Freescale Semiconductor
Application Note

Document Number: AN3904
Rev. 0, 08/2009

MPC5121e Serial Peripheral Interface (SPI)

by: Pavel Boháčik
Rožnov Czech System Center
Czech Republic

1 Introduction

The purpose of this application note is to describe the serial peripheral interface bus controller (SPI) implemented on Freescale's MPC5121e microcontroller. It describes how to configure and use the programmable serial controller (PSC) and PSC centralized FIFO controller (FIFOC) in all supported SPI modes.

1.1 Objective

The objective of this application note is to describe the necessary steps needed to initialize and configure PSC in the SPI mode at all supported modes.

1.2 Definition of the SPI

The Serial Peripheral Interface (SPI) protocol is asynchronous serial data standard, primarily used to allow a microprocessor to communicate with other microprocessors or ICs such as memories, liquid crystal

Contents

diodes (LCD), analog-to-digital converter subsystems, etc.

The SPI is a very simple synchronous serial data, master/slave protocol based on four lines:

- Clock line (SCLK)
- Serial output (MOSI)
- Serial input (MISO)
- Slave select (SS)

Every SPI system consists of one master and one or more slaves, where a master initiates the communication by asserting the SS line. When a slave device is selected, the master starts clocking out the data through the MOSI line to the selected slave device. The master sends and receives one bit for every clock edge. One byte can be exchanged in eight clock cycles. The master finishes communication by de-asserting the SS line.

The SPI is a primitive protocol without an acknowledgement mechanism for checking received or sent data. For safe communication, a flow control has to be implemented in the communications protocol on s a higher level.

2 Description of the SPI module

2.1 SPI module in MPC5121e

The MPC5121e PSC module in SPI mode is capable of master and slave mode as well. The MPC5121e has a centralized FIFO controller that contains data to be transmitted plus the received data for all twelve PSC modules. FIFO is divided into twenty-four slices. For each PSC module, one Tx and one Rx FIFO space is available. The size of each memory slice is fully user-programmable, depending on the free FIFO space. The FIFO slice is able to allocate maximum available memory but the user has to prevent overlay of the individual slices in the memory. The available memory space for all slices together is 32b × 1024 (4 KB).

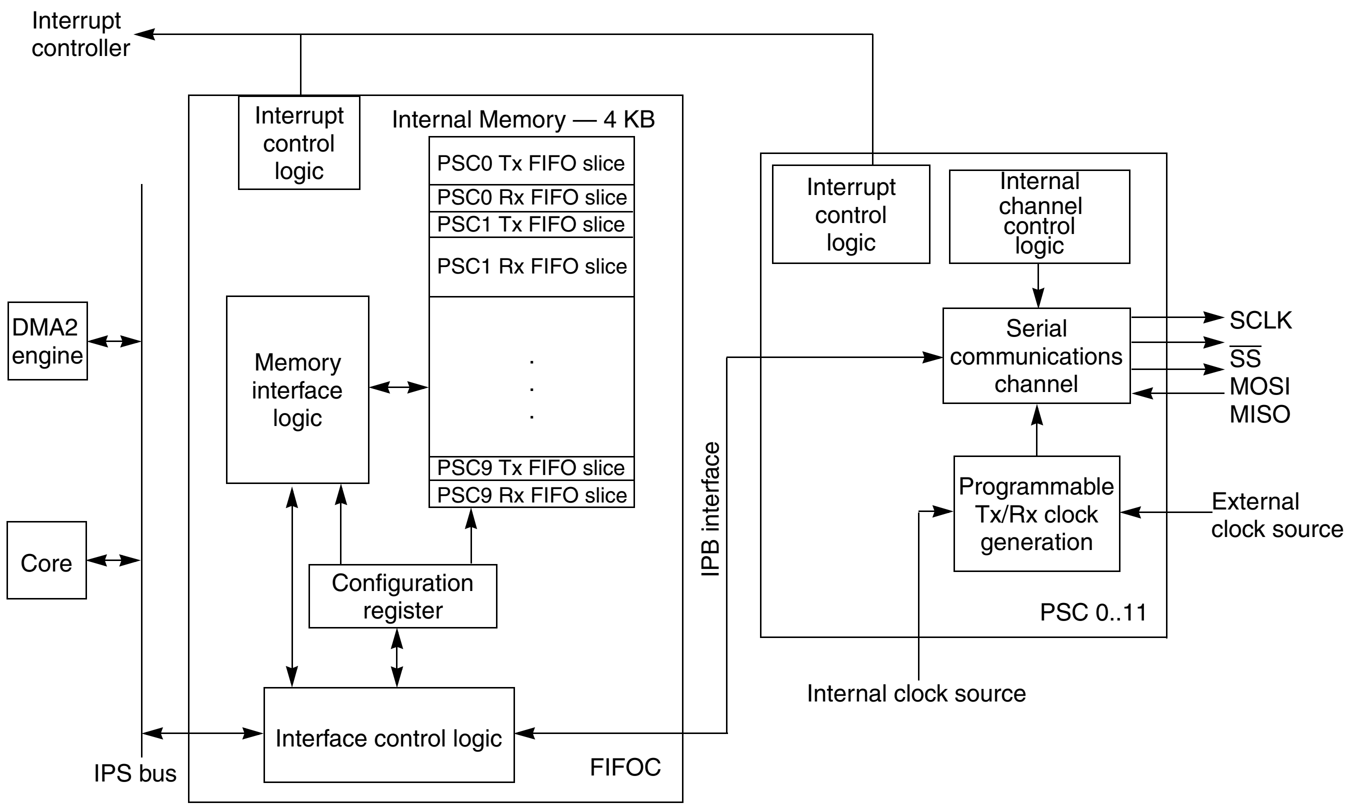

Figure 1. MPC5121e PSC and FIFOC module system interconnection

The PSC requests new data if the Tx shift register is empty or writes Rx data to FIFOC if the Rx shift register is full. This communication is independent of external interrupt or request signals. To make sure the transfer is successful and avoid an overrun/underrun event, both transceiver and receiver must be always enabled and the core/DMA must make sure that the data slices or Tx or Rx shift registers never become full/empty.

2.2 Serial peripheral interface register list

The PSC and FIFOC available in the MPC5121e use these registers for self-configuration and for communication with the connected device. The register address is calculated as the base address for the relevant PSC plus the offset value. Table 1 shows the register list related to the PSC and Table 2 shows the register list related to the FIFOC.

For further information and detail on these registers see these chapters in Freescale document MPC5121ERM, *MPC5121e Microcontroller Reference Manual*:

- Chapter 30, “Programmable Serial Controller”
- Chapter 31, “PSC Centralized FIFO Controller”

Table 1. Register list — PSC

Dress	Register Name	Description
Base Address + 00	Mode Register 1 (MR1)	Controls configuration
Base Address + 00	Mode Register 2 (MR2)	Controls configuration
Base Address + 04	Status Register (SR)	Status of PSC
Base Address + 04	Clock Select Register (CSR)	Default
Base Address + 08	Command Register (CR)	Provides commands to the PSC
Base Address + 0C	Rx Buffer Register (RB)	Reads data directly from the Rx shift register
Base Address + 0C	Tx Buffer Register (TB)	Writes data directly from the Tx shift register
Base Address + 10	Input Port Change Register (IPCR)	Default
Base Address + 10	Auxiliary Control Register (ACR)	Default
Base Address + 14	Interrupt Status Register (ISR)	Status for all potential interrupt sources
Base Address + 14	Interrupt Mask Register (IMR)	Selects corresponding bits in the ISR that cause an interrupt
Base Address + 18	Counter Timer Upper Register (CTUR)	Together with CTLR affects delay after transfer
Base Address + 1C	Counter Timer Lower Register (CTLR)	Together with CTUR affects delay after transfer
Base Address + 20	Codec Clock Register (CCR)	Define DSCKLL delay and SPI baud rate
Base Address + 24	AC97 Slots Register (AC97Slots)	Default
Base Address + 28	AC97 Command Register (AC97CMD)	Default
Base Address + 2C	AC97 Status Data Register (AC97Data)	Default
Base Address + 30	Reserved	Default
Base Address + 34	Input Port Register (IP)	Default
Base Address + 38	Output Port 1 Bit Set (OP1)	Default
Base Address + 3C	Output Port 0 Bit Set (OP0)	Default
Base Address + 40	Serial Interface Control Register (SICR)	Sets the main operation mode

Table 2. Register list — FIFOC

Address		Register Name	Description
11…8	7…0		
Base Address + 0x0*n* (*n* = PSC number)	0x80	Command register for PSC*n* Tx slice — PSC*n*_Tx_CMD	Provides commands to the FIFOC
	0x84	Alarm level for PSC*n* Tx slice — PSC*n*_Tx_ALARM	Defines alarm level
	0x88	Status register for PSC*n* Tx slice — PSC*n*_Tx_SR	Shows internal status of the FIFO slice
	0x8C	Interrupt status register for PSC*n* Tx slice — PSC*n*_Tx_ISR	Status of all potential interrupts
	0x90	Interrupt mask register for PSC*n* Tx slice — PSC*n*_Tx_IMR	Selects corresponding bits in the ISR that cause an interrupt
	0x94	FIFO count for PSC*n* Tx slice — PSC*n*_Tx_COUNT	Number of bytes in the FIFO
	0x98	FIFO pointer for PSC*n* Tx slice — PSC*n*_Tx_POINTER	Reads or modifies pointer in the FIFO slice
	0x9C	FIFO size register for PSC*n* Tx slice — PSC*n*_Tx_SIZE	Sets start address and size of the FIFO slice
	0xBC	FIFO data register for PSC*n* Tx slice — PSC*n*_Tx_DATA	FIFO data register
	0xC0	Command register for PSC*n* Rx slice — PSC*n*_Rx_CMD	Provides commands to the FIFOC
	0xC4	Alarm level for PSC*n* Rx slice — PSC*n*_Rx_ALARM	Defines alarm level
	0xC8	Status register for PSC*n* Rx slice — PSC*n*_Rx_STAT	Shows internal status of the FIFO slice
	0xCC	Interrupt status register for PSC*n* Rx slice — PSC*n*_Rx_INTSTAT	Status of all potential interrupts
	0xD0	Interrupt mask register for PSC*n* Rx slice — PSC*n*_Rx_INTMASK	Selects corresponding bits in the ISR that cause an interrupt
	0xD4	FIFO count for PSC*n* Rx slice — PSC*n*_Rx_COUNT	Number of bytes in the FIFO
	0xD8	FIFO pointer for PSC*n* Rx slice — PSC*n*_Rx_POINTER	Reads or modifies pointer in the FIFO slice
	0xDC	FIFO size register for PSC*n* Rx slice — PSC*n*_Rx_SIZE	Sets start address and size of the FIFO slice
	0xFC	FIFO data register for PSC*n* Rx slice — PSC*n*_Rx_DATA	FIFO data register
Base Address + 0xF00		FIFO command	Default
Base Address + 0xF04		FIFO interrupt status	Shows all PSCs with currently pending interrupts

Table 2. Register list — FIFOC (continued)

Address		Register Name	Description
11…8	7…0		
Base Address + 0xF08		FIFO DMA request	Shows all PSCs with currently pending requests
Base Address + 0xF0C		FIFO AXE request	Default
Base Address + 0xF10		FIFO debug	Default

2.3 Signal description and connection scheme

The serial peripheral interface bus has four external lines, described in Table 3. Figure 2 shows how the slave device is connected to the master in the single master, single slave SPI implementation. Figure 3 shows single master, multiple slave SPI implementations. Multiple slave SPI implementation is not supported by the MPC5121e without additional external hardware.

Table 3. Signal description

Signal	Description
SCLK	Serial clock signal with direction from master to slave device.
MOSI/SIMO	Master-out/slave-in signal for data transmission from master to slave in 8-bit, 12-bit, 16-bit, 20-bit, 24-bit, or 32-bit width (output from master).
MISO/SOMI	Master-in/slave-out signal for data transmission from slave to master in 8-bit, 12-bit, 16-bit, 20-bit, 24-bit, or 32-bit width (output from slave).
SS	Slave select signal, initiated by master to select slave device.

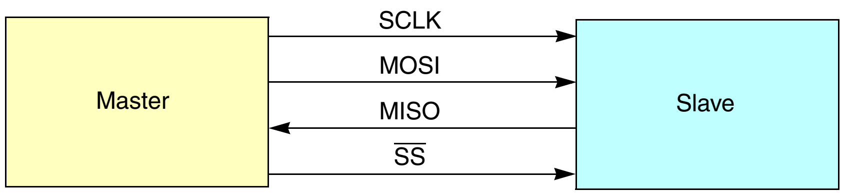

Figure 2. Single master, single slave SPI implementation

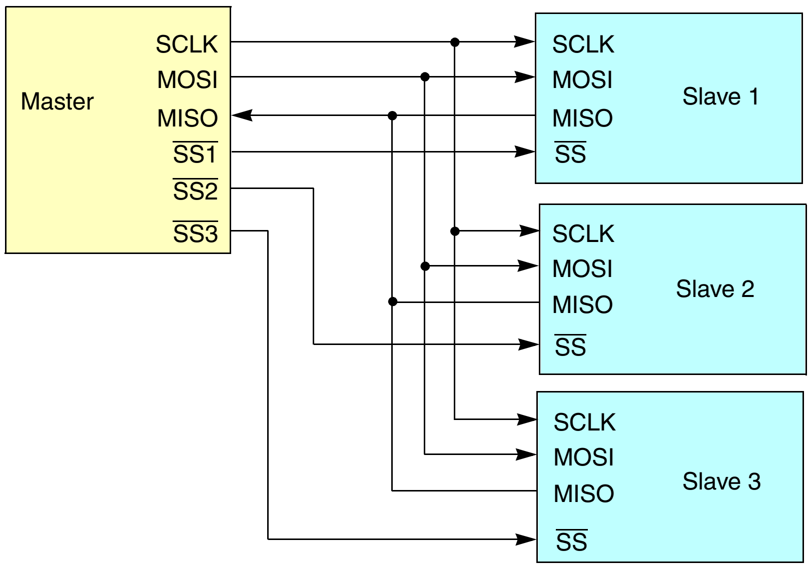

Figure 3. Multiple slave SPI implementations

3 Initialization

To assure proper PSC-SPI module functionality, each module must be initialized before usage. When initializing the PSC-SPI module, the user must obey these instructions.

3.1 PIN muxing

External signals of the MPC5121e are grouped (for a pinout diagram showing pin numbers and a listing of all the electrical and mechanical specifications, refer to Freescale document MPC5121e, *MPC5121e Data Sheet,* at www.freescale.com). Any functionality which is not the primary function is multiplexed. For more detail see Freescale document MPC5121ERM, *MPC5121e Microcontroller Reference Manual*, chapter 3, “Signal Descriptions.”

When the user wishes to use MPC5121e and its PSC*n* pins, it is necessary to set up I/O control registers. The I/O control block controls the functional muxing and configuration of the pads. Configurable parameters include slew rate, Schmitt-trigger input, internal pulldown/pullup, and PCI hold timing.

For proper initialization it is necessary to set up I/O control GP and PAD registers.

For further information on how to set up these registers see these chapters in Freescale document MPC5121ERM, *MPC5121e Microcontroller Reference Manual*:

- Chapter 3, “Signal Descriptions”
- Chapter 22, “IO Control”

3.2 IPS bus

When setting up a system with the MPC5121e, proper clocking must be assured. One of the important clock sources is the IPS bus clock ($IPS\_CLK_{Frequency}$). The IPS bus clock is used as a source for the clock generation unit CSR and CT registers. This clock can be as high as 83 MHz.

3.3 MCLK frequency

Each PSC can select from multiple clock sources. A single clock input is provided that allows the PSC_MCLK_IN to be used as a master clock reference by all PSCs. The MCLK frequency source is the main clock source which is used as the peripheral clock source. Figure 4 shows the circuit which is replicated for each PSC individually.

Many peripheral clocks may be disabled to reduce power consumption in the system clock control register (SCCR1, SCCR2). If the user wishes to operate PSC in the SPI mode, the user must enable clocks for the related PSC and FIFOC in the SCCR1 register.

When PSC and FIFOC clocks are enabled, it is necessary to set up the PSC*n* clock control (PnCCR) register, which controls:

- PSC*n* MCLK divider ratio (PSCn_MCLK_DIV) — the divider determines the F_{MCLK_Out} frequency, which is used for example for calculating the SPI clock (SCLK) frequency. This value can only be changed when the value of MCLK_EN = 0. Two examples of the F_{MCLK_Out} are mentioned in Equation 1 and Equation 4.
- PSC*n* MCLK divider enable (MCLK0_EN) — enable/disable PSC*n* divider
- PSC*n* MCLK divider source (PSCn_MCLK_0_SRC) — define MCLK divider source (SYS_CLK, REF_CLK, PSC_MCLK_IN, SPDIF_TxCLK)
- PSC*n* MCLK source (PSCn_MCLK_1_SRC) — define MCLK source (MCLK_DIV, SPDIF_RxCLK)

PnCCR register sets all parameters shown in Figure 4.

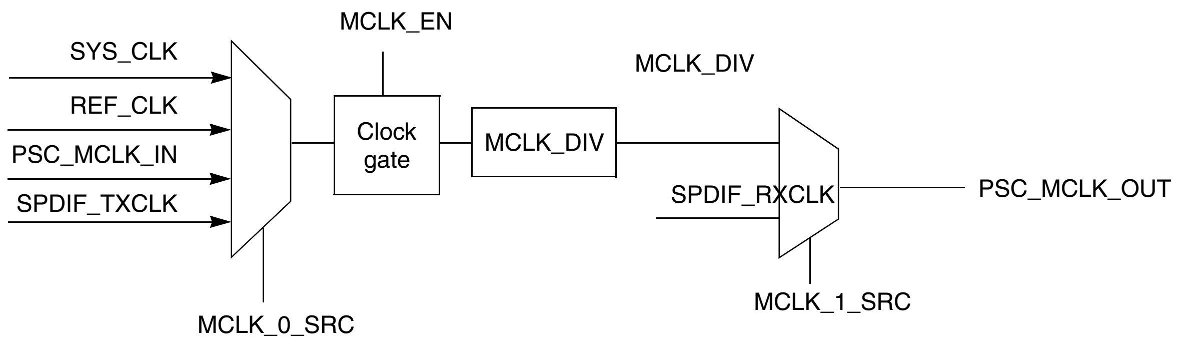

Figure 4. PSC (MCLK) clock generation

Equation 1 and Equation 2 show the calculation of a 1 MHz SCLK frequency derived from the SYS_CLK frequency.

$$PSC\_MCLK\_OUT = \frac{SYS\_CLK}{MCLK\_DIV + 1} = \frac{400\text{ MHz}}{5 + 1} = 66.67\text{ MHz} \le IPS\_CLK\ (83\text{ MHz}) \qquad \textbf{Eqn. 1}$$

$$SCLK_{Frequency} = \frac{PSC\_MCLK\_OUT}{BCLK\_DIV + 1} = \frac{66.67\text{ MHz}}{66 + 1} = 0.995\text{ MHz} \qquad \textbf{Eqn. 2}$$

CAUTION

IPS frequency must be greater than or equal to PSC_MCLK_OUT frequency.

The maximum SCLK frequency for SYS_CLK input frequency can be easily calculated as 20 MHz — see Equation 3 and Equation 4.

$$PSC\_MCLK\_OUT = \frac{SYS\_CLK}{MCLK\_DIV + 1} = \frac{400MHz}{4 + 1} \equiv 80MHz \le IPS\_CLK(83MHz) \qquad \textbf{Eqn. 3}$$

$$SCLK_{Frequency} = \frac{PSC\_MCLK\_OUT}{BCLK\_DIV + 1} = \frac{80MHz}{3 + 1} = 20MHz \qquad \textbf{Eqn. 4}$$

NOTE

The PSC:CCR register contains BCLKDiv (bit clock divider) information. Note that BCLKDiv bytes in CCR register are swapped.

NOTE

Minimum BCLKDiv for SPI mode equals 3.

3.4 Data polling, interrupts, and DMA

The user can chose how to get the data from the memory to the FIFOC slice or how to put data in the memory from the FIFOC slice. The user can also choose between active polling, interrupts, and DMA. The main differences, advantages, and disadvantages are:

- Active polling mode — core periodically checks status of Tx and Rx FIFO slices and writes data to Tx FIFO slice or reads data from Rx FIFO slice.

  Disadvantages — core loading, FIFO slice status does not show the actual status of the serial shift registers due to a small delay (a few processor ticks). It can happen that data is received in the Rx shift register and Rx FIFO slice shows as empty. This can cause small difficulties at the end of the communication.
- Interrupt mode — lower core loading compared to active polling mode. The core is not used for periodical status checks of the Tx and Rx FIFO slices, but it is used for handling interrupt events (sending and receiving data). The number of interrupts depends on the user application. Usage of FIFOC alarm level interrupts can decrease the number of requested interrupts.
- DMA mode — the lowest core loading compared with active polling or interrupt mode. DMA is used for transfer data (array) from memory to the FIFO slice. The DMA engine takes care of the FIFOC status as well. The core is used to start Rx and Tx DMA tasks and to handle interrupts signaling the end of transfer.

Disadvantages — end of frame mode is not supported in DMA mode. The last data packet has to be sent using polling or interrupt mode. All data has to be pre-prepared in the array/buffer before the DMA transfer starts.

The following sections show how to initialize each one of these modes.

3.5 PSC and FIFOC initialization

The MPC5121e has a centralized FIFO controller that contains data to be transmitted plus received data for all twelve PSC modules. The FIFO is divided into twenty-four slices. For each PSC module, one Tx and one Rx FIFO space is available. The size of each memory slice is fully user-programmable, depending on the free FIFO space. The FIFO slice is able to allocate maximum available memory but the user has to prevent overlay of the individual slices in the memory. The available memory space for all slices together is 32b × 1024 (4 KB).

If the user wishes to transfer data with the use of the SPI, then the programmable serial controller must be initialized in SPI mode and the FIFOC must be initialized as well.

3.5.1 PSC initialization

Several steps are necessary for proper initialization of the PSC module in the SPI mode. These steps have to be done in the proper sequence as shown in Figure 5 (master) and Figure 6 (slave).

1. Disable receiver and transmitter — disable the Tx and Rx part in the PSC-CMD configuration register, if the PSC was enabled earlier.
2. Set the main operation mode — set the main operation mode in the Serial Interface Control Register (SICR). For details see Table 4.
3. Define the delay before SCLK (DSCKLL) (not required for slave mode) — when the PSC is in SPI mode (SICR[SPI] = 1), the FrameSyncDiv divider is used to determine the length of time the PSC delays after SS goes low/active before the first SCKL transition of the serial transfer. The delay before SCKL depends on the connected slave device. The delay before SCKL (DSCKLL) can for example be set to 0.5 µs. Equation 5 determines the actual delay before SCKL and an example of the calculation as well.

$$DSCKL_{Delay} = \frac{FrameSyncDiv + 1}{MCLK_{Frequency}} = \frac{32 + 1}{66.6 \times 10^6} = 0.5\mu s \qquad \textbf{Eqn. 5}$$

4. Define the divider for the bit clock generation (not required for slave mode) — in SPI master mode, the bit clock frequency BCLK (SCKL) is generated by dividing the MCLK frequency. In addition to the BCLK generation, the DSCLK delay and the DTL delay must be defined. SCKL is generated internally by dividing the MCLK frequency, as determined by Equation 6. This equation contains the example for $\mathrm{SCKL}_{\mathrm{Frequency}}$ = 1 MHz.

$$SCK_{Frequency} = \frac{MCLK_{Frequency}}{BCLKDiv + 1} = \frac{66.6 \times 10^6}{66 + 1} = 0.99MHz \qquad \textbf{Eqn. 6}$$

NOTE

PSC:CCR register contains BCLKDiv (bit clock divider) information. Note that BCLKDiv bytes in the CCR register are swapped.

5. Set delay after transfer (not required for slave mode) — the delay between consecutive transfers is created by dividing the IPS_CLK clock frequency. The delay after transfer is usually set to 2 µs. Equation 7 determines the actual delay after transfer.

$$DTL = \frac{CT + 2}{IPS\_CLK_{Frequency}} = \frac{3}{MCLK_{Frequency}} = \frac{132 + 2}{66.6 \times 10^6} = 2\mu s \qquad \textbf{Eqn. 7}$$

where

CT[0:7] = CTUR[0:7]

CT[8:15] = CTLR [0:7]

6. Reset mode registers PSC-MR1 and PSC-MR2 — mode register 1 and mode register 2 must be set to 0 before SPI communication.
7. Enable receiver and transmitter — enable the Tx and Rx part in the PSC-CMD register.

Table 4. Serial interface control register used in SPI mode

SICR bitfield	Description
SHDIR	Shift direction
SIM	PSC operation mode (8-bit, 16-bit, and 32-bit mode is supported in SPI)
GenClk	Generate bit clock and FrameSync — this bit must be set to enable the SPI master mode and must be cleared to enable SPI slave mode.
ClkPol	Bit clock polarity
SPI	PSC behaves like an SPI
MSTR	SPI master mode Takes effect only when bit SICR[SPI] equals 1. Also, the GenClk bit must be set to enable the clock generation behavior of the SPI master mode. MSTR bit is used to define master or slave SPI mode.
CPOL	SPI clock polarity Takes effect only when bit SICR[SPI] equals 1. This bit selects an inverted or non-inverted SPI clock.
CPHA	SPI clock phase This bit is used to shift the SCKL serial clock.
UseEOF	Use end-of-frame flag Takes effect only when bit SICR[SPI mode] equals 1. Multiple bytes are transferred while maintaining SS low, up to and including the next byte read from the Tx FIFO that has its EOF flag set.
EN_OutBuf	Enable output buffer The output logic can be enabled by setting this bit or by the Command Register (CR). After setting this bit, the internal generated signals are visible on the output of the device.

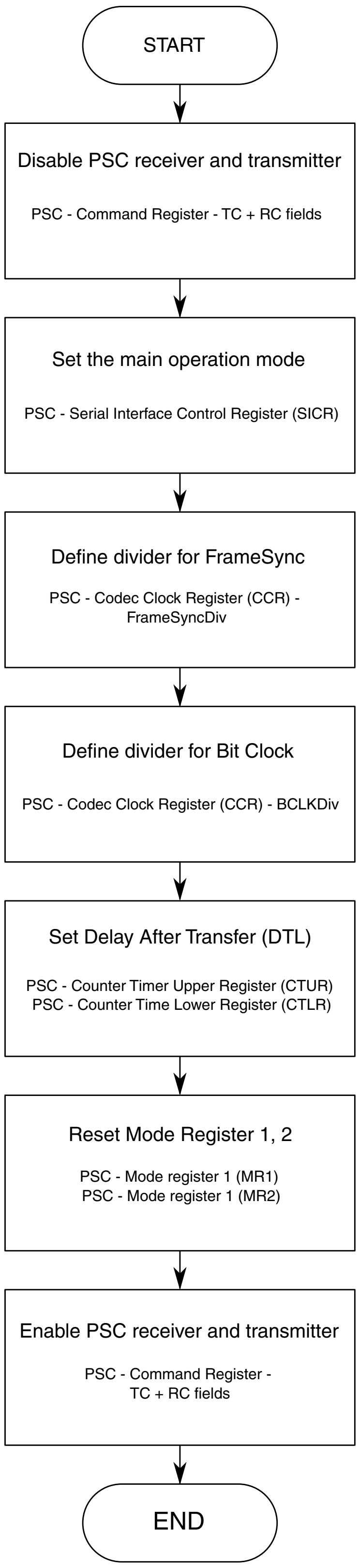
START
Disable PSC receiver and transmitter
PSC - Command Register - TC + RC fields
Set the main operation mode
PSC - Serial Interface Control Register (SICR)
Define divider for FrameSync
PSC - Codec Clock Register (CCR) - FrameSyncDiv
Define divider for Bit Clock
PSC - Codec Clock Register (CCR) - BCLKDiv
Set Delay After Transfer (DTL)
PSC - Counter Timer Upper Register (CTUR)
PSC - Counter Time Lower Register (CTLR)
Reset Mode Register 1, 2
PSC - Mode register 1 (MR1)
PSC - Mode register 1 (MR2)
Enable PSC receiver and transmitter
PSC - Command Register - TC + RC fields
END

Figure 5. PSC SPI master initialization

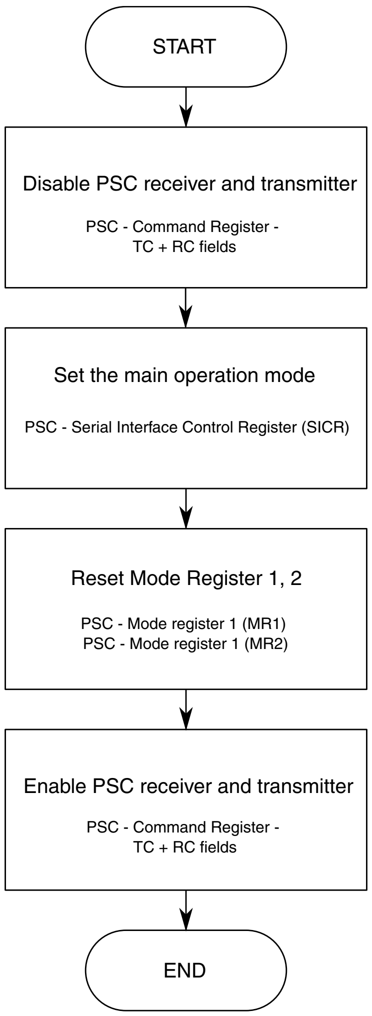
START
Disable PSC receiver and transmitter
PSC - Command Register -
TC + RC fields
Set the main operation mode
PSC - Serial Interface Control Register (SICR)
Reset Mode Register 1, 2
PSC - Mode register 1 (MR1)
PSC - Mode register 1 (MR2)
Enable PSC receiver and transmitter
PSC - Command Register -
TC + RC fields
END

Figure 6. PSC SPI slave initialization

3.5.2 FIFOC initialization

Several steps are necessary for proper initialization of the FIFOC module as well. These steps have to be done in the proper sequence as shown in Figure 8. For a better understanding of the process of setting FIFOC parameters, see Figure 7.

1. Set the start address for Tx/Rx FIFO slice — PSC-FIFOC:Size register defines the start address for the transmitter's and receiver's FIFO slices. The user must take care not to allow overlap in the memory areas to appear. The start address of the FIFO memory is 0x0. The PSC, CPU, and DMA can access this memory, but it is reserved for the PSC's FIFO, and users cannot store their own data or code there.
2. Set the size for Tx/Rx FIFO slice — PSC-FIFOC:Size register defines the size of the FIFO slice as well. Several slices can be located in the FIFO memory area. User software defines the size of the FIFO slices depending on the usage current slice.

NOTE

The allocated area does not depend on the defined transfer mode — in other words, it does not depend on whether 8-, 16-, or 32-bit data will be stored in the FIFO slice. When the user defines the size of the slice, always define $n \times 32$-bit words.

3. Set an alarm level for the Tx/Rx FIFO slice — the alarm level defines the number of data bytes in the FIFO when the alarm status appears. For the Tx FIFO area, the alarm status appears if the amount of data in the Tx FIFO is below this alarm level. For the Rx FIFO area, the alarm status appears if the amount of data in the Rx FIFO is above this alarm level. The alarm level and the request lines deassert if the amount of data crosses this level again.
4. Reset Tx/Rx FIFO slice — The reset FIFO slice command mainly clears all data in the FIFO slice and resets the slice's internal pointer. It also clears the underrun, overrun, and memory access error bits.

NOTE

If the reset FIFO slice command is used, then the FIFO slice becomes disabled.

5. Enable Tx/Rx FIFO slice — Enable FIFO slice command enables the slice. The FIFO controller provides the data for the transmitter and stores the data from the receiver. If the size of the FIFO slice is zero, it is not possible to enable the FIFO.

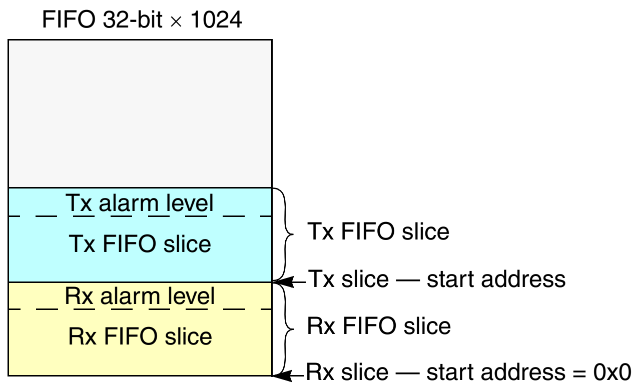

Figure 7. FIFOC setting parameters

NOTE

For better debugging of user code, the internal pointer register FIFOC:PTR is very useful. This register shows the current write and read positions in the FIFO memory without the offset for the slice number, and is writable in debug mode.

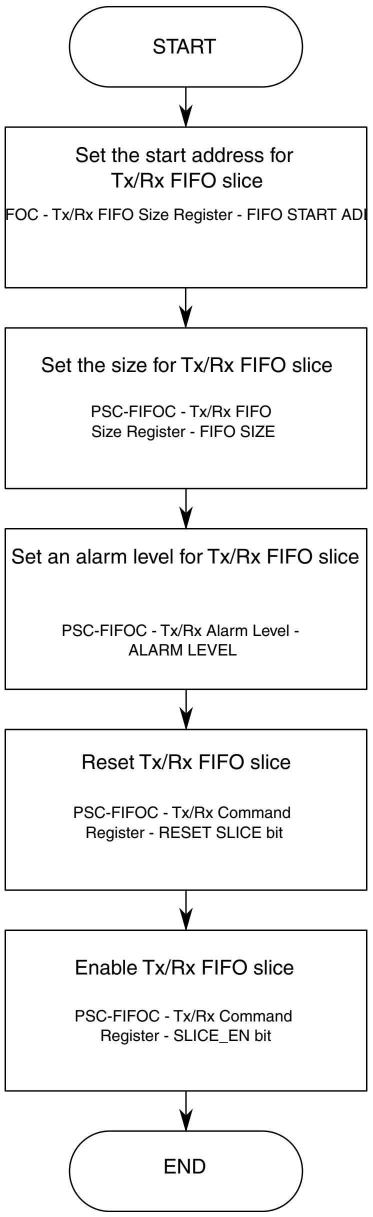
START
Set the start address for Tx/Rx FIFO slice
FOC - Tx/Rx FIFO Size Register - FIFO START ADI
Set the size for Tx/Rx FIFO slice
PSC-FIFOC - Tx/Rx FIFO Size Register - FIFO SIZE
Set an alarm level for Tx/Rx FIFO slice
PSC-FIFOC - Tx/Rx Alarm Level - ALARM LEVEL
Reset Tx/Rx FIFO slice
PSC-FIFOC - Tx/Rx Command Register - RESET SLICE bit
Enable Tx/Rx FIFO slice
PSC-FIFOC - Tx/Rx Command Register - SLICE_EN bit
END

Figure 8. FIFOC SPI initialization

For further information on how to set up FIFOC and PSC in the SPI mode see these chapters in Freescale document MPC5121ERM, *MPC5121e Microcontroller Reference Manual*:

- Chapter 30, "Programmable Serial Controller"
- Chapter 31, "PSC Centralized FIFO Controller"

3.6 Interrupt initialization

If the user wishes to use interrupts instead of polling mode, then the ability to pass external interrupts to the core must be enabled. The user must also enable interrupt from the FIFO controller in the internal programmable interrupt controller (IPIC). Specific interrupt sources must be enabled in the Interrupt Mask register (IMR) in the FIFO controller. Figure 9 shows that initial sequence. The interrupts are used to detect:

- MEM ERROR — memory access error (access to the data register)
- DATA READY — data ready status
- ORERR — overrun error
- URERR — underrun error
- ALARM — FIFO alarm
- FULL — FIFO full
- EMPTY — FIFO empty

Interrupts are enabled by writing one to the proper position in the interrupt mask register in the FIFO host controller and also in the IPIC.

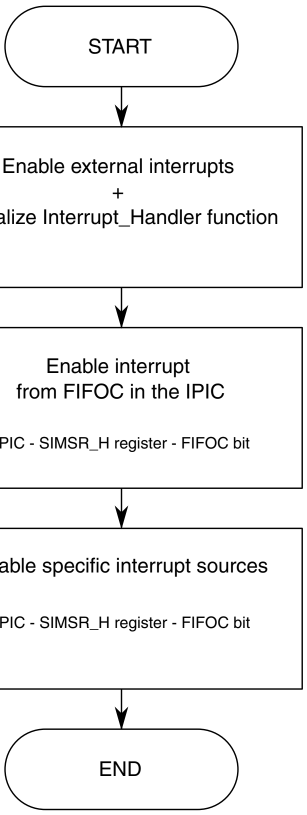
START
Enable external interrupts
+
Initialize Interrupt_Handler function
Enable interrupt
from FIFOC in the IPIC
IPIC - SIMSR_H register - FIFOC bit
Enable specific interrupt sources
IPIC - SIMSR_H register - FIFOC bit
END

Figure 9. SPI interrupt mode initialization

For an example and description of how to use SPI in the interrupt mode, see Section 4.4, "Interrupt mode."

For further information on how to enable and handle interrupts, see these chapters in Freescale document MPC5121ERM, *MPC5121eMicrocontroller Reference Manual*:

- Chapter 20, "Integrated Programmable Interrupt Controller"
- Chapter 30, "Programmable Serial Controller"
- Chapter 31, "PSC Centralized FIFO Controller"

3.7 Direct memory access

If the user wishes to use direct memory access (DMA) instead of polling mode or interrupt mode, several steps are necessary for proper DMA task initialization. These steps are given in the section below and in Figure 10 as well.

- Enable external interrupts to be passed to the core.
- Enable interrupt from the DMA2 in the internal programmable interrupt controller (IPIC) and enable the request to the DMA engine in the IPIC:SIMSR register. The FIFO controller generates a request for this slice to the DMA engine if this slice is enabled and the current data pointer reaches the alarm level.
- Fill the transfer control descriptor (TCD). Each DMA channel requires a 32-byte transfer control descriptor for defining the desired data movement operation.
- Clear the specified channel's DONE bit in its TCD. The DMACDNE register provides a simple memory-mapped mechanism to perform this operation.
- Enable the specific interrupt sources in the TCD (TCDn-Word 7, INT_Maj) in the DMA module. It enables an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the DMAINT register when the current major iteration count reaches zero.

DMA initialization is shown in Figure 10.

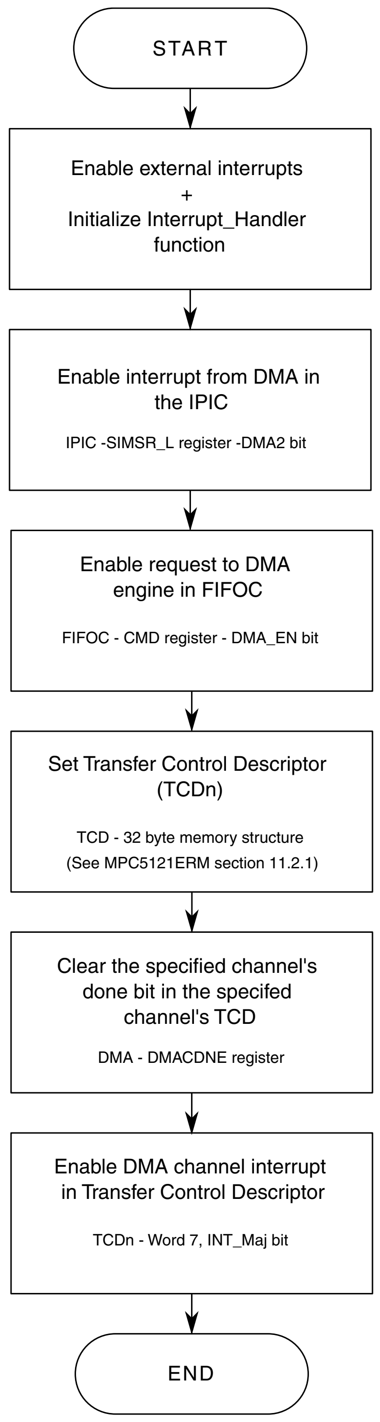
START
Enable external interrupts
+
Initialize Interrupt_Handler function
Enable interrupt from DMA in the IPIC
IPIC -SIMSR_L register -DMA2 bit
Enable request to DMA engine in FIFOC
FIFOC - CMD register - DMA_EN bit
Set Transfer Control Descriptor (TCDn)
TCD - 32 byte memory structure
(See MPC5121ERM section 11.2.1)
Clear the specified channel's done bit in the specifed channel's TCD
DMA - DMACDNE register
Enable DMA channel interrupt in Transfer Control Descriptor
TCDn - Word 7, INT_Maj bit
END

Figure 10. SPI DMA mode initialization

For further information on how to enable interrupts and handle interrupts and enable DMA see these chapters in Freescale document MPC5121ERM, *MPC5121eMicrocontroller Reference Manual*:

- Chapter 11, “Direct Memory Access”
- Chapter 20, “Integrated Programmable Interrupt Controller”

- Chapter 30, “Programmable Serial Controller”
- Chapter 31, “PSC Centralized FIFO Controller”

4 Modes of operation

This section describes the operation of the SPI, focusing on:

- Features
  - PSC bit width
  - End of frame mode
  - Bit clock polarity, clock phase and polarity
  - Shift direction
- Mode of operation
  - Polling mode (master and slave)
  - Interrupt mode (master and slave)
  - DMA mode (master and slave)

All PSCs operates in master and slave mode and support a full duplex SPI mode. Master and slave modes support polling, interrupt, and DMA mode as well. For the proper functionality perform the steps given in Section 4.1, “SPI configuration,” Section 4.2, “Master mode,” Section 4.3, “Slave mode,” and Section 4.4, “Interrupt mode.”

4.1 SPI configuration

4.1.1 PSC operation mode

All PSCs support 8-, 16-, and 32-bit PSC operational mode for SPI mode. Also 12-, 20-, and 24-bit PSC operational modes are supported for SPI mode. When 20-bit operational mode is used, the user writes 32-bit data to the Tx FIFO slice. The SPI module sends the first 20 bits only — the second part of the word is cut. A similar situation occurs when the 12-bit or 24-bit PSC operational mode is used.

NOTE

When 8-bit operational mode is selected and 32-bit data is written to the Tx FIFO slice, the SPI module sends this data in 4 × 8-bit packets. When 4 × 8-bit data is received by the Rx FIFO slice, the user may use 32-bit read access to read data in one command. When 5 × 8-bit data is received, the user has to avoid FIFO slice underrun and may use 32-bit read access once only. The rest of the data may be read by 8-bit access.

CAUTION

When the operating mode is changed, all Rx/Tx and error statuses are reset. Rx and Tx are disabled.

4.1.2 End of frame mode

EOF mode is supported to enable the user to transfer an increased quantity of data in one frame. When using EOF mode the PSC reads data out of the Tx FIFO, the SS signal is driven low (asserted), and the SPI continues to transmit. The last data of the frame is sent when the EOF flag is set. This presence of the EOF flag shows that the last data in the frame has been reached. The EOF flag is set before the last data is written to the Tx FIFO slice. The FIFO controller automatically sends this information to the Tx shift register and stops the transfer after the data is sent.

EOF mode should not be used for single data transfers, and is limited by data packages which must be bigger than the chosen codec size. When using EOF mode, the next data EOF frame flag should not be used to toggle the SS signal after every individual transfer. If the SPI is used to transmit data packages in single codec size packages, the useEOF bit in the SICR register should be cleared (0), and the value of the SICR[SIM] field should be set for the appropriate data transfer size.

4.1.3 Bit clock polarity, clock phase and polarity

It is possible to select the SPI bit clock polarity, SPI clock polarity and SPI clock phase in which the data shift actually occurs. These selections are performed by ClkPol, CPOL and CPHA bits located in the serial interface control (SICR) register.

CPOL — this bit selects an inverted or non-inverted SPI clock. To transmit data between SPI modules, the SPI modules must have identical CPOL values.

CPHA — this bit is used to shift the SCKL serial clock. To transmit data between SPI modules, the SPI modules must have identical CPHA values.

A master/slave pair must use the same parameter values to communicate. For detailed information see Table 5.

Table 5. Possible configuration of bit clock polarity, clock phase and polarity

Parameter	Value	Description
ClkPol	0	Data in is sampled on the falling edge of the BCLK and data out is shifted on the rising edge
	1	Data in is sampled on the rising edge of the BCLK and data out is shifted on the falling edge
CPOL	0	Active-low clocks selected; SCKL idles high
	1	Active-high clocks selected; SCKL idles low
CPHA	0	Data transfer starts with assertion of SS
	1	Data transfer starts with the first edge of SCKL

4.1.4 Shift direction

The SPI support MSB and LSB shift direction. This direction can be set by the SHDIR bit in the SICR register. For more detail see Table 6.

Table 6. Possible configuration shift direction

Parameter	Value	Description
SHDIR	0	MSB first
	1	LSB first

CAUTION

The bit clock polarity PSC-SICR[ClkPol] has to be set to 0 when PSC behaves as an SPI device. Bit clock polarity is used for other protocols of the codec mode.

4.2 Master mode

In master mode (SICR[MSTR] = 1), the SPI provides the serial clock on the SCLK pin for the entire serial communication network. Data is output on the MOSI pin and latched from the MISO pin. The CCR register determines both transmit and receive bit transfer rates for the network.

Data written to the FIFOC[Tx DATA] register initiates data transmission on the MOSI pin, when the transmission is enabled. Simultaneously, received data is shifted through the MISO pin into the PSC[RB]. When the selected number of bits has been transmitted, the received data is transferred to the FIFOC where the user can use the FIFOC[Rx DATA] register for reading this data.

When the specified number of data bits has been shifted through FIFOC[Tx/Rx DATA], the following events occur:

1. FIFOC data register FIFOC[Data] transfers data to/from PSC's transfer/receiver buffer PSC[TB/RB].
2. FIFOC pointer register FIFOC[PTR] is changed.
3. FIFOC counter register FIFOC[Tx and Rx Count] is changed.

4.2.1 SPI transfer in active polling master mode

It is recommended to use interrupt mode or DMA mode to service SPI data requests, instead of the polling mode. The status byte returned on the Tx/Rx FIFO SR register can be polled at a given rate after strobing the transmitter or receiver, to see if there is free space for more bytes in the Tx slice or more bytes to read form the Rx slice. If the user wishes to use the polling mechanism to service an SPI data request, then the user is advised to use SR[Empty] or SR[ALARM] instead of SR[FULL]. When SR[FULL] is used, an overrun may occur.

NOTE

The PSC will not write to a full FIFO, so as to avoid data loss. Therefore a PSC access can't generate an ORERR situation. To see an ORERR event during a PSC transmission the SW must check the ORERR bit in the PSC ISR register.

The master SPI communication in the polling mode is summarized in Figure 11 and in the following paragraphs:

Emptying Tx FIFO and filling of Rx FIFO occurs simultaneously in master and in slave.

1. Clock initialization — see Section 3.2, “IPS bus” and Section 3.3, “MCLK frequency.”
2. PSC master initialization — see Section 3.5.1, “PSC initialization.”
3. FIFOC initialization — see Section 3.5.2, “FIFOC initialization.”
4. If EOF mode is used, then the EOF flag has to be set. This flag defines the next data written to the data register as the last in this frame.
5. The user application writes data to the Tx FIFO data register.
6. Test if the Rx FIFO slice is empty. If the Rx FIFO slice contains data then this data has to be read by the application.
7. Before sending the next set of data to the Tx FIFO, test if the Tx FIFO slice’s ALARM is reached. If the ALARM level is reached then you cannot send any data to the Tx FIFO, due to the possibility of overflow.
8. If you want send more data, then go to step 4; otherwise continue to step 9.
9. There is a possibility that the Tx FIFO contains some data which has to be sent out to the SPI shift register. You have to test if the Tx FIFO slice is empty or contains data waiting for transfer. If the Tx FIFO slice contains data then you have to wait. An Rx FIFO slice overflow may occur in this wait time, and you have to test if the Rx FIFO slice is empty in this wait time.
10. When step 7 goes to step 8, then the Tx FIFO slice does not contain any data. However, we have to count with the delay (a few processor ticks) between the Tx/Rx FIFO slices and the Tx/Rx shift register in the SPI module. There are a few possible ways to do this. The simplest way to cover it is to compare the amount of transmitted data and amount of received data. This is why polling mode is not recommended for SPI mode.

NOTE

In SPI master mode the PSC controls the serial data transfers. If the Tx FIFO becomes empty (underrun) or the Rx FIFO becomes full (overflow) in the middle of a multi-byte transfer, rather than set the Tx underrun or Rx overflow status bits, the PSC keeps the slave select signal low/active and stops the SCKL serial clock. When the Tx FIFO is no longer empty and the Rx FIFO no longer full, the transfer proceeds.

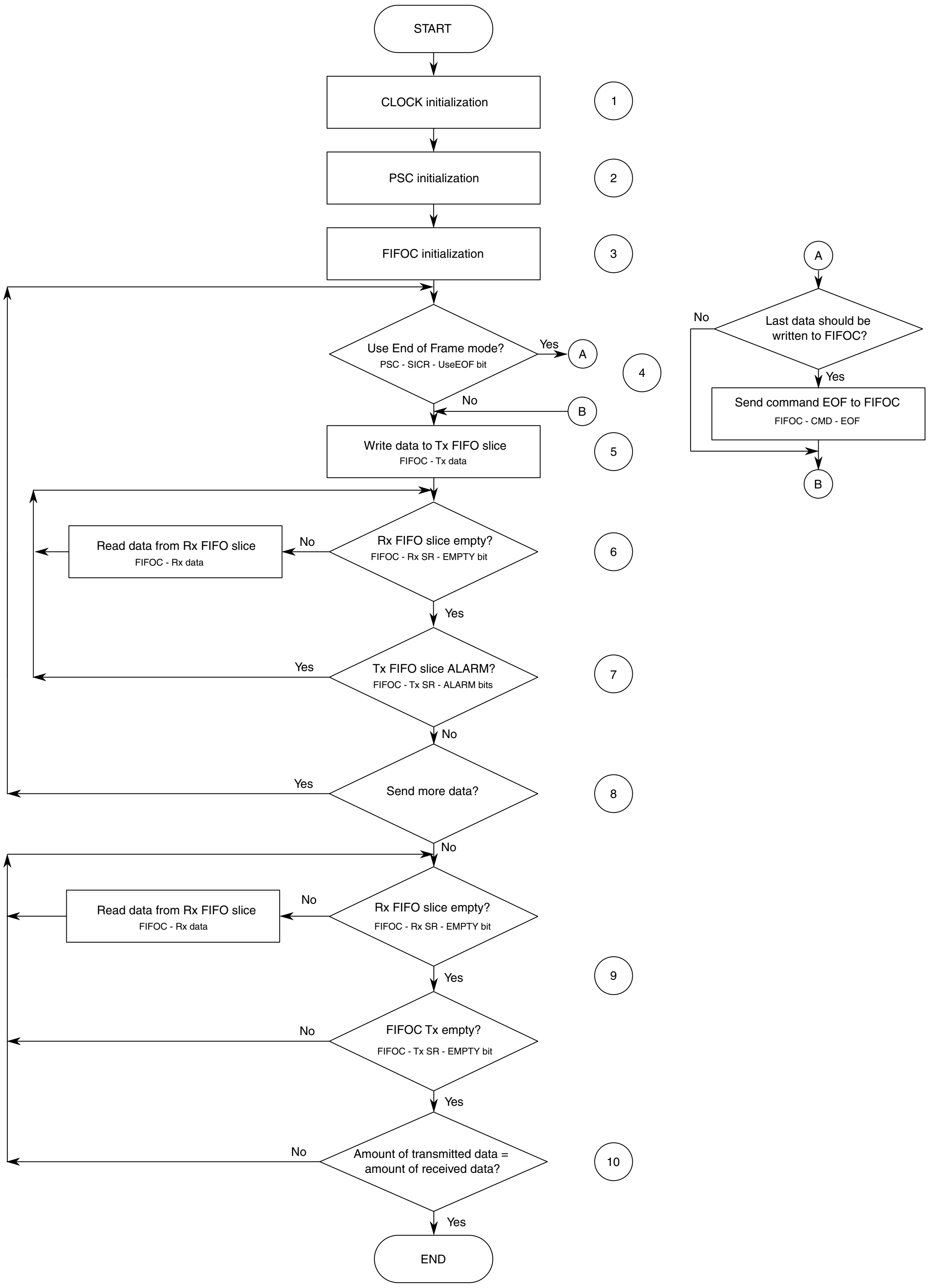

Figure 11. SPI master polling mode

4.3 Slave mode

In slave mode (SICR[MSTR] = 0), data shifts out on the SOMI pin and in on the SIMO pin.

The SCLK pin is used as the input for the serial shift clock, which is supplied from the external network by the master. The transfer rate is defined by this clock. Data written to the FIFOC[Tx DATA] register is transmitted to the network when appropriate edges of the SCLK signal are received from the network master.

To receive data, the SPI waits for the network master to send the SCLK signal and then shifts the data on the SIMO pin into the FIFOC[Rx DATA] register.

The slave select pin allows the transfer described above. An active-low signal on the SS pin allows the slave SPI to transfer data to the serial data line; an inactive-high signal causes the slave SPI serial shift register to stop and its serial output pin to be put into the high-impedance state. This allows many slave devices to be tied together on the network, although only one slave device is selected at a time.

When the specified number of data bits has been shifted through FIFOC[Tx/Rx DATA], the following events occur:

1. The FIFOC data register FIFOC[Data] transfers data to/from PSC's transfer/receiver buffer PSC[TB/RB].
2. FIFOC pointer register FIFOC[PTR].
3. FIFOC counter register FIFOC[Tx and Rx Count] is changed.

NOTE

In SPI slave mode, the MCLK must be running/enabled even though it is not used to generate the serial clock SCLK, which is provided by the external master SPI device. The frequency of MCLK is not critical, as long as it is faster than the SCLK frequency.

4.3.1 SPI transfer in active polling slave mode

The slave SPI communication in the polling mode is summarized in Figure 12 and in the following paragraphs:

Emptying Tx FIFO and filling of Rx FIFO occurs simultaneously as in master and in slave.

1. Clock initialization — see Section 3.2, "IPS bus," and Section 3.3, "MCLK frequency."
2. PSC slave initialization — see Section 3.5.1, "PSC initialization."
3. FIFOC initialization — see Section 3.5.2, "FIFOC initialization."
4. It is necessary to pre-fill the Tx FIFO slice to forestall underflow. The condition can minimally test the alarm level or whether the FIFO count register equals 1. (This is not recommended if using high SPI baud rate and with EOF mode used on the master side, due to delay between Tx/Rx FIFO slices and Tx/Rx shift register in the SPI module.)
5. Test if the Rx FIFO slice is empty. When the Rx FIFO slice contains data then this data has to be read by the core.
6. If all data has been sent to the Tx FIFO slice, then continue to step 7; otherwise go to step 4.

7. When the last data has been sent, then the slave tests if the Rx FIFO slice is empty. If the Rx FIFO slice still contains data then this data has to be read by the core.
8. Test if the Tx FIFO slice is empty.
9. When step 7 goes to step 8 then the Tx FIFO slice does not contain any data. However, we have to count with the delay (a few processor tics) between the Tx/Rx FIFO slices and the Tx/Rx shift register in the SPI module. There are a few possible ways to cover it. The simplest way to cover it is to compare the amount of transmitted data and the amount of received data. This is why polling mode is not recommended for SPI mode.

NOTE

To identify a URERR event during a PSC transmission, the SW must check the URERR bit in the PSC SR or ISR register.

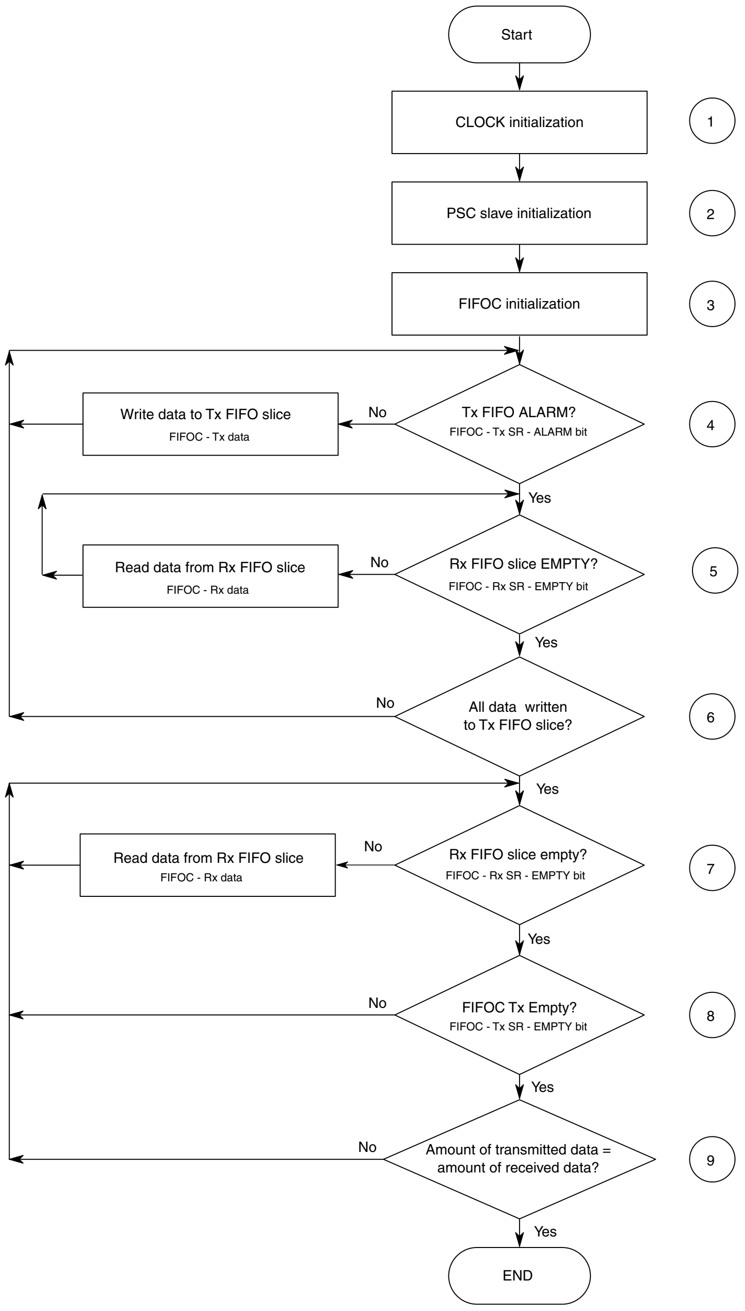
Start
CLOCK initialization
1
PSC slave initialization
2
FIFOC initialization
3
Tx FIFO ALARM?
FIFOC - Tx SR - ALARM bit
4
No
Write data to Tx FIFO slice
FIFOC - Tx data
Yes
Rx FIFO slice EMPTY?
FIFOC - Rx SR - EMPTY bit
5
No
Read data from Rx FIFO slice
FIFOC - Rx data
Yes
All data written to Tx FIFO slice?
6
No
Yes
Rx FIFO slice empty?
FIFOC - Rx SR - EMPTY bit
7
No
Read data from Rx FIFO slice
FIFOC - Rx data
Yes
FIFOC Tx Empty?
FIFOC - Tx SR - EMPTY bit
8
No
Yes
Amount of transmitted data = amount of received data?
9
No
Yes
END

Figure 12. SPI slave polling mode

4.4 Interrupt mode

The PSC and FIFOC modules generate interrupts which are addressed to the IPIC module. Interrupt vectors are provided on two levels — IPIC and peripheral levels. The IPIC module generally provides information about the peripheral which requested interrupt handling. Interrupts are enabled using the System Internal Interrupt Mask register (SIMSR) located in the IPIC where the user allows the IPIC to receive interrupt requests from the peripheral; and using the Interrupt Mask register (IMR) located in the

PSC and FIFOC where the user enables the type of interrupt event for the peripheral. When SPI transfer in the interrupt mode is used:

- FIFOC interrupts are used to service the SPI transfer.
- PSC interrupts are used to service an overrun/underrun error.

CAUTION

When an overrun or underrun situation occurs, the PSC does not write data to a full FIFO, so as to avoid data loss. Therefore the core will not receive information about an overrun or underrun situation from the FIFOC. Corresponding registers provided by the core are not re-read, and the user has to look into the interrupt status register provided by the PSC.

When an interrupt is received, the interrupt controller sets the corresponding bit in the System Internal Interrupt Pending registers (SIPNR). Each bit corresponds to an internal interrupt source. The user has two possibilities: check an internal interrupt source by SIPNR register and take care of priority management manually, or read the currently pending interrupt source with the highest priority level provided by the IPIC in the System Global Interrupt Vector register (SIVCR).

SIVCR provides a 7-bit code that represents the interrupt source to be handled/executed. Table 7 shows important codes and sources for the SPI interrupt mode. Possible interrupt events for the FIFO controller are shown in Table 8.

Table 7. SIVCR code used for SPI interrupt transfer

Source	Code
FIFOC	0x28
PSC0	0x44
PSC1	0x45
PSC2	0x46
PSC3	0x47
PSC4	0x20
PSC5	0x21
PSC6	0x22
PSC7	0x23
PSC8	0x24
PSC9	0x25
PSC10	0x26
PSC11	0x27

CAUTION

When a pending interrupt is managed, it is necessary to clear the corresponding interrupt pending bit in the Interrupt Status register PSC[ISR] by writing 1 to this register. When the interrupt handler is left and a pending interrupt is not cleared, the core immediately goes to the interrupt routine again.

Table 8. Possible interrupt events for FIFO controller

Interrupt event	Description
MEM error	Memory access error interrupt The access to the data register generates an access error interrupt.
Data ready	Data ready interrupt The FIFOC generates a data ready interrupt when the FIFO contains one or more data words.
ORERR	Overrun error interrupt The FIFOC generates an overrun error interrupt when an overrun error occurs (overrun = write access to an already full FIFO). See caution above.
URERR	Underrun error interrupt The FIFOC generates an underrun error interrupt when underrun error occurs (underrun = read access from an already empty FIFO). See caution above.
Alarm	Alarm interrupt The FIFOC generates an alarm interrupt when the amount of data in the FIFO reaches the alarm level.
Full	Full interrupt the FIFOC generates a full interrupt when the FIFO is full.
Empty	Empty interrupt The FIFOC generates an empty interrupt when the FIFO is empty.

An example of the interrupt SPI communication is shown in Figure 13, Figure 14, and Figure 15, and is summarized in the following paragraphs.

SPI interrupt mode receiving — Rx alarm level interrupt is used for managing received data. When alarm level is set to one then the alarm level interrupt is similar to the data ready interrupt. Increasing the alarm level decreases the number of Rx interrupt events. The amount of data which the user wishes to receive has to be dividable by the alarm level to ensure that all data is received correctly.

SPI interrupt mode transferring — Tx empty interrupt is used for managing the transfer of data. The command to enable empty interrupt source is used to start the data transfer. When an empty interrupt occurs, a data packet equal to the alarm level can be sent to the Tx FIFO slice. When all data is sent to the Tx FIFO slice and an empty interrupt occurs, then the user has to disable empty interrupt source to avoid an infinite loop situation.

SPI interrupt transfer — main loop:

1. Clock initialization —— see Section 3.1, “PIN muxing,” and Section 3.2, “IPS bus.”
2. PSC initialization — see Section 3.5.1, “PSC initialization.”
3. FIFOC initialization — see Section 3.5.2, “FIFOC initialization.”
4. SPI interrupt initialization — see Section 3.3, “MCLK frequency.”

5. SPI transfer starts by enabling a Tx empty interrupt.
6. Enable Tx FIFO empty interrupt source.
7. Wait till SPI transfer is complete.

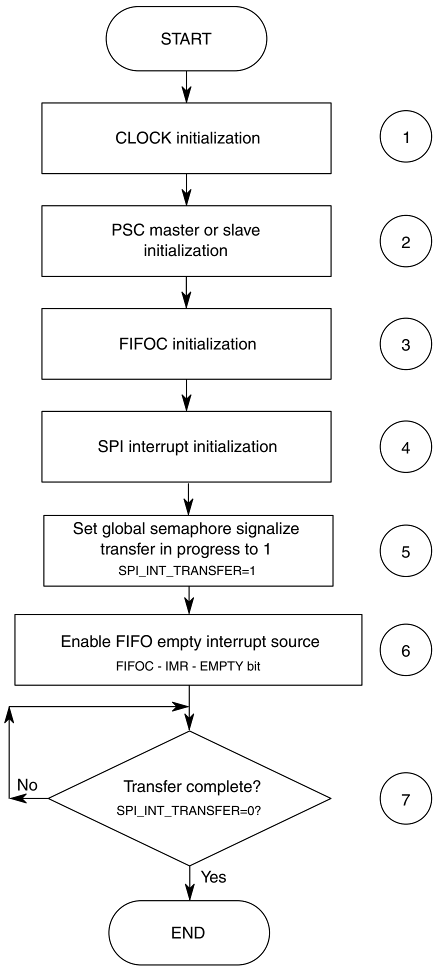
START
CLOCK initialization
1
PSC master or slave initialization
2
FIFOC initialization
3
SPI interrupt initialization
4
Set global semaphore signalize transfer in progress to 1
SPI_INT_TRANSFER=1
5
Enable FIFO empty interrupt source
FIFOC - IMR - EMPTY bit
6
No
Transfer complete?
SPI_INT_TRANSFER=0?
7
Yes
END

Figure 13. SPI interrupt mode example — main loop

SPI interrupt handler:

1. Check global interrupt vector number for FIFOC interrupt number. The global interrupt vector specifies a 7-bit unique number that represents the interrupt source to be handled/executed. When an interrupt request occurs, the System Global Interrupt Vector register (SIVCR) can be read and latches the highest priority interrupt. Interrupt vector numbers are assigned to each module and cannot be changed. FIFOC always returns 0x28 as its interrupt vector regardless of its relative priority in the SYSC group.
2. Detect the number of PSC and type of the slice with currently pending interrupt. One bit per PSC FIFO shows all PSCs with currently pending interrupts separately for Tx and Rx slice.
3. Get the reason for the interrupt for the Rx FIFO slice — the Interrupt Status register (ISR) shows the reason for the interrupt event. For descriptions of the possible interrupt events see Table 8.

4. If the interrupt reason from step 3 is an alarm interrupt, read data from the Rx FIFO slice by the core until the Rx FIFO slice is empty.
5. When a pending interrupt is managed, clear the corresponding bit in the interrupt status register PSC[ISR] by writing a one to this register.
6. Get the reason for the interrupt for the Tx FIFO slice — the Interrupt Status register (ISR) shows the reason for the interrupt event. For descriptions of the possible interrupt events see Table 8.
7. If the interrupt reason from step 6 is an empty interrupt, test to see if some data needs to be sent. If data is waiting for transfer to the FIFO slice, then go to step 11. Otherwise, go to step 8.
8. Set the global semaphore to signal “transfer complete.”
9. When a pending interrupt is managed, clear the corresponding bit in the interrupt status register PSC[ISR] by writing one to this register.
10. Disable the Tx FIFO empty interrupt source.
11. Calculate the remaining data for transfer. A data packet for which the size equals the alarm level is always transferred in the interrupt routine. This step calculates which data packet has to be sent next.
12. Steps 12–14 handle the data packet transfer. If the amount of sent data equals the alarm level, go to step 15. Otherwise, go to step 13.
13. If end of frame mode is used, send command EOF to the FIFOC before the last data is written to the FIFO slice.
14. Write data to the Tx FIFO slice.
15. When a pending interrupt is managed, clear the corresponding bit in the Interrupt Status register PSC[ISR] by writing one to this register.

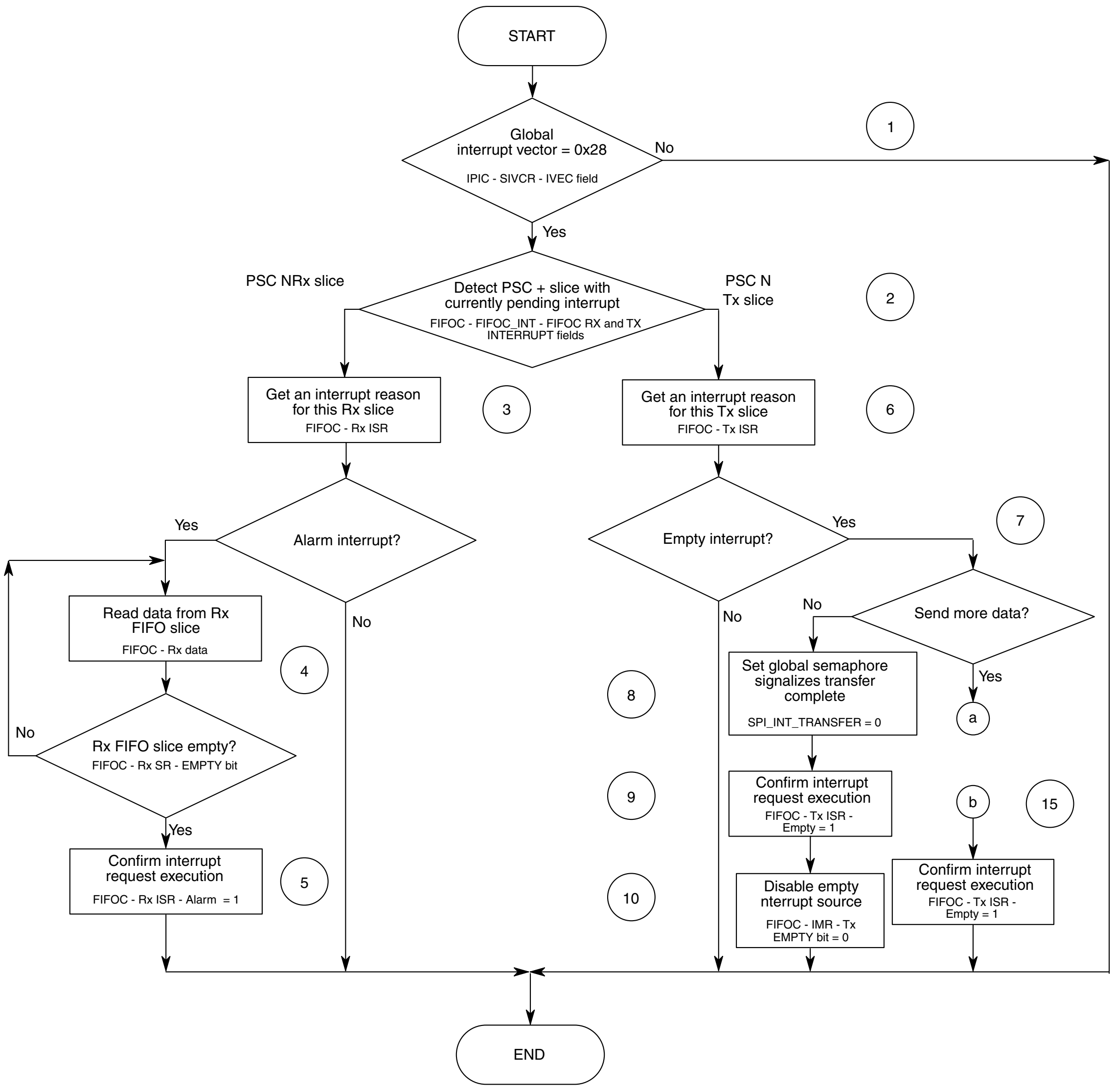
START
Global interrupt vector = 0x28
IPIC - SIVCR - IVEC field
No
Yes
1
Detect PSC + slice with currently pending interrupt
FIFOC - FIFOC_INT - FIFOC RX and TX INTERRUPT fields
PSC NRx slice
PSC N Tx slice
2
Get an interrupt reason for this Rx slice
FIFOC - Rx ISR
3
Get an interrupt reason for this Tx slice
FIFOC - Tx ISR
6
Alarm interrupt?
Yes
No
Empty interrupt?
Yes
No
7
Send more data?
No
Yes
a
Read data from Rx FIFO slice
FIFOC - Rx data
4
Rx FIFO slice empty?
FIFOC - Rx SR - EMPTY bit
No
Yes
Confirm interrupt request execution
FIFOC - Rx ISR - Alarm = 1
5
8
9
10
Set global semaphore signalizes transfer complete
SPI_INT_TRANSFER = 0
Confirm interrupt request execution
FIFOC - Tx ISR - Empty = 1
Disable empty nterrupt source
FIFOC - IMR - Tx EMPTY bit = 0
b
15
Confirm interrupt request execution
FIFOC - Tx ISR - Empty = 1
END

Figure 14. SPI interrupt mode example – interrupt handler

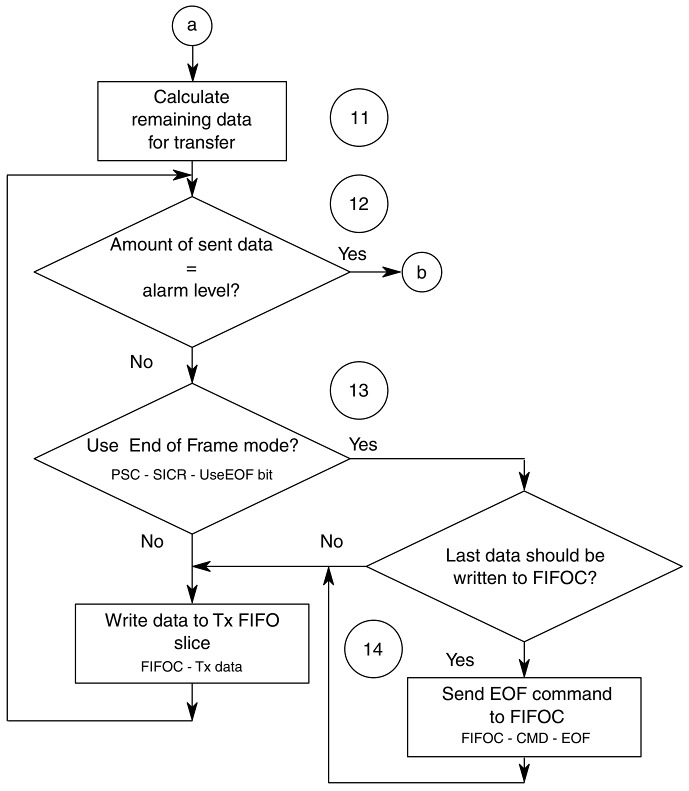
a
Calculate remaining data for transfer
11
12
Amount of sent data = alarm level?
Yes
b
No
13
Use End of Frame mode?
PSC - SICR - UseEOF bit
Yes
No
No
Last data should be written to FIFOC?
Write data to Tx FIFO slice
FIFOC - Tx data
14
Yes
Send EOF command to FIFOC
FIFOC - CMD - EOF

Figure 15. SPI interrupt mode example – interrupt handler (continued)

4.5 DMA mode

The FIFOC supports a DMA to transmit and receive data. It is recommended to use the DMA for data transfers, to decrease the number of interrupts and core loading.

When the DMA engine is used, the DMA transfers a packet of data located in the memory (variable or array) to the FIFOC or back. The FIFOC module is automatically controlled by the DMA engine. For proper functionality of the DMA transfer the SPI DMA initialization is required — see Section 3.7, “Direct memory access.”

A DMA transfer begins by starting a DMA Rx or Tx task. The DMA transfers/receives data until the alarm level is reached. The Tx FIFO slice is filled to the alarm level, and the DMA task has finished its first loop. The next loop starts when the Tx FIFO slice reports an empty status. When all data has been sent, the DMA sends a transfer complete interrupt request to the IPIC module.

The Rx FIFO slice is filled to the alarm level by the PSC. When the alarm level is reached, the DMA starts to transfer data from the Rx FIFO slice to the memory. When the Rx FIFO slice is empty, the DMA task has finished the first loop. The Next loop starts when the Rx FIFO slice reaches the alarm level. When all data has been received, the DMA sends a transfer complete interrupt request to the IPIC module.

NOTE

The amount of transferred or received data is set in the TCD structure. The DMA module does not send a transfer complete interrupt request to the IPIC module unless the amount of data set in the TCD has not been moved.

The DMA module generates interrupts which are addressed to the IPIC module. Interrupt vectors are provided on two levels: IPIC and peripheral.

Interrupts are enabled using the System Internal Interrupt Mask register (SIMSR) located in the IPIC and command register (CMD) located in the FIFOC. All interrupts from the FIFOC and PSC are disabled.

When an interrupt is received, the interrupt controller sets the corresponding System Internal Interrupt Pending registers (SIPNR). Each bit corresponds to an internal interrupt source. The user has two possibilities: check an internal interrupt source by SIPNR register and take care of priority management in that way, or read currently pending interrupt source with the highest priority level provided by the IPIC in the System Global Interrupt Vector register (SIVCR). The SIVCR provides 7-bit code representing the unmasked interrupt source of the highest priority level. When a DMA interrupt has to be managed, the SIVCR provides code 0x41.

CAUTION

The DMA engine has limited support for a transfer which uses the end of frame mode. The DMA engine does not send an EOF command to the FIFOC command register. The EOF command defines the next data written to the data register as the last data of this frame. When the user wishes to use end of frame mode, the TCD structure for the Tx channel needs to be changed. The current major iteration count needs to be set one loop less and the last source address adjustment needs to be re-counted as well. The DMA Tx transfer (minor × major interaction counts) is decreased for the last loop. Data from the last loop plus the EOF flag needs to be transferred using polling or interrupt mode. The DMA task for the Rx channel remains without any change.

An example of the DMA SPI communication is show in Figure 16 and Figure 17, and is summarized in the following paragraphs.

SPI DMA transfer — main loop:

1. Clock initialization — see Section 3.1, “PIN muxing,” and Section 3.2, “IPS bus.”
2. PSC initialization — see Section 3.5.1, “PSC initialization.”
3. FIFOC initialization — see Section 3.5.2, “FIFOC initialization.”
4. SPI DMA initialization — see Section 3.4, “Data polling, interrupts, and DMA.”
5. When proper SPI DMA initialization is done and all data is ready to be sent, the user can start the SPI DMA transfer by writing the specified Rx channel number to the DMA channel register, then set the Enable Request (DMASERQ) register. The DMA starts the transfer, which is controlled by the FIFOC module, and transfers data from the Rx FIFO slice to the defined memory automatically, without core loading. Dma_int_cntRx variable is set and shows the Rx transfer in progress.

6. When the Rx DMA transfer is in progress, the Tx DMA transfer can be started by writing the specified Tx channel number to the DMASERQ register. The DMA starts the transfer which is controlled by the FIFOC module, and transfers data from the defined memory to the Tx FIFO slice. The dma_int_cntTx variable is set and shows the Tx transfer in progress.
7. In the main loop, the user is informed about the finished Tx transfer by global variable dma_int_cntTx = 0.
8. When SPI master mode and EOF mode is used, the EOF flag has to be set. The program continues on to step 10.
9. In the main loop, the user is informed about the finished Rx transfer by global variable dma_int_cntRx = 0.
10. When EOF mode is used, the current major iteration count is set to one loop less. One loop needs to be sent in interrupt or polling mode.

    Let's use polling mode for the transfer. The first step is to calculate the position in the transferred array, where the DMA finished the transfer. The size of the one loop equals the Tx alarm level and the specified position equals the size of the transferred array — the Tx alarm level. This data will be sent using polling mode.
11. Test if the Tx FIFO slice reports an alarm event.
12. When the last data should be sent to the FIFOC, send an EOF command to the FIFOC command register.
13. Write data to the Tx FIFO slice.
14. When another data packet needs to be sent, go to step 11. Otherwise, go to step 9.

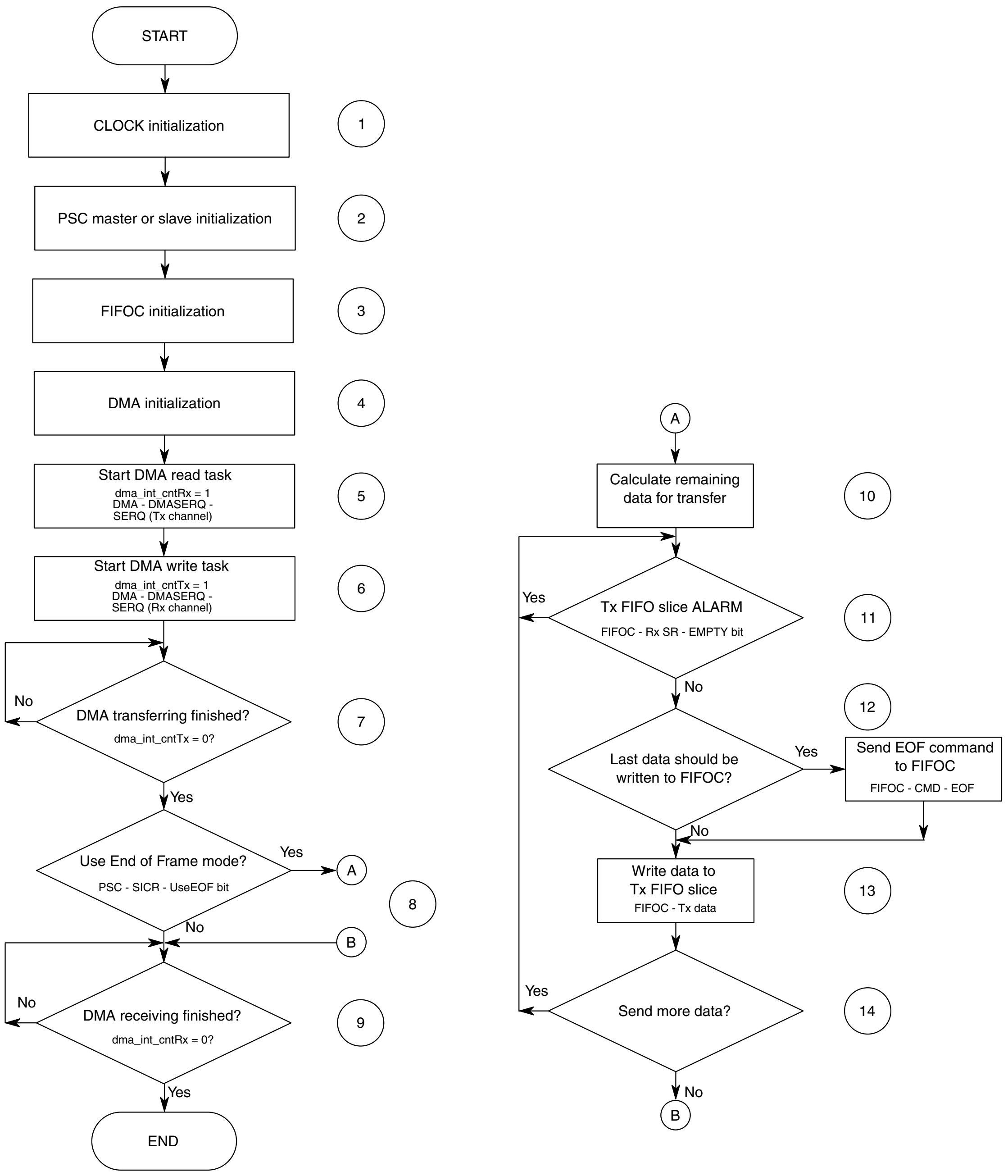

Figure 16. SPI DMA mode example – main loop

SPI DMA transfer — interrupt handler:

1. Check the global interrupt vector number for the DMA interrupt number. The global interrupt vector specifies a 7-bit unique number of the IPIC's highest priority interrupt source pending to the core. When an interrupt request occurs, the System Global Interrupt Vector register (SIVCR) can be read and latches the highest priority interrupt. Interrupt vector numbers are assigned to each module and cannot be changed. DMA always returns 0x41 as its interrupt vector regardless of its relative priority in the SYSC group.

2. Detect the DMA channel with a currently pending interrupt. One bit per DMAINTH or DMAINTL register shows all DMA channels with currently pending interrupts.
3. If an Rx channel interrupt is handled, clear the dma_int_cntRx flag which follows the status of the DMA transfer.
4. When a pending interrupt is managed, clear the DMA pending interrupt in the DMA Clear Interrupt Request register DMACINT[CINT] by writing the channel number to this register.
5. If Tx channel interrupt is handled, clear the dma_int_cntTx flag which follows the status of the DMA transfer.
6. When a pending interrupt is managed, clear the DMA pending interrupt in the DMA Clear Interrupt Request register DMACINT[CINT] by writing the channel number to this register.

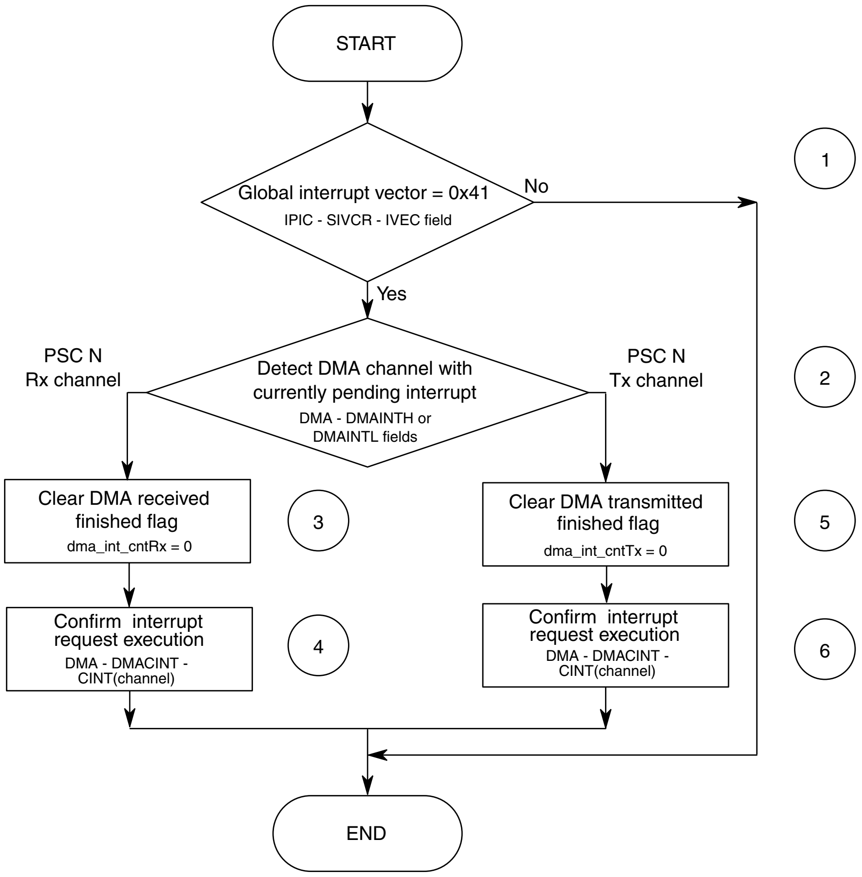

Figure 17. SPI DMA mode example – interrupt handler

5 References

1. MPC5121ERM, *MPC5121e Microcontroller Reference Manual*
2. MPC5121E, *MPC5121e/MPC5123 Data Sheet*
3. *HCS08 Unleashed: Designer's Guide to the HCS08 Microcontrollers*. Fábio Pereira. 2008. ISBN: 1-4196-8592-9
4. http://embedded.com

THIS PAGE IS INTENTIONALLY BLANK

Document Number: AN3904
Rev. 0
08/2009