contains bits counter and shift register. The MOSI line is sampled at the proper edge of the SCK signal and received bit is shifted into the shift register. The RDRF flag of the STATUS register (1.8.2) signals that all received bits of data word were saved in the RDR register. The data word remains in the register until a new data is received. Then the data word will be replaced by the new one. Therefore the data needs to be read by the user program before their replacement. Otherwise, the data will be lost. This will be signaled by the controller by setting the OVR flag (1.8.2).

1.6.4 Slave Select

The SPI controller sends data through the MISO line and samples data at the MOSI line only when the nSS line is in the low state. The high state of the nSS line forces reset of both receiving and sending counters and any edge of the SCK signal is ignored thereafter. Therefore it is recommended to use the nSS line to signal the start and the end of the data transfer. This makes it possible to maintain proper synchronization of counters with the SCK signal.

1.6.5 Pad Output Enable

The figure below presents the behaviour of output enable signal that drives MISO line.

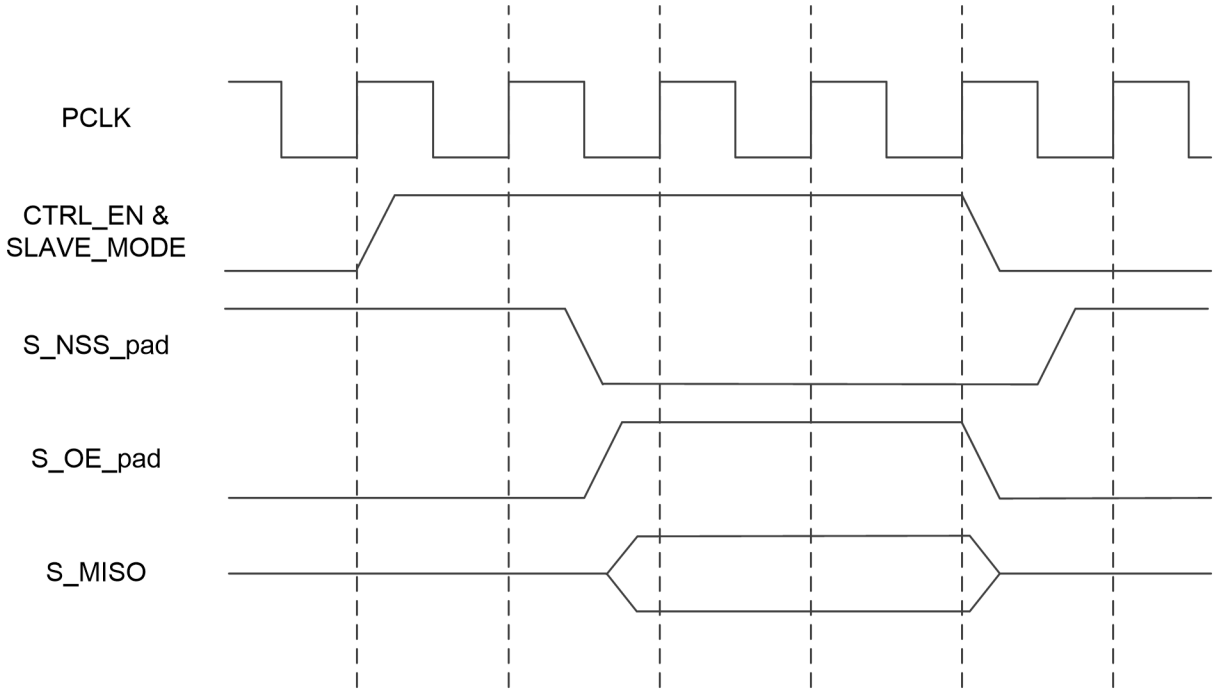


Figure 1.7. SPI Slave pad output enable scheme.

1.7 Interrupts

The SPI controller may rise interrupts according to one or more configurable events.

1.7.1 TXC Interrupt

The interrupt is raised when transmission is complete and there are no new data in the TDR register (1.8.4) - the TDRE bit of the STATUS register (1.8.2) is set “1”. The interrupt is released after reading the STATUS register.

1.7.2 TDRE Interrupt

The interrupt is raised when the data from the TDR register (1.8.4) are loaded into the sender’s shift register. The interrupt line is set low after writing to the TDR register.

1.7.3 RDRF Interrupt

The interrupt is raised when the data from the receiver’s shift register are loaded into the RDR register (1.8.5). The interrupt line is set low after reading the STATUS register (1.8.2).

1.7.4 OVR Interrupt

The interrupt is raised when the data from the receiver's shift register are loaded into the RDR register (1.8.5) and previous data were not read before. The interrupt line is set low after reading the STATUS register (1.8.2).

1.7.5 NSS Rising Edge Interrupt

The NSS Rising Edge interrupt is available only in the Slave mode. The line is set high when the rising edge is detected at the nSS input. The interrupt line is set low after reading the STATUS register (1.8.2).

1.7.6 NSS Falling Interrupt

The NSS Falling interrupt is available only in the Slave mode. The line is set high when the falling edge is detected at the nSS input. The interrupt line is set low after reading the STATUS register (1.8.2).

1.8 Configuration Registers

1.8.1 Registers List

Address Offset	Register	Name
0x00	STATUS	Status Register
0x04	CTRL	Control Register
0x08	TDR	Transmit Data Register
0x0C	RDR	Receive Data Register
0x10	IRQM	Interrupt Mask Register
0x14	IRQMAP	Interrupt Mapping Register
0x18	INFO	Info Register
0x1C	NSS	Slave Select Register

1.8.2 Status Register

Address: 0x00

31	30	...	...	...	...	9	8
		...	...	...	...		
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
NSS	FE	RE	BUSY	OVR	RDRF	TDRE	TXC
R	R	R	R	R	R	R	R
0	0	0	0	0	0	1	0

TXC *Transmission Complete*

0 No data available in the sender's TDR register.

1 The TXC bit is set "1" when data frame transmission completes and the TDR register does not contain new frame to send (the TDRE bit is "1").

The bit is set low after writing new data into the TDR register or reading from the status register.

TDRE *Transmission Data Register Empty*

0 The TDR register contains new data waiting to be transmitted.

1 The TDR register is empty.

The TDRE bit is set "1" after writing data from the TDR register into the sender's shift register. The bit is set low after writing data into the TDR register.

RDRF *Read Data Register Full*

0 The RDR register contains no actual data.

1 The RDR register contains new data ready to be read.

The RDRF bit is set “1” when new data frame was loaded into the RDR register after the transmission was complete.

OVR *Overrun Error*

0 There is no overflow of the receiving buffer.

1 The receiving buffer was overwritten.

The bit is set “1” when new data frame was received before previous data frame in the RDR register was read. The bit is set low after reading the status register.

BUSY *Busy*

0 The module does not transmit data.

1 The module transmits data.

The Master mode: The BUSY bit is set “1” during the data transmission and set low when completes (when the TXC bit is set “1”). The Slave mode: The BUSY bit is set “1” during each frame transmission and set low when each frame transmission completes.

NSS_RE *Rising Edge on nSS*

The bit is set “1” when the rising edge of the nSS line is detected. The bit is set low after reading the status register.

NSS_FE *Falling Edge on nSS*

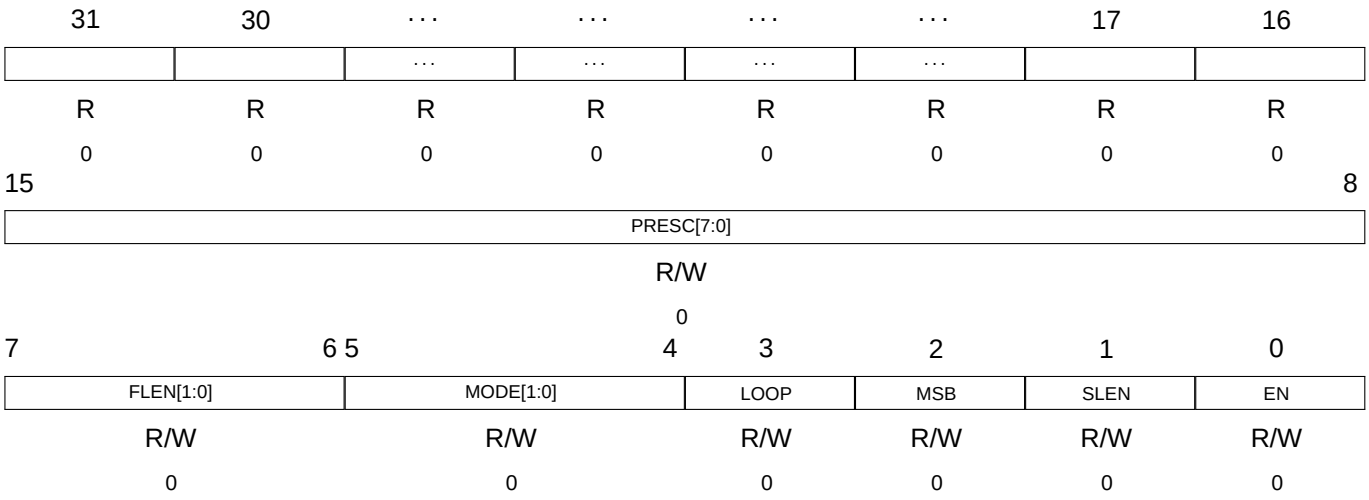
The bit is set “1” when the falling edge of the nSS line is detected. The bit is set low after reading the status register.

NSS *nSS Input State*

The bit value follows the state of the nSS line.

1.8.3 Control Register

Address: 0x04



EN *SPI Enable*

0 The SPI module is inactive. Clock to the module is stopped.

1 The SPI module is active. Clock to the module is supplied.

SLEN *SPI Slave Enable*

0 The module is in Master mode.

1 The module is in Slave mode.

MSB *MSB First*

0 The data frame bits are transmitted starting from least significant bit.

1 The data frame bits are transmitted starting from most significant bit.

LOOP *Loopback Mode Enable*

0 The loopback mode is inactive.

1 The loopback mode is active.

The closed loop mode is available only in the Master mode. The sender register output is directed onto the MOSI line, an IO pad and into the MISO input of the receiver.

MODE[1:0] *Transmission Mode*

Configuration of the transmission mode - the combination of the phase and the polarity of the SCK signal:

MODE[1:0]	Mode	First SCK edge	Second SCK edge
00	MODE 0	raising - bit sampling	falling - bit exposure
01	MODE 1	raising - bit exposure	falling - bit sampling
10	MODE 2	falling - bit sampling	raising - bit exposure
11	MODE 3	falling - bit exposure	raising - bit sampling

FLEN[1:0] *Frame Length*

The data frame length configuration:

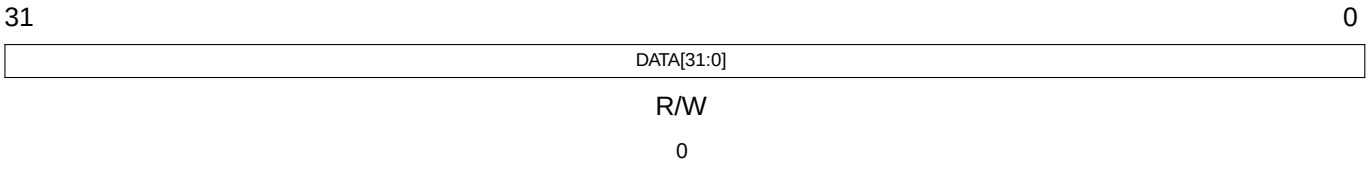
FLEN[1:0]	Frame length
00	8 bits
01	16 bits
10	24 bits
11	32 bits

PRESC[7:0] *SCK Prescaler*

The SCK signal frequency is set according to the equation: $F_{SCK} = \frac{F_{PCLK}}{2 * (PRESC + 1)}$.

1.8.4 Transmit Data Register

Address: 0x08

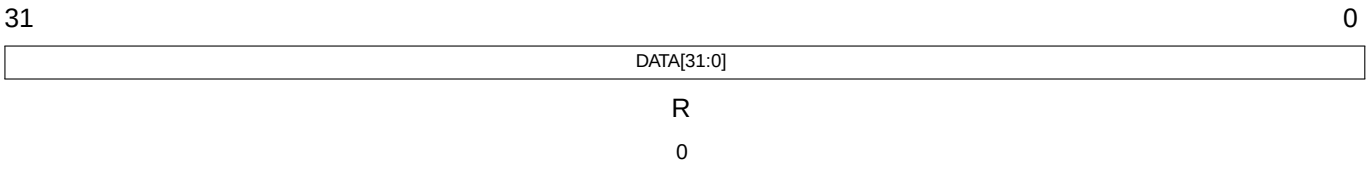


DATA[31:0] *Transmission Buffer*

If the TDRE flag of the STATUS register is 0 then the sender's TDR register contains data not send yet (saving in the register would result in data loss). The length of the frame is set up in the CONTROL register. Data in the TDR register should be aligned to the least significant bit. If the SPI module is in the Master mode and does not conduct any transmission then writing into the TDR register would begin the data communication.

1.8.5 Receive Data Register

Address: 0x0C

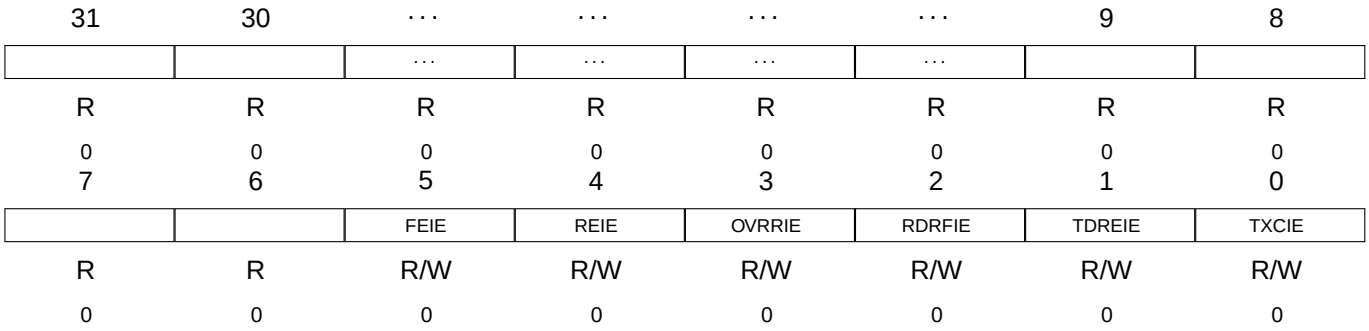


DATA[31:0] *Reception Buffer*

Data received from the external sender are saved in the RDR buffer. If the RDRF flag of the STATUS register is set then the data in the buffer are available. The data frame length is programmed in the CTRL register. Data in the RDR register is aligned with the least significant bit.

1.8.6 Interrupt Mask Register

Address: 0x10



TXCIE *Transmission Complete Interrupt Enable*

0 Disabled interrupt indicating end of frame transfer.

1 Enabled interrupt indicating end of frame transfer.

TDREIE *Transmit Data Register Empty Interrupt Enable*

0 Disabled interrupt indicating no data in the TDR register.

1 Enabled interrupt indicating no data in the TDR register.

RDRFIE *Receive Data Register Empty Interrupt Enable*

0 Disabled interrupt indicating no data in the RDR register.

1 Enabled interrupt indicating no data in the RDR register.

OVRRIE *Overrun Error Interrupt Enable*

0 Disabled interrupt indicating overflow of the receiver's buffer.

1 Enabled interrupt indicating overflow of the receiver's buffer.

REIE *nSS Rising Edge Interrupt Enable*

0 Disabled interrupt indicating rising edge detection of the nSS signal.

1 Enabled interrupt indicating rising edge detection of the nSS signal.

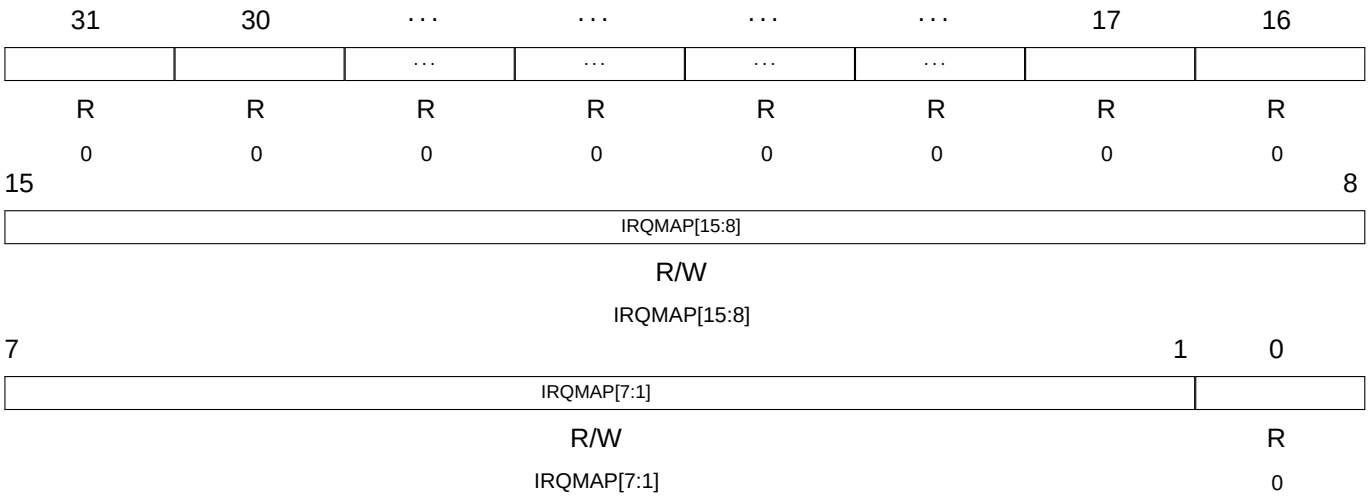
FEIE *nSS Falling Edge Interrupt Enable*

0 Disabled interrupt indicating falling edge detection of the nSS signal.

1 Enabled interrupt indicating falling edge detection of the nSS signal.

1.8.7 Interrupt Mapping Register

Address: 0x14

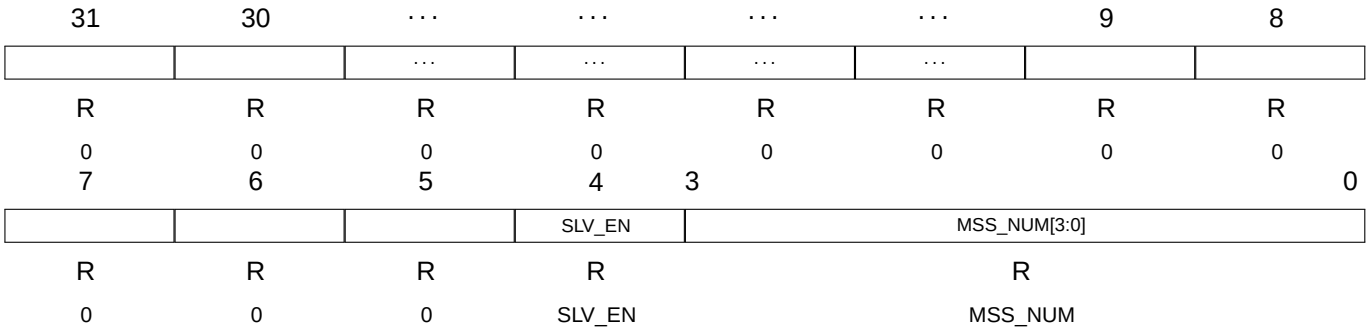


IRQMAP[15:1] *Interrupt Mapping*

Each set bit represents the interrupt number that will be passed to interupt controller. It is allowed to set more than one bit.

1.8.8 Info Register

Address: 0x18



MSS_NUM *Slave Select Number*

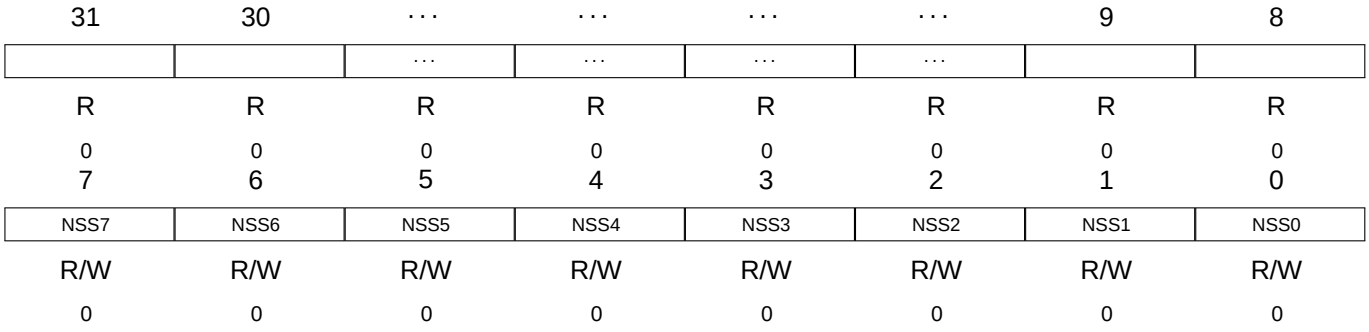
Number of dedicated slave select lines.

SLV_EN *Slave Enable*

Bit indicates if SPI supports slave operations.

1.8.9 Slave Select Register

Address: 0x1C



NSSn *Slave Select n*

Active (high) bit indicates that the corresponding slave select line is active (low). Incorporated hardware prevents from selecting more than one line. In case of conflict the most LBS line will be activated. Register contains only implemented *MSS_NUM* number of bits.

1.9 Implementation

1.9.1 Design Structure

The synthesible RTL IP core part (*AXI/rtl*, *COMMON/rtl* and *SPI/rtl* folder) utilizes Verilog 2005 HDL. The testbench part (*AXI/tb* and *SPI/tb* folder) uses SystemVerilog language.

```
AXI
├── rtl
│   ├── AXI_PERIPH
│   │   └── amba_axilite_apb_bridge.v
└── tb
    ├── AXI_PERIPH
    │   ├── APB
    │   │   └── virtual_APB_slave.sv
    │   ├── AXI
    │   │   └── tb_amba_axilite_tasks.sv
    │   ├── common
    │   │   └── timescale.v
    │   ├── run
    │   │   └── ncvlog_amba_axilite_apb_bridge.sh
    │   ├── tests
    │   │   └── tb_read_write_test.sv
    │   └── tb_amba_axilite_apb_bridge.sv
COMMON
├── rtl
    ├── DFF_en.v
    ├── edge_detector.v
    └── synchronizer.v
SPI
├── beh
├── rtl
    ├── APB_SPI.v
    ├── AXILITE_SPI.v
    ├── SPI_config.v
    ├── SPI_defines.v
    ├── SPI_master.v
    ├── SPI_PDMA_stream_interface.v
    ├── SPI_slave.v
    ├── SPI.v
└── tb
    ├── APB
    │   ├── tb_APB_SPI_init.v
    │   └── tb_APB_SPI_reg_access_tasks.v
    ├── common
    │   ├── tb_SPI_config_tasks.v
    │   ├── tb_SPI_data_transfer_tasks.v
    │   └── timescale.v
    ├── run
    │   ├── irun_apb_spi_all.sh
    │   ├── irun_apb_spi_master.sh
    │   └── irun_apb_spi_slave.sh
    ├── spi
    │   ├── virtual_SPI_master.sh
    │   └── virtual_SPI_slave.sh
```


```
    ├── tests
    │   ├── tb_interrupt_MAPPING_test.sv
    │   ├── tb_MASTER_NSS_test.sv
    │   ├── tb_MASTER_OVR_ERROR_interrupt_test.sv
    │   ├── tb_MASTER_OVR_ERROR_test.sv
    │   ├── tb_MASTER_PDMA_TX_RX_test.sv
    │   ├── tb_MASTER_SpecialCase_test.sv
    │   ├── tb_MASTER_TDRE_interrupt_test.sv
    │   ├── tb_MASTER_TDRE_test.sv
    │   ├── tb_MASTER_TXC_interrupt_test.sv
    │   ├── tb_MASTER_TXC_test.sv
    │   ├── tb_SLAVE_OVR_ERROR_interrupt_test.sv
    │   ├── tb_SLAVE_OVR_ERROR_test.sv
    │   ├── tb_SLAVE_PDMA_TX_RX_test.sv
    │   ├── tb_SLAVE_TDRE_interrupt_test.sv
    │   ├── tb_SLAVE_TDRE_test.sv
    │   ├── tb_SLAVE_TXC_interrupt_test.sv
    │   └── tb_SLAVE_TXC_test.sv
    ├── tb_APB_SPI.sv
    ├── tb_SPI_master.sv
    ├── tb_SPI_sampling_slave.sv
    ├── tb_virtual_SPI_master.sv
    └── tb_virtual_SPI_slave.sv
└── compile.list
```

1.9.2 Simulation Flow

The IP Core is provided with self-checking testbench to verify the correct operation of the IP prior to use in a design. The testbench is divided into two environments. The first one tests the APB_SPI module. To run the simulation using Cadence® Incisive® Enterprise Simulator run *irun_apb_spi_all.sh* script located in the *SPI/tb/run* folder. The simulation should end with reporting no errors. The second environment tests the AXI4-Lite to APB3 converter. To run the simulation using Cadence® Incisive® Enterprise Simulator run *ncvlog_amba_axilite_apb_bridge.sh* script located in the *AXI/tb/AXI_PERIPH/run* folder. The simulation should end with reporting no errors. The AXILITE_SPI top module is composed of the APB_SPI core and the amba_axilite_apb_bridge AXI4-Lite to APB3 converter.

1.9.3 Clock and Reset

The CC-SPI-AXI utilizes a fully synchronous design with one positive edge clocking domain and negative asynchronous reset assertion. External reset synchronizer has to be used to ensure synchronous reset deassertion.

1.9.4 Constraints

In most cases only module output ports are registered. Therefore, it is a good practice to reserve the entire clock cycle for module inputs combinational logic and set minimal input delay (*set_input_delay* command). Registered outputs leave the entire clock cycle for external logic (*set_output_delay* command).

By default external module inputs are synchronized using Synchronizer and Synchronizer_set_en modules located in the synchronizer.v file. If possible, they should be replaced with integrated 2FF synchronizers from the target technology library. Otherwise, max delay (*set_max_delay* command) of 10% to 20% of one destination clock cycle should be set between synchronizer stages. Do not use dynamic FFs to implement synchronizer module.

1.9.5 Configuration Options

The table below shows the generic parameters of the core.

Generic name	Description	Range	Default
PDMA_support	Configure PDMA interface support	0,1	1
slave_support	Configure SPI slave support	0,1	1
default_interrupt_MAPPING	Reset value of interrupt_MAPPING register	0:32767	0
mss_number	Number of slave select pins	0:8	2

The table below shows the define parameters of the core (SPI_config.v file).

Define name	Description	Default
SPI_SLAVE_INPUTS_SYNCHRONIZATION	Comment to remove MOSI, NSS, SCK synchronizing flip flops for SPI slave	defined
SPI_MASTER_INPUTS_SYNCHRONIZATION	Comment to remove MISO synchronizing flip flops for SPI master	undefined

1.9.6 Signals Description

Signal name	Description	I/O	Active	Type
ACLK	Synchronous clock	I	rising	clock
ARESETn	Asynchronous reset	I	low	reset
AWADDR[4:2]	AXI4-Lite write address	I	data	comb.
AWPROT[2:0][1]	AXI4-Lite write address protection type	I	data	comb.
AWVALID	AXI4-Lite write address valid	I	high	comb.
AWREADY	AXI4-Lite write address ready	O	high	reg.
WDATA[31:0]	AXI4-Lite write data	I	data	comb.
WSTRB[3:0]	AXI4-Lite write strobe	I	high	comb.
WVALID	AXI4-Lite write valid	I	high	comb.
WREADY	AXI4-Lite write ready	O	high	reg.
BRESP[1:0]	AXI4-Lite write response	O	data	reg.
BVALID	AXI4-Lite write response valid	O	high	reg.
BREADY	AXI4-Lite write respnse ready	I	high	comb.
ARADDR[4:2]	AXI4-Lite read address	I	data	comb.
ARPROT[2:0][1]	AXI4-Lite read address protection type	I	data	comb.
ARVALID	AXI4-Lite read address valid	I	high	comb.
ARREADY	AXI4-Lite read address ready	O	high	reg.
RDATA[31:0]	AXI4-Lite read data	O	data	reg.
RRESP[1:0]	AXI4-Lite read response	O	data	reg.
RVALID	AXI4-Lite read valid	O	high	reg.
RREADY	AXI4-Lite read ready	I	high	comb.
interrupt_TXC	Transmission complete interrupt	O	high	reg.
interrupt_TDRE	Transmit data register empty interrupt	O	high	reg.
interrupt_RDRF	Read data register full interrupt	O	high	reg.
interrupt_OVR_ERROR	Overrun interrupt	O	high	reg.
interrupt_NSS_RE	NSS rising edge interrupt	O	high	reg.
interrupt_NSS_FE	NSS falling edge interrupt	O	high	reg.
interrupt_MAPPING[15:1]	Interrupt mapping vector	O	data	reg.
Downstream_enable	PDMA downstream enable signal	I	high	comb.
Downstream_busy	PDMA downstream busy signal	O	high	reg.
Downstream_request	PDMA downstream request signal	O	high	reg.
Downstream_ack	PDMA downstream ack signal	I	high	comb.
Downstream_data[31:0]	PDMA downstream data	I	data	comb.
Upstream_enable	PDMA upstream enable signal	I	high	comb.
Upstream_busy	PDMA upstream busy signal	O	high	reg.
Upstream_request	PDMA upstream request signal	O	high	reg.
Upstream_ack	PDMA upstream ack signal	I	high	comb.

Upstream_data[31:0]	PDMA upstream data	O	data	reg.
clock_request	Clock request signal	O	high	reg.
S_SCK_pad	SPI Slave clock	I	rising or falling	reg./comb.[2]
S_MOSI_pad	SPI Slave Master Out/Slave In	I	data	reg./comb.[2]
S_NSS_pad	SPI Slave serial select	I	low	reg./comb.[2]
S_MISO_pad	SPI Slave Master In/Slave Out	O	data	reg.
S_OE	SPI Slave output enable	O	high	comb.[3]
M_MISO_pad	SPI Master Master In/Slave Out	I	data	comb.[4]
M_SCK_pad	SPI Master clock	I	rising or falling	reg.
M_MOSI_pad	SPI Master Master Out/Slave In	O	data	reg.
M_NSS_pad[mss_number:0][5]	SPI Master serial select	O	low	reg.
M_OE	SPI Master output enable	O	high	reg.

1.9.7 Instantiation

```
icg
icg_spi (
    .E(ARVALID|AWVALID|spi_clock_request),
    .clk(ACLK),
    .gclk(spi_clk),
    .scan_enable(scan_enable));

AXILITE_SPI #(
    .slave_support(CFG_SPI_SLV_EN),
    .PDMA_support(CFG_DMA_EN),
    .mss_number(CFG_MSS_NUMBER),
    .default_interrupt_MAPPING(CFG_DEF_INT_MAPPING))
AXILITE_SPI_u (
    .ACLK(spi_clk),
    .ARESETn(ARESETn),
    .AWADDR(AWADDR[4:2]),
    .AWPROT(AWPROT),
    .AWVALID(AWVALID),
    .AWREADY(AWREADY),
    .WDATA(WDATA),
```

[1] Signal is not used in the design.
[2] Depending on SPI_SLAVE_INPUTS_SYNCHRONIZATION setting. Defined value will insert input synchronization flip-flops.
[3] Two-input and-gate with S_NSS_pad and register output.
[4] Depending on SPI_MASTER_INPUTS_SYNCHRONIZATION setting. Defined value will insert input synchronization flip-flops. Raw input is muxed with M_MOSI_pad to form loopback.
[5] The actual usable width of M_NSS_pad signal is [mss_number-1:0]. M_NSS_pad[mss_number] bit is hardcoded as one to avoid generating negative index when mss_number = 0.

```
.WSTRB(WSTRB),
.WVALID(WVALID),
.WREADY(WREADY),
.BRESP(BRESP),
.BVALID(BVALID),
.BREADY(BREADY),
.ARADDR(ARADDR[4:2]),
.ARPROT(ARPROT),
.ARVALID(ARVALID),
.ARREADY(ARREADY),
.RDATA(RDATA),
.RRESP(RRESP),
.RVALID(RVALID),
.RREADY(RREADY),
.S_MISO_pad(SMISO),
.S_SCK_pad(SSCK),
.S_MOSI_pad(SMOSI),
.S_NSS_pad(SNSS),
.S_OE_pad(SPI_SLV_ENABLE),
.M_MISO_pad(MMISO),
.M_SCK_pad(MSCK),
.M_MOSI_pad(MMOSI),
.M_NSS_pad({dummy_MNSS,MNSS[CFG_MSS_NUMBER-1:0]}),
.M_OE_pad(SPI_MST_ENABLE),
.interrupt_TXC(spi_interrupt_TXC),
.interrupt_TDRE(spi_interrupt_TDRE),
.interrupt_RDRF(spi_interrupt_RDRF),
.interrupt_OVR_ERROR(spi_interrupt_OVR_ERROR),
.interrupt_NSS_RE(spi_interrupt_NSS_RE),
.interrupt_NSS_FE(spi_interrupt_NSS_FE),
.interrupt_MAPPING(spi_interrupt_MAPPING),
.Downstream_enable(spi_downstream_enable),
.Downstream_busy(spi_downstream_busy),
.Downstream_request(spi_downstream_request),
.Downstream_ack(spi_downstream_ack),
.Downstream_data(downstream_data),
.Upstream_enable(spi_upstream_enable),
.Upstream_busy(spi_upstream_busy),
.Upstream_request(spi_upstream_request),
.Upstream_ack(spi_upstream_ack),
.Upstream_data(spi_upstream_data),
.clock_request(spi_clock_request));
```

```
assign spi_irq        = spi_interrupt_TXC | spi_interrupt_TDRE |
                        spi_interrupt_RDRF | spi_interrupt_OVR_ERROR |
                        spi_interrupt_NSS_RE | spi_interrupt_NSS_FE;

assign spi_irq_vector = spi_interrupt_MAPPING & {15{spi_irq}};

generate
if (CFG_SEPARATE_PADS == 1) begin

    io_pad_model #(
        .IO_NUM(1))
    msck_pad_model (
        .core_input(1'b0),
        .core_output(MSCK),
        .IO_pad(MSCK_pad),
        .output_enable(SPI_MST_ENABLE));

    io_pad_model #(
        .IO_NUM(1))
    mmosi_pad_model (
        .core_input(1'b0),
        .core_output(MMOSI),
        .IO_pad(MMOSI_pad),
        .output_enable(SPI_MST_ENABLE));

    io_pad_model #(
        .IO_NUM(1))
    mmiso_pad_model (
        .core_input(MMISO),
        .core_output(1'b0),
        .IO_pad(MMISO_pad),
        .output_enable(1'b0));

    io_pad_model #(
        .IO_NUM(CFG_MSS_NUMBER))
    mnss_pad_model (
        .core_input({CFG_MSS_NUMBER{1'b0}}),
        .core_output(MNSS),
        .IO_pad(MNSS_pad),
        .output_enable({CFG_MSS_NUMBER{SPI_MST_ENABLE}}));
```


```
    io_pad_model #(
        .IO_NUM(1))
    ssck_pad_model (
        .core_input(SSCK),
        .core_output(1'b0),
        .IO_pad(SSCK_pad),
        .output_enable(1'b0));

    io_pad_model #(
        .IO_NUM(1))
    smosi_pad_model (
        .core_input(SMOSI),
        .core_output(1'b0),
        .IO_pad(SMOSI_pad),
        .output_enable(1'b0));

    io_pad_model #(
        .IO_NUM(1))
    smiso_pad_model (
        .core_input(1'b0),
        .core_output(SMISO),
        .IO_pad(SMISO_pad),
        .output_enable(SPI_SLV_ENABLE));

    io_pad_model #(
        .IO_NUM(1))
    snss_pad_model (
        .core_input(SNSS),
        .core_output(1'b0),
        .IO_pad(SNSS_pad),
        .output_enable(1'b0));

end else begin

    io_pad_model #(
        .IO_NUM(1))
    sck_pad_model (
        .core_input(SSCK),
        .core_output(MSCK),
        .IO_pad(SCK_pad),
        .output_enable(SPI_MST_ENABLE));
```

```
    io_pad_model #(
        .IO_NUM(1))
    mosi_pad_model (
        .core_input(SMOSI),
        .core_output(MMOSI),
        .IO_pad(MOSI_pad),
        .output_enable(SPI_MST_ENABLE));

    io_pad_model #(
        .IO_NUM(1))
    miso_pad_model (
        .core_input(MMISO),
        .core_output(SMISO),
        .IO_pad(MISO_pad),
        .output_enable(SPI_SLV_ENABLE));

    io_pad_model #(
        .IO_NUM(CFG_MSS_NUMBER))
    mnss_pad_model (
        .core_input({CFG_MSS_NUMBER{1'b0}}),
        .core_output(MNSS),
        .IO_pad(MNSS_pad),
        .output_enable({CFG_MSS_NUMBER{SPI_MST_ENABLE}}));

    io_pad_model #(
        .IO_NUM(1))
    snss_pad_model (
        .core_input(SNSS),
        .core_output(1'b0),
        .IO_pad(SNSS_pad),
        .output_enable(1'b0));

end
endgenerate
```


1.10 Revision History

Doc. Rev.	Date	Comments
1.2	11-2018	Editorial corrections in 1.9.7 Instantiation section.
1.1	12-2017	New functionality introduced. SPI transmit module will delay data transmission until RDRF bit is cleared with both upstream and downstream DMA channels enabled to prevent overflow (1.5.3 Data Receiving section).
1.0	11-2017	First Issue.

ChipCraft Sp. z o.o. Dobrzańskiego 3 lok. BS073, 20-262 Lublin, POLAND www.chipcraft-ic.com

 CC-SPI-AXI-Doc_112018.