APPLICATION NOTE

AT07890: SAM4 Serial Peripheral Interface (SPI)

ASF PROGRAMMERS MANUAL

SAM4 Serial Peripheral Interface (SPI)

This driver for SAM4 devices provides an interface for the configuration and management of the device's Serial Peripheral Interface functionality.

The Serial Peripheral Interface is a synchronous serial data link that provides communication with external devices in master or slave mode. It also enables communication between processors if an external processor is connected to the system.

The following peripherals are used by this module:

- SPI (Serial Peripheral Interface)

The outline of this documentation is as follows:

Table of Contents

Software License

1. Prerequisites

There are no prerequisites for this module.

2. Module Overview

The Serial Peripheral Interface (SPI) is a synchronous serial data link that provides communication with external devices in master or slave mode.

The SPI is essentially a shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI device acts as the master which controls the data flow, while the other devices act as slaves. Data is clocked into and out of the slave(s) by the master.

2.1 Bus Topology

SPI systems can be configured in two topologies:

- Single Master Protocol: Contains a single CPU that is always the master while other devices in the system are always slaves.
- Multiple Master Protocol: Contains different CPUs and they take turns being the master.

Note One master may simultaneously shift data into multiple slaves. However, only one slave may drive its output to write data back to the master at any given time.

2.2 Master to Slave Device Interface

A slave device is selected when the master asserts its NSS signal. If multiple slave devices exist, the master generates a separate slave select signal for each slave (NPCS). The SPI system consists of two data lines and two control lines:

- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input(s) of the slave(s).
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master. There may be no more than one slave transmitting data during any particular transfer.
- Serial Clock (SPCK): This control line is driven by the master and regulates the flow of the data bits. The master may transmit data at a variety of baud rates; the SPCK line cycles once for each bit that is transmitted.
- Slave Select (NSS): As long as this control line is high, nothing will be shifted out/in from the slave's shift register when the SPI module is configured in slave mode.

2.3 Data Transfer

Four combinations of polarity (CPOL) and phase (NCPHA) are available for data transfers. The clock polarity and phase can be controlled by user applications. These two parameters determine the edges of the clock signal on which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another:

SPI Mode	CPOL	NCPHA	Shift SPCK Edge	Capture SPCK Edge	SPCK Inactive Level
0	0	1	Falling	Rising	Low
1	0	0	Rising	Falling	Low
2	1	1	Rising	Falling	High
3	1	0	Falling	Rising	High

Note A pair of master/slave devices must use the same clock polarity and phase values to communicate. If multiple slaves are used and fixed in different configurations, the master must reconfigure itself each time it needs to communicate with a slave of a different type.

2.4 Application

The SPI module can be used to communicate with external memories such as DataFlash® and 3-wire EEPROMs and a wide variety of external peripherals e.g. ADCs, DACs, LCD Controllers, CAN Controllers, sensors, and co-processors.

3. Special Considerations

3.1 I/O Lines

The pins used for interfacing to external devices may be multiplexed with GPIO lines. The user application must first program the GPIO controller to assign these pins to the SPI module.

3.2 Power Management

The SPI module may be clocked through the Power Management Controller (PMC), thus the user application must first configure the PMC to enable the SPI clock.

3.3 Interrupt

The SPI module has an interrupt line connected to the Nested Vectored Interrupt Controller (NVIC). Handling the SPI interrupt requires that the NVIC is configured before configuring the SPI.

3.4 Peripheral DMA Controller (PDC/PDCA)

The SPI module can be used in conjunction with the PDC in order to reduce processor overhead. For a full description of the PDC/PDCA, refer to the corresponding section in the device-specific datasheet.

4. Extra Information

For extra information, see Extra Information for Serial Peripheral Interface Driver. This includes:

- Acronyms
- Dependencies
- Errata
- Module History

5. Examples

For a list of examples related to this driver, see Examples for SPI Driver.

6. API Overview

6.1 Variable and Type Definitions

6.1.1 Type spi_cs_behavior_t

```
typedef enum spi_cs_behavior spi_cs_behavior_t
```

SPI Peripheral Chip Select (NPCSx) behavior modes while transferring data in Master mode.

6.2 Macro Definitions

6.2.1 Macro spi_get_pcs

```
#define spi_get_pcs(chip_sel_id) \
```

Note When chip select n is asserted, NPCSn is set to a low level.

Table 6-1. Parameters

Data direction	Parameter name	Description
[in]	chip_sel_id	The chip select number used

6.3 Function Definitions

6.3.1 Function spi_calc_baudrate_div()

Calculate the baudrate divider.

```
int16_t spi_calc_baudrate_div(
  const uint32_t baudrate,
  uint32_t mck)
```

Table 6-2. Parameters

Data direction	Parameter name	Description
[in]	baudrate	Baudrate value
[in]	mck	SPI module input clock frequency (MCK clock in Hz)

Returns The baudrate divider or an error code.

Table 6-3. Return Values

Return value	Description
>=1	Valid baudrate divisor in the range 1 to 255
-1	The desired baudrate cannot be achieved with the supplied parameters

6.3.2 Function spi_configure_cs_behavior()

Configure the Peripheral Chip Select (NPCSx) behavior for SPI data transfer.

```
void spi_configure_cs_behavior(
  Spi * p_spi,
  uint32_t ul_pcs_ch,
  uint32_t ul_cs_behavior)
```

Table 6-4. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer
[in]	ul_pcs_ch	Peripheral Chip Select channel (range 0 to 3)
[in]	ul_cs_behavior	Behavior of the Chip Select after transfer

6.3.3 Function spi_disable()

Disable the SPI module.

```
void spi_disable(
  Spi * p_spi)
```

Note Peripheral Chip Select (NPCSx) is de-asserted, which indicates that the last data is done, and the user application should check TX_EMPTY before disabling SPI.

Table 6-5. Parameters

Data direction	Parameter name	Description
[out]	p_spi	Module hardware register base address pointer

6.3.4 Function spi_disable_clock()

Disable the SPI module clock.

```
void spi_disable_clock(
  Spi * p_spi)
```

Table 6-6. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

6.3.5 Function spi_disable_interrupt()

Disable SPI interrupts.

```
void spi_disable_interrupt(
  Spi * p_spi,
  uint32_t ul_sources)
```

Table 6-7. Parameters

Data direction	Parameter name	Description
[out]	p_spi	Module hardware register base address pointer
[in]	ul_sources	A bitmask of interrupts to be disabled

Where input parameter *ul_sources* is a bitmask containing one or more of the following:

Parameter Value	Description
SPI_IDR_RDRF	Receive Data Register Full interrupt disable
SPI_IDR_TDRE	Transmit Data Register Empty interrupt disable
SPI_IDR_MODF	Mode Fault Error interrupt disable
SPI_IDR_OVRES	Overrun Error interrupt disable
SPI_IDR_ENDRX	End of Receive Buffer interrupt disable
SPI_IDR_ENDTX	End of Transmit Buffer interrupt disable
SPI_IDR_RXBUFF	Receive Buffer Full interrupt disable
SPI_IDR_TXBUFE	Transmit Buffer Empty interrupt disable
SPI_IDR_NSSR	NSS Rising interrupt disable
SPI_IDR_TXEMPTY	Transmission Registers Empty interrupt disable
SPI_IDR_UNDES	Underrun Error interrupt disable

6.3.6 Function spi_disable_loopback()

Disable SPI internal loopback mode.

```
void spi_disable_loopback(
  Spi * p_spi)
```

Table 6-8. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

6.3.7 Function spi_disable_mode_fault_detect()

Disable Mode Fault Detection.

```
void spi_disable_mode_fault_detect(
  Spi * p_spi)
```

Table 6-9. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

6.3.8 Function spi_disable_peripheral_select_decode()

Disable Peripheral Select Decode.

```
void spi_disable_peripheral_select_decode(
  Spi * p_spi)
```

Table 6-10. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

6.3.9 Function spi_disable_tx_on_rx_empty()

Disable the SPI module starting data transfers only when the Receive Data Register is empty.

```
void spi_disable_tx_on_rx_empty(
  Spi * p_spi)
```

Table 6-11. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

6.3.10 Function spi_enable()

Enable the SPI module.

```
void spi_enable(
  Spi * p_spi)
```

Table 6-12. Parameters

Data direction	Parameter name	Description
[out]	p_spi	Module hardware register base address pointer

6.3.11 Function spi_enable_clock()

Enable the SPI module clock.

```
void spi_enable_clock(
  Spi * p_spi)
```

Table 6-13. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

6.3.12 Function spi_enable_interrupt()

Enable SPI interrupts.

```
void spi_enable_interrupt(
  Spi * p_spi,
  uint32_t ul_sources)
```

Table 6-14. Parameters

Data direction	Parameter name	Description
[out]	p_spi	Module hardware register base address pointer
[in]	ul_sources	A bitmask of interrupts to be enabled

Where input parameter *ul_sources* is a bitmask containing one or more of the following:

Parameter Value	Description
SPI_IER_RDRF	Receive Data Register Full interrupt enable
SPI_IER_TDRE	Transmit Data Register Empty interrupt enable
SPI_IER_MODF	Mode Fault Error interrupt enable
SPI_IER_OVRES	Overrun Error interrupt enable
SPI_IER_ENDRX	End of Receive Buffer interrupt enable
SPI_IER_ENDTX	End of Transmit Buffer interrupt enable
SPI_IER_RXBUFF	Receive Buffer Full interrupt enable
SPI_IER_TXBUFE	Transmit Buffer Empty interrupt enable
SPI_IER_NSSR	NSS Rising interrupt enable
SPI_IER_TXEMPTY	Transmission Registers Empty interrupt enable
SPI_IER_UNDES	Underrun Error interrupt enable

6.3.13 Function spi_enable_loopback()

Enable SPI internal loopback mode.

```
void spi_enable_loopback(
  Spi * p_spi)
```

Table 6-15. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

6.3.14 Function spi_enable_mode_fault_detect()

Enable Mode Fault Detection.

```
void spi_enable_mode_fault_detect(
  Spi * p_spi)
```

Table 6-16. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

6.3.15 Function spi_enable_peripheral_select_decode()

Enable Peripheral Select Decode.

```
void spi_enable_peripheral_select_decode(
  Spi * p_spi)
```

Table 6-17. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

6.3.16 Function spi_enable_tx_on_rx_empty()

Enable the SPI module to start data transfers only when the Receive Data Register is empty.

```
void spi_enable_tx_on_rx_empty(
  Spi * p_spi)
```

Table 6-18. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

6.3.17 Function spi_get()

Read SPI data.

```
uint16_t spi_get(
  Spi * p_spi)
```

Table 6-19. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns The data value.

6.3.18 Function spi_get_mode()

Get SPI Master/Slave operating mode.

```
uint32_t spi_get_mode(
  Spi * p_spi)
```

Table 6-20. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns The SPI module operating mode.

Table 6-21. Return Values

Return value	Description
0	Operating in Slave Mode
1	Operating in Master Mode

6.3.19 Function spi_get_mode_fault_detect_setting()

Check if mode fault detection is enabled.

```
uint32_t spi_get_mode_fault_detect_setting(
  Spi * p_spi)
```

Table 6-22. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns The mode fault detection status.

Table 6-23. Return Values

Return value	Description
0	Mode fault detection enabled
1	Mode fault detection disabled

6.3.20 Function spi_get_pdc_base()

Get PDC registers base address.

```
Pdc * spi_get_pdc_base(
  Spi * p_spi)
```

Note This function is not available on SAM4L devices.

Table 6-24. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns PDC registers base for PDC driver to access.

6.3.21 Function spi_get_peripheral_select_decode_setting()

Get Peripheral Select Decode mode.

```
uint32_t spi_get_peripheral_select_decode_setting(
  Spi * p_spi)
```

Table 6-25. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns The Peripheral Select Decode mode.

Table 6-26. Return Values

Return value	Description
0	Direct mode is enabled
1	Decode mode is enabled

6.3.22 Function spi_get_peripheral_select_mode()

Get Peripheral Select mode.

```
uint32_t spi_get_peripheral_select_mode(
  Spi * p_spi)
```

Table 6-27. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns The Peripheral Select mode.

Table 6-28. Return Values

Return value	Description
0	Configured in Fixed Peripheral Select mode
1	Configured in Variable Peripheral Select mode

6.3.23 Function spi_get_rx_access()

Get receive data register address for DMA operation.

```
void * spi_get_rx_access(
  Spi * p_spi)
```

Note This function is only available on SAM3U and SAM3XA devices.

Table 6-29. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns Receive address for DMA access.

6.3.24 Function spi_get_tx_access()

Get transmit data register address for DMA operation.

```
void * spi_get_tx_access(
  Spi * p_spi)
```

Note This function is only available on SAM3U and SAM3XA devices.

Table 6-30. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns Transmit address for DMA access.

6.3.25 Function spi_get_tx_on_rx_empty_setting()

Check if SPI nodule is performing data transfers that start only when the Receive Data Register is empty.

```
uint32_t spi_get_tx_on_rx_empty_setting(
  Spi * p_spi)
```

Table 6-31. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns

The transmit on receive data empty behavior.

Table 6-32. Return Values

Return value	Description
0	Transmit does not wait for receive data to be empty before starting transmission.
1	Transmit waits for receive data to be empty

6.3.26 Function spi_get_writeprotect_status()

Indicate write protect status.

```
uint32_t spi_get_writeprotect_status(
  Spi * p_spi)
```

Table 6-33. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns

SPI_WPSR value.

6.3.27 Function spi_is_enabled()

Check if the SPI module is enabled.

```
uint32_t spi_is_enabled(
  Spi * p_spi)
```

Table 6-34. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns The SPI module status.

Table 6-35. Return Values

Return value	Description
0	SPI module is disabled
1	SPI module is enabled

6.3.28 Function spi_is_rx_full()

Check if the SPI module contains received data.

```
uint32_t spi_is_rx_full(
  Spi * p_spi)
```

Table 6-36. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns The Receive Data Holding Register status.

Table 6-37. Return Values

Return value	Description
0	No data has been received
1	Data has been received

6.3.29 Function spi_is_rx_ready()

Check if all the SPI data receivers are ready.

```
uint32_t spi_is_rx_ready(
  Spi * p_spi)
```

Table 6-38. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns The SPI data receiver status.

Table 6-39. Return Values

Return value	Description
0	Data receivers are not ready
1	Data receivers are ready

6.3.30 Function spi_is_tx_empty()

Check if all the SPI data transmissions are complete.

```
uint32_t spi_is_tx_empty(
  Spi * p_spi)
```

Table 6-40. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns The SPI data transmission complete status.

Table 6-41. Return Values

Return value	Description
0	All data transmissions are not complete
1	All data transmissions are complete

6.3.31 Function spi_is_tx_ready()

Check if the SPI data transmitter is ready.

```
uint32_t spi_is_tx_ready(
  Spi * p_spi)
```

Table 6-42. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns The SPI data transmitter ready status.

Table 6-43. Return Values

Return value	Description
0	Data transmitter is not ready
1	Data transmitter is ready

6.3.32 Function spi_put()

Write SPI data.

```
void spi_put(
  Spi * p_spi,
  uint16_t data)
```

Table 6-44. Parameters

Data direction	Parameter name	Description
[out]	p_spi	Module hardware register base address pointer
[in]	data	The data value to be transmitted

6.3.33 Function spi_read()

Read SPI data and it's peripheral chip select value.

```
spi_status_t spi_read(
  Spi * p_spi,
  uint16_t * us_data,
  uint8_t * p_pcs)
```

Table 6-45. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer
[out]	us_data	Pointer to the location where to store the received data word
[out]	p_pcs	Pointer to fill Peripheral Chip Select value

Note The Peripheral Chip Select value is only valid when the SPI module is configured to operate in Master Mode.

Returns SPI operation error code.

Table 6-46. Return Values

Return value	Description
SPI_OK	SPI Receive Data Register read successfully
SPI_ERROR_TIMEOUT	A Software time-out occurred while waiting for incoming data

6.3.34 Function spi_read_interrupt_mask()

Read SPI interrupt mask.

```
uint32_t spi_read_interrupt_mask(
  Spi * p_spi)
```

Table 6-47. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns The interrupt mask value.

6.3.35 Function spi_read_status()

Read SPI status register.

```
uint32_t spi_read_status(
  Spi * p_spi)
```

Table 6-48. Parameters

Data direction	Parameter name	Description
[in]	p_spi	Module hardware register base address pointer

Returns SPI status register value.

6.3.36 Function spi_reset()

Perform a software-triggered hardware reset of the SPI interface and set it into Slave Mode.

```
void spi_reset(
  Spi * p_spi)
```

Table 6-49. Parameters

Data direction	Parameter name	Description
[out]	p_spi	Module hardware register base address pointer

Note PDC channels are not affected by a software reset.

6.3.37 Function spi_set_baudrate_div()

Set the Serial Clock Baud Rate divider value (SCBR).

```
void spi_set_baudrate_div(
  Spi * p_spi,
  uint32_t ul_pcs_ch,
  uint8_t uc_baudrate_divider)
```

Table 6-50. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

Data direction	Parameter name	Description
[in]	ul_pcs_ch	Peripheral Chip Select channel (range 0 to 3)
[in]	uc_baudrate_divider	Baudrate divider from MCK

6.3.38 Function spi_set_bits_per_transfer()

Set the number of bits per SPI data transfer.

```
void spi_set_bits_per_transfer(
  Spi * p_spi,
  uint32_t ul_pcs_ch,
  uint32_t ul_bits)
```

Table 6-51. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Pointer to an SPI instance
[in]	ul_pcs_ch	Peripheral Chip Select channel (range 0 to 3)
[in]	ul_bits	Number of bits (range 8 to 16)

Where the input parameter *ul_bits* can be one of the following :

Parameter Value	Description
SPI_CSR_BITS_8_BIT	8bit data transfers
SPI_CSR_BITS_9_BIT	9bit data transfers
SPI_CSR_BITS_10_BIT	10bit data transfers
SPI_CSR_BITS_11_BIT	11bit data transfers
SPI_CSR_BITS_12_BIT	12bit data transfers
SPI_CSR_BITS_13_BIT	13bit data transfers
SPI_CSR_BITS_14_BIT	14bit data transfers
SPI_CSR_BITS_15_BIT	15bit data transfers
SPI_CSR_BITS_16_BIT	16bit data transfers

6.3.39 Function spi_set_clock_phase()

Set the SPI data capture/change phase.

```
void spi_set_clock_phase(
  Spi * p_spi,
  uint32_t ul_pcs_ch,
  uint32_t ul_phase)
```

Table 6-52. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

Data direction	Parameter name	Description
[in]	ul_pcs_ch	Peripheral Chip Select channel (range 0 to 3)
[in]	ul_phase	Data change/capture on SPCK edge: • 0 for data to be changed on the leading edge and captured on the following edge. • 1 for data captured on the leading edge and changed on the following edge.

6.3.40 Function spi_set_clock_polarity()

Set the SPI clock (SPCK) inactive state.

```
void spi_set_clock_polarity(
  Spi * p_spi,
  uint32_t ul_pcs_ch,
  uint32_t ul_polarity)
```

Table 6-53. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer
[in]	ul_pcs_ch	Peripheral Chip Select channel (range 0 to 3)
[in]	ul_polarity	Inactive clock state of SPCK: • 0 for logic level zero • 1 for logic level one

6.3.41 Function spi_set_delay_between_chip_select()

Set the delay between chip selects (in MCK clocks).

```
void spi_set_delay_between_chip_select(
  Spi * p_spi,
  uint32_t ul_delay)
```

Note If DLYBCS <= 6, 6 MCK clocks will be inserted by default.

Table 6-54. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

Data direction	Parameter name	Description
[in]	ul_delay	Delay between chip selects (in number of MCK clocks)

6.3.42 Function spi_set_fixed_peripheral_select()

Set Fixed Peripheral Select. Peripheral Chip Select is controlled by SPI_MR.

```
void spi_set_fixed_peripheral_select(
  Spi * p_spi)
```

Table 6-55. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

6.3.43 Function spi_set_lastxfer()

Issue a LASTXFER command. The next transfer is the last transfer and after that CS is de-asserted.

```
void spi_set_lastxfer(
  Spi * p_spi)
```

Table 6-56. Parameters

Data direction	Parameter name	Description
[out]	p_spi	Module hardware register base address pointer

6.3.44 Function spi_set_master_mode()

Configure the SPI module to operate in Master Mode.

```
void spi_set_master_mode(
  Spi * p_spi)
```

Table 6-57. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

6.3.45 Function spi_set_peripheral_chip_select_value()

Set the Peripheral Chip Select (NPCSx) value in 4 to 16 line decoder mode.

```
void spi_set_peripheral_chip_select_value(
  Spi * p_spi,
```

```
  uint32_t ul_value)
```

Table 6-58. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer
[in]	ul_value	Peripheral Chip Select value If PCS decode mode is not used, use spi_get_pcs to build the value to use. On reset the decode mode is not enabled. The decode mode can be enabled/ disabled by the following functions: spi_enable_peripheral_select_decode, spi_disable_peripheral_select_decode.

6.3.46 Function spi_set_slave_mode()

Configure the SPI module to operate in Slave Mode.

```
void spi_set_slave_mode(
  Spi * p_spi)
```

Table 6-59. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

6.3.47 Function spi_set_transfer_delay()

Configure the timing for SPI data transfer.

```
void spi_set_transfer_delay(
  Spi * p_spi,
  uint32_t ul_pcs_ch,
  uint8_t uc_dlybs,
  uint8_t uc_dlybct)
```

Table 6-60. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Pointer to an SPI instance
[in]	ul_pcs_ch	Peripheral Chip Select channel (range 0 to 3)
[in]	uc_dlybs	Delay before SPCK (in MCK clocks)
[in]	uc_dlybct	Delay between consecutive transfers (in MCK clocks)

6.3.48 Function spi_set_variable_peripheral_select()

Set Variable Peripheral Select. Peripheral Chip Select can be controlled by SPI_TDR.

```
void spi_set_variable_peripheral_select(
  Spi * p_spi)
```

Table 6-61. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer

6.3.49 Function spi_set_writeprotect()

Enable or disable write protection of the SPI registers.

```
void spi_set_writeprotect(
  Spi * p_spi,
  uint32_t ul_enable)
```

Table 6-62. Parameters

Data direction	Parameter name	Description
[out]	p_spi	Module hardware register base address pointer
[in]	ul_enable	1 to enable, 0 to disable

6.3.50 Function spi_write()

Write SPI data with specified peripheral chip select value.

```
spi_status_t spi_write(
  Spi * p_spi,
  uint16_t us_data,
  uint8_t uc_pcs,
  uint8_t uc_lspi)
```

Table 6-63. Parameters

Data direction	Parameter name	Description
[in, out]	p_spi	Module hardware register base address pointer
[in]	us_data	The data to transmit
[in]	uc_pcs	Peripheral Chip Select value. Only valid when Variable Peripheral Select is enabled, set to 0 otherwise.
[in]	uc_last	Indicate whether this data item is the last. Only valid when Variable Peripheral Select is enabled, set to 0 otherwise.

Returns SPI operation error code.

Table 6-64. Return Values

Return value	Description
SPI_OK	SPI data transmitted successfully
SPI_ERROR_TIMEOUT	A Software time-out occurred while waiting to transmit

6.4 Enumeration Definitions

6.4.1 Enum spi_cs_behavior

SPI Peripheral Chip Select (NPCSx) behavior modes while transferring data in Master mode.

Table 6-65. Members

Enum value	Description
SPI_CS_KEEP_LOW	CS does not rise until a new transfer is requested on different chip select.
SPI_CS_RISE_NO_TX	CS rises if there is no more data to transfer.
SPI_CS_RISE_FORCED	CS is de-asserted systematically during a time DLYBCS.

6.4.2 Enum spi_status_t

Status codes used by the SPI driver.

Table 6-66. Members

Enum value	Description
SPI_ERROR	A non specific error occurred.
SPI_OK	No error encountered.
SPI_ERROR_TIMEOUT	An activity timeout occurred.
SPI_ERROR_ARGUMENT	Invalid argument specified.
SPI_ERROR_OVERRUN	Data overrun occurred.
SPI_ERROR_MODE_FAULT	Invalid master/slave mode specified.
SPI_ERROR_OVERRUN_AND_MODE_FAULT	Data overrun and and invalid master/slave mode specified.

7. Extra Information for Serial Peripheral Interface Driver

7.1 Acronyms

Below is a table listing the acronyms used in this module, along with their intended meanings.

Acronym	Definition
ADC	Analog to Digital Converter
CAN	Controller Area Network
CPU	Central Processor Unit
CS	Chip Select
DAC	Digital to Analog Converter
DMA	Direct Memory Access
DMAC	Direct Memory Access Controller
GPIO	General Purpose Input Output
LCD	Liquid Crystal Display
MCK	Master Clock
PDC	Peripheral DMA Controller
PDCA	Peripheral DMA Controller (SAM4L devices)
QSG	Quick Start Guide
UART	Universal Asynchronous Receiver Transmitter

7.2 Dependencies

This driver has the following dependencies:

- General Purpose I/O (GPIO) Driver
- System Clock Management (sysclk)

7.3 Errata

There are no errata related to this driver.

7.4 Module History

An overview of the module history is presented in the table below, with details on the enhancements and fixes made to the module since its first release. The current version of this corresponds to the newest version in the table.

Changelog
Initial document release

8. Examples for SPI Driver

This is a list of the available Quick Start Guides (QSGs) and example applications for SAM4 Serial Peripheral Interface (SPI). QSGs are simple examples with step-by-step instructions to configure and use this driver in a selection of use cases. Note that QSGs can be compiled as a standalone application or be added to the user application.

- Quick Start Guide for SPI driver
- Serial Peripheral Interface (SPI) Master/Slave example
- Serial Peripheral Interface (SPI) DMA slave example
- Serial Peripheral Interface (SPI) PDC example

8.1 Quick Start Guide for SPI driver

This is the quick start guide for the SAM4 Serial Peripheral Interface (SPI), with step-by-step instructions on how to configure and use the driver in a selection of use cases.

The use cases contain several code fragments. The code fragments in the steps for setup can be copied into a custom initialization function, while the steps for usage can be copied into, e.g. the main application function.

8.1.1 Basic Use Case

In this basic use case, the SPI module are configured for:

- Master mode
- Interrupt-based handling

8.1.1.1 Prerequisites

- System Clock Management (sysclock)

8.1.2 Setup Steps

8.1.2.1 Example Code

Add the following into your application C-file:

```
*   void spi_master_init(Spi *p_spi)
*   {
*       spi_enable_clock(p_spi);
*       spi_reset(p_spi);
*       spi_set_master_mode(p_spi);
*       spi_disable_mode_fault_detect(p_spi);
*       spi_disable_loopback(p_spi);
*       spi_set_peripheral_chip_select_value(p_spi,
*                                             spi_get_pcs(DEFAULT_CHIP_ID));
*       spi_set_fixed_peripheral_select(p_spi);
*       spi_disable_peripheral_select_decode(p_spi);
*       spi_set_delay_between_chip_select(p_spi, CONFIG_SPI_MASTER_DELAY_BCS);
*   }
*   void spi_master_setup_device(Spi *p_spi, struct spi_device *device,
*       spi_flags_t flags, uint32_t baud_rate, board_spi_select_id_t sel_id)
*   {
*       spi_set_transfer_delay(p_spi, device->id, CONFIG_SPI_MASTER_DELAY_BS,
*                              CONFIG_SPI_MASTER_DELAY_BCT);
*
*       spi_set_bits_per_transfer(p_spi, device->id, CONFIG_SPI_MASTER_BITS_PER_TRANSFER);
*       spi_set_baudrate_div(p_spi, device->id,
*                            spi_calc_baudrate_div(baud_rate, sysclk_get_cpu_hz()));
```

```
 *
 *       spi_configure_cs_behavior(p_spi, device->id, SPI_CS_KEEP_LOW);
 *
 *       spi_set_clock_polarity(p_spi, device->id, flags >> 1);
 *       spi_set_clock_phase(p_spi, device->id, ((flags & 0x1) ^ 0x1));
 *   }
```

8.1.2.2 Workflow

1. Initialize the SPI in Master Mode:

```
void spi_master_init(SPI_EXAMPLE);
```

2. Set up an SPI device:

```
void spi_master_setup_device(SPI_EXAMPLE, &SPI_DEVICE_EXAMPLE,
*              SPI_MODE_0, SPI_EXAMPLE_BAUDRATE, 0);
```

Note The returned device descriptor structure must be passed to the driver whenever that device should be used as the current slave device.

3. Enable SPI module:

```
spi_enable(SPI_EXAMPLE);
```

8.2 Serial Peripheral Interface (SPI) Master/Slave example

8.2.1 Purpose

This example uses the Serial Peripheral Interface (SPI) of one EK board in Slave Mode to communicate with another EK board's SPI in Master Mode.

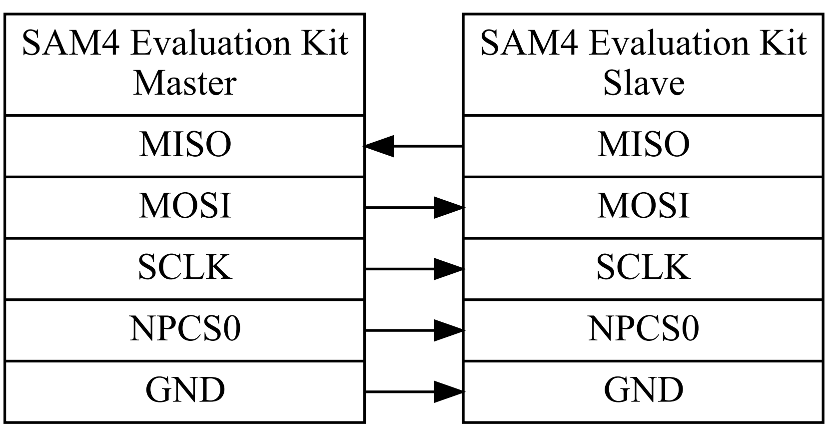
8.2.2 Requirements

This example can be used with two SAM4 evaluation kits such as the SAM4S Xplained, the SAM4S EK2, and other evaluation kits. Refer to the list of kits available for the actual device on http://www.atmel.com.

8.2.2.1 Pin Connections

Connect the spi pins from one board to another as shown in Figure 8-1: Pin Connections on page 33.

Figure 8-1. Pin Connections



8.2.3 Description

This example shows how to configure and transfer data using SPI. By default, example runs in SPI slave mode, waiting for both SPI slave data input and UART user input.

8.2.4 Main Files

- spi.c: Serial Peripheral Interface driver
- spi.h: Serial Peripheral Interface driver header file
- spi_example.c: Serial Peripheral Interface example application

8.2.5 Compilation Information

This software is written for GNU GCC and IAR Embedded Workbench for Atmel. Other compilers may or may not work.

8.2.6 Usage

1. Build the program and download it into the evaluation board
2. On the computer, open, and configure a terminal application (e.g., HyperTerminal on Microsoft Windows) with these settings:
   - 115200 baud
   - 8 bits of data
   - No parity
   - 1 stop bit
   - No flow control
3. Start the application
4. In the terminal window, the following text should appear:

```
*      -- Spi Example  --
*      -- xxxxxx-xx
*      -- Compiled: xxx xx xxxx xx:xx:xx --
*
*     Menu :
*     ------
*       0: Set SPCK = 500000 Hz
*       1: Set SPCK = 1000000 Hz
*       2: Set SPCK = 2000000 Hz
*       3: Set SPCK = 5000000 Hz
*       t: Perform SPI master
*       h: Display menu again
```

   Initially the SPI module will be configured to operate in Slave Mode.

5. Selecting menu option 't' will start the SPI transfer test:
   - Configure the SPI module as master, and set up the SPI clock
   - Send 4-byte CMD_TEST to indicate the start of a test
   - Send several 64-byte blocks, and after transmitting the next block, the content of the last block is returned and checked.

- Send CMD_STATUS command and wait for the status reports from the slave
- Send CMD_END command to indicate the end of test

6. The resulting terminal messages detail operations carried out in this SPI example, displaying success and/or error messages depending on the results of the commands.

8.3 Serial Peripheral Interface (SPI) DMA slave example

8.3.1 Purpose

This example uses the Serial Peripheral Interface (SPI) of one EK board in Slave Mode to communicate with another EK board's SPI in Master Mode using the DMAC module.

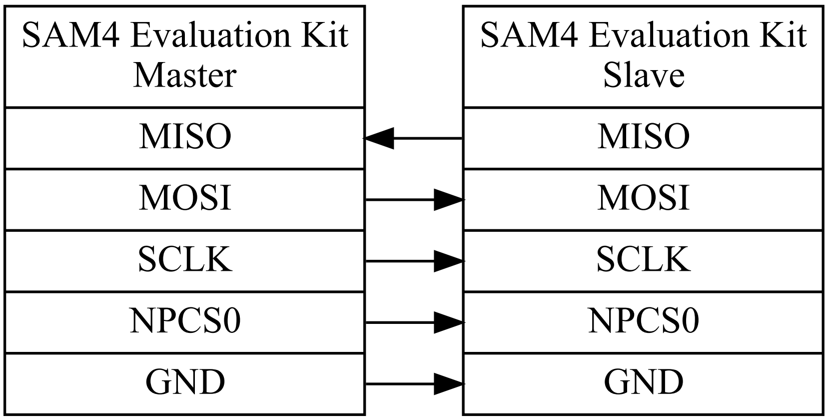
8.3.2 Requirements

This example can be used with SAM4 evaluation kits that support DMAC operation such as the SAM4E EK, and other evaluation kits. Refer to the list of kits available for the actual device on http://www.atmel.com.

8.3.2.1 Pin Connections

Connect the spi pins from one board to another as shown in Figure 8-2: Pin Connections on page 35.

Figure 8-2. Pin Connections



8.3.3 Description

This example shows how to configure and transfer data using SPI. By default, example runs in SPI slave mode, waiting for both SPI slave data input and UART user input.

8.3.4 Main Files

- spi.c: Serial Peripheral Interface driver
- spi.h: Serial Peripheral Interface driver header file
- spi_dmac_slave_example.c: Serial Peripheral Interface DMAC example application

8.3.5 Compilation Information

This software is written for GNU GCC and IAR Embedded Workbench® for Atmel®. Other compilers may or may not work.

8.3.6 Usage

1. Build the program and download it into the evaluation board
2. On the computer, open, and configure a terminal application (e.g., HyperTerminal on Microsoft® Windows®) with these settings:

- 115200 baud
- 8 bits of data
- No parity
- 1 stop bit
- No flow control

3. Start the application
4. In the terminal window, the following text should appear:

```
*      -- Spi DMA Slave Example --
*      -- xxxxxx-xx
*      -- Compiled: xxx xx xxxx xx:xx:xx --
*
*     SPI works in slave mode.
*
*     Menu :
*     ------
*       0: Set SPCK = 500000 Hz
*       1: Set SPCK = 1000000 Hz
*       2: Set SPCK = 2000000 Hz
*       3: Set SPCK = 5000000 Hz
*       s: Configure SPI as slave
*       m: Configure SPI as master
*       t: Perform communication sequence
*       h: Display menu again
```

Initially the SPI module will be configured to operate in Slave Mode.

5. Selecting menu option 't' will start the SPI transfer test:
   - Configure the SPI module as master, and set up the SPI clock
   - Send 4-byte CMD_TEST to indicate the start of a test
   - Send several 64-byte blocks, and after transmitting the next block, the content of the last block is returned and checked.
   - Send CMD_STATUS command and wait for the status reports from the slave
   - Send CMD_END command to indicate the end of test
6. The resulting terminal messages detail operations carried out in this SPI example, displaying success and/or error messages depending on the results of the commands.

8.4 Serial Peripheral Interface (SPI) PDC example

8.4.1 Purpose

This example uses the Serial Peripheral Interface (SPI) of one EK board in Slave Mode to communicate with another EK board's SPI in Master Mode using the PDC functionality.

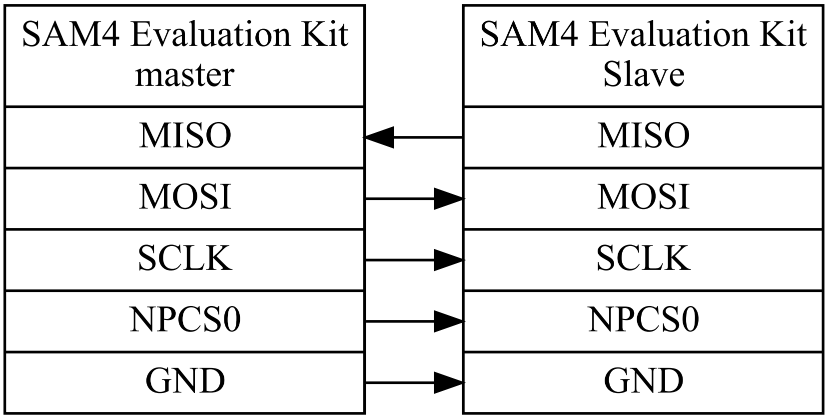
8.4.2 Requirements

This example can be used with two SAM4 evaluation kits such as the SAM4N Xplained Pro, SAM4S EK and other evaluation kits. Refer to the list of kits available for the actual device on http://www.atmel.com.

8.4.2.1 Pin Connections

Connect the spi pins from one board to another as shown in Figure 8-3: Pin Connections on page 37.

Figure 8-3. Pin Connections



8.4.3 Description

This example shows how to configure and transfer data using SPI. By default, example runs in SPI slave mode, waiting for both SPI slave data input and UART user input.

8.4.4 Main Files

- spi.c: Serial Peripheral Interface driver
- spi.h: Serial Peripheral Interface driver header file
- spi_pdc_example.c: Serial Peripheral Interface PDC example application

8.4.5 Compilation Information

This software is written for GNU GCC and IAR Embedded Workbench for Atmel. Other compilers may or may not work.

8.4.6 Usage

1. Build the program and download it into the evaluation board
2. On the computer, open, and configure a terminal application (e.g., HyperTerminal on Microsoft Windows) with these settings:
   - 115200 baud
   - 8 bits of data
   - No parity
   - 1 stop bit
   - No flow control
3. Start the application
4. In the terminal window, the following text should appear:

```
*      -- Spi Pdc Example  --
*      -- xxxxxx-xx
```

```
*      -- Compiled: xxx xx xxxx xx:xx:xx --
*
*     Menu :
*     ------
*       0: Set SPCK = 500000 Hz
*       1: Set SPCK = 1000000 Hz
*       2: Set SPCK = 2000000 Hz
*       3: Set SPCK = 5000000 Hz
*       t: Perform SPI master
*       h: Display menu again
```

Initially the SPI module will be configured to operate in Slave Mode.

5. Selecting menu option 't' will start the SPI transfer test:
   - Configure the SPI module as master, and set up the SPI clock
   - Send a 64-byte data block and check that it is received correctly
6. The resulting terminal messages detail operations carried out in this SPI example, displaying success and/or error messages depending on the results of the commands.

Index

E

F

M

T

Document Revision History

Doc. Rev.	Date	Comments
42290A	05/2014	Initial document release

Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**