SPI Driver

User Guide

V1.00.01

Support Chips:

ISD9160

Support Platforms:

Nuvoton

Content

1. SPI Driver

1.1 SPI Introduction

The Serial Peripheral Interface (SPI) is a synchronous serial data communication protocol which operates in full duplex mode. Devices communicate in master/slave mode with 4-wire bi-directional interface. The ISD91XX series contains an SPI controller performing a serial-to-parallel conversion of data received from an external device, and a parallel-to-serial conversion of data transmitted to an external device. The SPI controller can be set as a master with up to 2 slave select (SSB) address lines to access two slave devices; it also can be set as a slave controlled by an off-chip master device.

1.2 SPI Feature

- Supports master or slave mode operation.
- Supports one or two channels of serial data.
- Configurable word length of up to 32 bits. Up to two words can be transmitted per a transaction, giving a maximum of 64 bits for each data transaction.
- Provide burst mode operation.
- MSB or LSB first transfer.
- 2 device/slave select lines in master mode, single device/slave select line in slave mode.
- Byte or word Sleep Suspend Mode.
- Support dual FIFO mode.
- PDMA access support.

1.3 Type Definition

E_DRVSPI_PORT

Enumeration Identifier	Value	Description
eDRVSPI_PORT0	0	SPI port 0

E_DRVSPI_MODE

Enumeration Identifier	Value	Description
eDRVSPI_MASTER	0	Master mode
eDRVSPI_SLAVE	1	Slave mode
eDRVSPI_JOYSTICK	2	Joystick mode

E_DRVSPI_TRANS_TYPE

Enumeration Identifier	Value	Description
eDRVSPI_TYPE0	0	SPI transfer type 0
eDRVSPI_TYPE1	1	SPI transfer type 1
eDRVSPI_TYPE2	2	SPI transfer type 2
eDRVSPI_TYPE3	3	SPI transfer type 3
eDRVSPI_TYPE4	4	SPI transfer type 4
eDRVSPI_TYPE5	5	SPI transfer type 5
eDRVSPI_TYPE6	6	SPI transfer type 6
eDRVSPI_TYPE7	7	SPI transfer type 7

E_DRVSPI_ENDIAN

Enumeration Identifier	Value	Description
eDRVSPI_LSB_FIRST	0	Send LSB first
eDRVSPI_MSB_FIRST	1	Send MSB first

E_DRVSPI_SSLTRIG

Enumeration Identifier	Value	Description
eDRVSPI_EDGE_TRIGGER	0	Edge trigger
eDRVSPI_LEVEL_TRIGGER	1	Level trigger

E_DRVSPI_SS_ACT_TYPE

Enumeration Identifier	Value	Description
eDRVSPI_ACTIVE_LOW_FALLING	0	Low-Level/Falling-Edge active
eDRVSPI_ACTIVE_HIGH_RISING	1	High-Level/Rising-Edge active

E_DRVSPI_SLAVE_SEL

Enumeration Identifier	Value	Description
eDRVSPI_NONE	0	No slave device was selected
eDRVSPI_SS0	1	Select the 1st slave select pin
eDRVSPI_SS1	2	Select the 2nd slave select pin
eDRVSPI_SS0_SS1	3	Both pins are selected

E_DRVSPI_JOYSTICK_INT_FLAG

Enumeration Identifier	Value	Description
eDRVSPI_JOYSTICK_CS_ACTIVE	0	Chip Select is actived.
eDRVSPI_JOYSTICK_DATA_READY	1	8-byte data available in the buffer.
eDRVSPI_JOYSTICK_CS_DEACT	2	Chip Select is de-actived.
eDRVSPI_JOYSTICK_NONE	3	None.

E_DRVSPI_JOYSTICK_RW_MODE

Enumeration Identifier	Value	Description
eDRVSPI_JOYSTICK_TRANSMIT_MODE	0	Master writes data to slave.
eDRVSPI_JOYSTICK_RECEIVE_MODE	1	Master read data from slave.

E_DRVSPI_DMA_MODE

Enumeration Identifier	Value	Description
eDRVSPI_TX_DMA	0	Enable Tx DMA
eDRVSPI_RX_DMA	1	Enable Rx DMA

SPI ERROR CODE

Enumeration Identifier	Value	Description
E_DRVSPI_ERR_BURST_CNT	0	Wrong Burst Number
E_DRVSPI_ERR_TRANSMIT_INTERVAL	1	Wrong Transmit Number
E_DRVSPI_ERR_BIT_LENGTH	2	Wrong Bit Length
E_DRVSPI_ERR_INIT	3	Init Fail

E_DRVSPI_ERR_BUSY	4	SPI is busy
E_DRVSPI_ERR_PORT	5	SPI Port does not exist

1.4 Driver Functions

DrvSPI_Open

Prototype

```
ERRCODE
DrvSPI_Open(
E_DRVSPI_PORT eSpiPort,
E_DRVSPI_MODE eMode,
E_DRVSPI_TRANS_TYPE eType,
int32_t i32BitLength
);
```

Description

This function is used to open SPI module. It decides the SPI to work in master or slave mode, SPI bus timing and bit length per transfer. The automatic slave select function will be enabled.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eMode [in]

To work in Master (eDRVSPI_MASTER) or Slave (eDRVSPI_SLAVE) mode.

eType [in]

Transfer type, i.e. the bus timing. It could be eDRVSPI_TYPE0~eDRVSPI_TYPE7.

eDRVSPI_TYPE0: the clock idle state is low; drive data at rising-edge of serial clock; latch data at rising-edge of serial clock.

eDRVSPI_TYPE1: the clock idle state is low; drive data at falling-edge of serial clock; latch data at rising-edge of serial clock.

eDRVSPI_TYPE2: the clock idle state is low; drive data at rising-edge of serial clock; latch data at falling-edge of serial clock.

eDRVSPI_TYPE3: the clock idle state is low; drive data at falling-edge of serial clock; latch data at falling-edge of serial clock.

eDRVSPI_TYPE4: the clock idle state is high; drive data at rising-edge of serial clock; latch data at rising-edge of serial clock.

eDRVSPI_TYPE5: the clock idle state is high; drive data at falling-edge of serial clock; latch data at rising-edge of serial clock.

eDRVSPI_TYPE6: the clock idle state is high; drive data at rising-edge of serial clock; latch data at falling-edge of serial clock.

eDRVSPI_TYPE7: the clock idle state is high; drive data at falling-edge of serial clock; latch data at falling-edge of serial clock.

i32BitLength

Bit length per transaction. The range is 1~32.

Include

Driver\DrvSPI.h

Return Value

E_SUCCESS : Success.

E_DRVSPI_ERR_INIT: The specified SPI port has been opened before.

E_DRVSPI_ERR_BIT_LENGTH: The bit length is out of range.

E_DRVSPI_ERR_BUSY: The specified SPI port is in busy status.

Example

```
/* Configure SPI0 as a master, 32-bit transaction */
DrvSPI_Open(eDRVSPI_PORT0, eDRVSPI_MASTER, eDRVSPI_TYPE1, 32);
```

DrvSPI_Close

Prototype

```
void
DrvSPI_Close(
E_DRVSPI_PORT eSpiPort
);
```

Description

Close the specified SPI module and disable the SPI interrupt.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Return Value

None

Example

```
/* Close SPI0 */
DrvSPI_Close(eDRVSPI_PORT0);
```

DrvSPI_Set2BitSerialDataIOMode

Prototype

void

DrvSPI_Set2BitSerialDataIOMode(

E_DRVSPI_PORT eSpiPort,

BOOL bEnable

);

Description

Set 2-bit transfer mode.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

bEnable [in]

Enable(TRUE)/Disable(FALSE)

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Enable 2-bit transfer mode of SPI0 */

DrvSPI_Set2BitTransferMode(eDRVSPI_PORT0, TRUE);

DrvSPI_SetEndian

Prototype

void

DrvSPI_SetEndian(

E_DRVSPI_PORT eSpiPort,

E_DRVSPI_ENDIAN eEndian

);

Description

This function is used to configure the bit order of each transaction.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eEndian [in]

Specify LSB first(eDRVSPI_LSB_FIRST) or MSB first (eDRVSPI_MSB_FIRST).

Include

Driver\DrvSPI.h

Return Value

None

Example

/* The transfer order of SPI0 is LSB first */

DrvSPI_SetEndian(eDRVSPI_PORT0, eDRVSPI_LSB_FIRST);

DrvSPI_SetBitLength

Prototype

ERRCODE

DrvSPI_SetBitLength(

E_DRVSPI_PORT eSpiPort,

int32_t i32BitLength

);

Description

This function is used to configure the bit length of SPI transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

i32BitLength

Specify the bit length (1~32 bits).

Include

Driver\DrvSPI.h

Return Value

E_SUCCESS: Success.

E_DRVSPI_ERR_BIT_LENGTH: The bit length is out of range.

Example

/* The transfer bit length of SPI0 is 8-bit */

DrvSPI_SetBitLength(eDRVSPI_PORT0, 8);

DrvSPI_SetByteSleep

Prototype

ERRCODE

DrvSPI_SetByteSleep(

E_DRVSPI_PORT eSpiPort,

BOOL bEnable

);

Description

Enable/disable Byte Sleep function. The Byte Sleep function is supported only in word (32 bits) transaction mode.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

bEnable [in]

Enable(TRUE)/Disable(FALSE)

Include

Driver\DrvSPI.h

Return Value

E_SUCCESS: Success.

E_DRVSPI_ERR_BIT_LENGTH: The bit length is not 32 bits.

Example

/* Enable SPI Port 0 byte sleep function */

DrvSPI_SetByteSleep(eDRVSPI_PORT0, TRUE);

DrvSPI_SetByteEndian

Prototype

ERRCODE

DrvSPI_SetByteEndian(

E_DRVSPI_PORT eSpiPort,

BOOL bEnable

);

Description

Enable/disable Byte Reorder function. The Byte Reorder function is supported only in 16-, 24- and 32-bit transaction mode.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

bEnable [in]

Enable(TRUE)/Disable(FALSE)

Include

Driver\DrvSPI.h

Return Value

E_SUCCESS: Success.

E_DRVSPI_ERR_BIT_LENGTH: The bit length is not 16-, 24- or 32-bit.

Example

/* Enable SPI Port 0 byte endian function */

DrvSPI_SetByteEndian (eDRVSPI_PORT0, TRUE);

DrvSPI_SetTriggerMode

Prototype

void

DrvSPI_SetTriggerMode(

E_DRVSPI_PORT eSpiPort,

E_DRVSPI_SSLTRIG eSSTriggerMode

);

Description

Set the trigger mode of slave select pin. In master mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eSSTriggerMode [in]

Specify the trigger mode.

eDRVSPI_EDGE_TRIGGER: edge trigger.

eDRVSPI_LEVEL_TRIGGER: level trigger.

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Level trigger */

DrvSPI_SetTriggerMode(eDRVSPI_PORT0, eDRVSPI_LEVEL_TRIGGER);

DrvSPI_SetSlaveSelectActiveLevel

Prototype

void

DrvSPI_SetSlaveSelectActiveLevel(

E_DRVSPI_PORT eSpiPort,

E_DRVSPI_SS_ACT_TYPE eSSActType

);

Description

Set the active level of slave select.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eSSActType [in]

Select the active type of slave select pin.

eDRVSPI_ACTIVE_LOW_FALLING:

Slave select pin is active low in level-trigger mode; or falling-edge trigger in edge-trigger mode.

eDRVSPI_ACTIVE_HIGH_RISING:

Slave select pin is active high in level-trigger mode; or rising-edge trigger in edge-trigger mode.

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Configure the active level of SPI0 slave select pin */

DrvSPI_SetSlaveSelectActiveLevel(,eDRVSPI_PORT0, eDRVSPI_ACTIVE_LOW_FALLING);

DrvSPI_GetLevelTriggerStatus

Prototype

BOOL

DrvSPI_GetLevelTriggerStatus(

E_DRVSPI_PORT eSpiPort

);

Description

This function is used to get the level-trigger transmission status of slave device.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Return Value

TRUE: The transaction number and transferred bit length met the specified requirements.

FALSE: The transaction number or transferred bit length of one transaction doesn’t meet the specified requirements.

Example

```
/* Level trigger */
DrvSPI_SetTriggerMode(eDRVSPI_PORT0, eDRVSPI_LEVEL_TRIGGER);
.
.
/* Check the level-trigger transmission status */
if(DrvSPI_GetLevelTriggerStatus(eDRVSPI_PORT0))
    DrvSPI_DumpRxRegister(eDRVSPI_PORT0, &au32DestinationData[u32Datacount],1); /*Read Rx
Buffer */
```

DrvSPI_EnableAutoCS

Prototype

```
void
DrvSPI_EnableAutoCS(
E_DRVSPI_PORT eSpiPort,
E_DRVSPI_SLAVE_SEL eSlaveSel
);
```

Description

This function is used to enable the automatic slave select function and select the slave select pins. The automatic slave select means the SPI will set the slave select pin to active state when transferring data and set the slave select pin to inactive state when one transfer is finished. For some devices, the slave select pin may need to be kept at active state for many transfers. User should disable the automatic slave select function and control the slave select pin manually for these devices. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eSlaveSel

Select the slave select pins which will be used.

eDRVSPI_NONE	: no slave was selected.
eDRVSPI_SS0	: the SS0 was selected.
eDRVSPI_SS1	: the SS1 was selected.

eDRVSPI_SS0_SS1	: both SS0 and SS1 were selected.

Include

Driver\DrvSPI.h

Return Value

None

Example

```
/* Enable the automatic slave slave select function of SS0 */
DrvSPI_EnableAutoSS(eDRVSPI_PORT0, eDRVSPI_SS0);
```

DrvSPI_EnableDMABurstSS

Prototype

```
void
DrvSPI_EnableDMABurstSS(
E_DRVSPI_PORT eSpiPort,
E_DRVSPI_SLAVE_SEL eSlaveSel
);
```

Description

Enable DMA Automatic SS function. When enabled, interface will automatically generate a SS signal for an entire PDMA access transaction.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eSlaveSel

Select the slave select pins which will be used.

eDRVSPI_NONE	: no slave was selected.
eDRVSPI_SS0	: the SS0 was selected.
eDRVSPI_SS1	: the SS1 was selected.
eDRVSPI_SS0_SS1	: both SS0 and SS1 were selected.

Include

Driver\DrvSPI.h

Return Value

None

Example

```
DrvSPI_EnableDMABurstSS(eDRVSPI_PORT0, eDRVSPI_SS0);
```

DrvSPI_DisableDMABurstSS

Prototype

void

DrvSPI_DisableDMABurstSS(

E_DRVSPI_PORT eSpiPort,

E_DRVSPI_SLAVE_SEL eSlaveSel

);

Description

Disable DMA Automatic SS function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eSlaveSel

Select the slave select pins which will be used.

eDRVSPI_NONE	: no slave was selected.
eDRVSPI_SS0	: the SS0 was selected.
eDRVSPI_SS1	: the SS1 was selected.
eDRVSPI_SS0_SS1	: both SS0 and SS1 were selected.

Include

Driver\DrvSPI.h

Return Value

None

Example

DrvSPI_DisableDMABurstSS(eDRVSPI_PORT0, eDRVSPI_SS0);

DrvSPI_DisableAutoCS

Prototype

void

DrvSPI_DisableAutoCS(

E_DRVSPI_PORT eSpiPort

);

Description

This function is used to disable the automatic slave selection function. If user wants to keep the slave select signal at active state during multiple words data transfer, user can disable the automatic slave selection function and control the slave select signal manually. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Disable the automatic slave select function of SPI0*/

DrvSPI_DisableAutoSS(eDRVSPI_PORT0);

DrvSPI_SetCS

Prototype

void

DrvSPI_SetCS(

E_DRVSPI_PORT eSpiPort,

E_DRVSPI_SLAVE_SEL eSlaveSel

);

Description

Configure the slave select pins. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eSlaveSel

In automatic slave select operation, use this parameter to select the slave select pins which will be used.

In manual slave slave select operation, the specified slave select pins will be set to active state.

It could be eDRVSPI_NONE, eDRVSPI_SS0, eDRVSPI_SS1, eDRVSPI_SS0_SS1.

eDRVSPI_NONE	: no slave was selected.
eDRVSPI_SS0	: the SS0 was selected.
eDRVSPI_SS1	: the SS1 was selected.
eDRVSPI_SS0_SS1	: both SS0 and SS1 were selected.

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Disable the automatic slave select function of SPI0 */

DrvSPI_DisableAutoCS(eDRVSPI_PORT0);

/* Set the SS0 pin to active state */

DrvSPI_SetCS(eDRVSPI_PORT0, eDRVSPI_SS0);

DrvSPI_ClrCS

Prototype

void

DrvSPI_ClrCS(

E_DRVSPI_PORT eSpiPort,

E_DRVSPI_SLAVE_SEL eSlaveSel

);

Description

Set the specified slave select pins to inactive state. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eSlaveSel

Specify slave select pins.

eDRVSPI_NONE	: no slave was selected.
eDRVSPI_SS0	: the SS0 was selected.
eDRVSPI_SS1	: the SS1 was selected.
eDRVSPI_SS0_SS1	: both SS0 and SS1 were selected.

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Disable the automatic slave select function of SPI0 */

DrvSPI_DisableAutoCS(eDRVSPI_PORT0);

/* Set the SS0 pin to inactive state */

DrvSPI_ClrCS(eDRVSPI_PORT0, eDRVSPI_SS0);

DrvSPI_Busy

Prototype

BOOL

DrvSPI_Busy(

E_DRVSPI_PORT eSpiPort

);

Description

Check the busy status of the specified SPI port.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Return Value

TRUE: The SPI port is in busy.

FALSE: The SPI port is not in busy.

Example

```
/* set the GO_BUSY bit of SPI0 */
DrvSPI_SetGo(eDRVSPI_PORT0);
/* Check the busy status of SPI0 */
while(DrvSPI_Busy(eDRVSPI_PORT0));
```

DrvSPI_BurstTransfer

Prototype

```
ERRCODE
DrvSPI_BurstTransfer(
E_DRVSPI_PORT eSpiPort,
int32_t i32BurstCnt,
int32_t i32Interval
);
```

Description

Configure the burst transfer settings. If i32BurstCnt is set to 2, it performs burst transfer. SPI controller will transfer two successive transactions. The suspend interval length between the twp transactions is determined by the value of i32Interval. In slave mode, the setting of i32Interval is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

i32BurstCnt [in]

Specify the transaction number in one transfer. It could be 1 or 2.

i32Interval [in]

Suspend interval length. Specify the number of SPI clock cycle between successive transactions. The range of this setting value is 2~17.

Include

Driver\DrvSPI.h

Return Value

E_SUCCESS: Success.

E_DRVSPI_ERR_BURST_CNT: The burst count is out of range.

E_DRVSPI_ERR_TRANSMIT_INTERVAL: The suspend interval setting is out of range.

Example

/* Configure the SPI0 burst transfer mode; two transaction in one transfer; 10 delay clocks between the transactions. */

DrvSPI_BurstTransfer(eDRVSPI_PORT0,2,10);

DrvSPI_SetClock

Prototype

uint32_t

DrvSPI_SetClock(

E_DRVSPI_PORT eSpiPort,

uint32_t u32Clock1,

uint32_t u32Clock2

);

Description

Configure the frequency of SPI clock. In master mode, the output frequency of serial clock is programmable. If the variable clock function is enabled, the output pattern of serial clock is defined in VARCLK. If the bit pattern of VARCLK is ‘0’, the output frequency of SPICLK is equal to the frequency of variable clock 1. Otherwise, the output frequency is equal to the frequency of variable clock 2. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

u32Clock1 [in]

Specify the SPI clock rate in Hz. It’s the clock rate of SPI engine clock and variable clock1.

u32Clock2 [in]

Specify the SPI clock rate in Hz. It’s the clock rate of variable clock 2.

Include

Driver\DrvSPI.h

Driver\DrvSYS.h

Return Value

The actual clock rate of SPI engine clock is returned. The actual clock may different to the target SPI clock due to hardware limitation.

Example

/* SPI0 clock rate of clock 1 is 2MHz; the clock rate of clock 2 is 1MHz */

DrvSPI_SetClock(eDRVSPI_PORT0, 2000000,1000000);

DrvSPI_GetClock1

Prototype

uint32_t

DrvSPI_GetClock1(

E_DRVSPI_PORT eSpiPort

);

Description

Get the SPI engine clock rate in Hz. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Driver\DrvSYS.h

Return Value

The frequency of SPI bus engine clock. The unit is Hz.

Example

/* Get the engine clock rate of SPI0 */

printf(“SPI clock rate:%dHz\n”,DrvSPI_GetClock1(eDRVSPI_PORT0));

DrvSPI_GetClock2

Prototype

uint32_t

DrvSPI_GetClock2(

E_DRVSPI_PORT eSpiPort

);

Description

Get the clock rate of variable clock 2 in Hz. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Return Value

The frequency of variable clock 2. The unit is Hz.

Example

```
/* Get the clock rate of SPI0 variable clock 2 */
printf(“SPI clock rate of variable clock 2:%dHz\n”, DrvSPI_GetClock2(eDRVSPI_PORT0));
```

DrvSPI_SetVariableClockPattern

Prototype

```
void
DrvSPI_SetVariableClockPattern(
E_DRVSPI_PORT eSpiPort,
uint32_t u32Pattern
);
```

Description

Set the variable clock function. The output pattern of serial clock is defined in VARCLK register. A two-bit combination in the VARCLK defines the pattern of one serial clock cycle. The bit field VARCLK[31:30] defines the first clock cycle of SPICLK. The bit field VARCLK[29:28] defines the second clock cycle of SPICLK and so on.

Note that when enable the variable clock function, the setting of transfer bit length must be programmed as 0x10(16 bits mode) only. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

u32Pattern [in]

Specify the variable clock pattern.

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Set the variable clock pattern */

DrvSPI_SetVariableClockPattern(eDRVSPI_PORT0, 0x007FFF87);

DrvSPI_SetVariableClockFunction

Prototype

void

DrvSPI_SetVariableClockFunction(

E_DRVSPI_PORT eSpiPort,

BOOL bEnable

);

Description

Enable/Disable the variable clock function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

bEnable [in]

Enable(TRUE)/Disable(FALSE)

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Enable the variable clock function */

DrvSPI_SetVariableClockFunction(eDRVSPI_PORT0, TRUE);

DrvSPI_EnableInt

Prototype

void

DrvSPI_EnableInt(

E_DRVSPI_PORT eSpiPort,

PFN_DRVSPI_CALLBACK pfnCallback,

uint32_t u32UserData

);

Description

Enable the SPI interrupt of the specified SPI port and install the callback function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

pfnCallback [in]

The callback function of the corresponding SPI interrupt.

u32UserData [in]

The parameter which will be passed to the callback function.

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Enable the SPI0 interrupt and install the callback function. The parameter 0 will be passed to the callback function. */

DrvSPI_EnableInt(eDRVSPI_PORT0, SPI0_Callback, 0);

DrvSPI_DisableInt

Prototype

void

DrvSPI_DisableInt(

E_DRVSPI_PORT eSpiPort

);

Description

Disable the SPI interrupt.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Disable the SPI0 interrupt */

DrvSPI_DisableInt(eDRVSPI_PORT0);

DrvSPI_SingleReadWrite

Prototype

BOOL

DrvSPI_SingleReadWrite(

E_DRVSPI_PORT eSpiPort,

uint32_t *pu32Data,

uint32_t pu32DataIn

);

Description

Read data from SPI Rx registers and trigger SPI for next transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

pu32DataIn [in]

Write data to the SPI bus

pu32Data [out]

Store the data got from the SPI bus.

Include

Driver\DrvSPI.h

Return Value

TRUE: The data stored in pu32Data is valid.

FALSE: The data stored in pu32Data is invalid.

Example

/* Write TxData for transmit and read data into RxData*/

DrvSPI_SingleReadWrite(eDRVSPI_PORT0, TxData, &RxData);

DrvSPI_SingleRead

Prototype

BOOL

DrvSPI_SingleRead(

E_DRVSPI_PORT eSpiPort,

uint32_t *pu32Data

);

Description

Read data from SPI RX registers and trigger SPI for next transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

pu32Data [out]

A buffer pointer. This buffer is used for storing the data got from SPI bus.

Include

Driver\DrvSPI.h

Return Value

TRUE: The data stored in pu32Data is valid.

FALSE: The data stored in pu32Data is invalid.

Example

```
/* Read the previous retrieved dat and trigger next transfer. */
unit32 u32DestinationData;
DrvSPI_SingleRead(eDRVSPI_PORT0, & u32DestinationData);
```

DrvSPI_SingleWrite

Prototype

```
BOOL
DrvSPI_SingleWrite(
E_DRVSPI_PORT eSpiPort,
uint32_t *pu32Data
);
```

Description

Write data to SPI TX0 register and trigger SPI to start transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

pu32DataIn [in]

Write data to the SPI bus

Include

Driver\DrvSPI.h

Return Value

TRUE: The data stored in pu32Data has been transferred.

FALSE: The SPI is busy. The data stored in pu32Data has not been transferred.

Example

/* Write the data stored in u32SourceData to TX buffer of SPI0 and trigger SPI0 to start transfer. */

unit32 u32SourceData;

DrvSPI_SingleWrite(eDRVSPI_PORT0, &u32SourceData);

DrvSPI_BurstRead

Prototype

BOOL

DrvSPI_BurstRead(

E_DRVSPI_PORT eSpiPort,

uint32_t *pu32Buf

);

Description

Read two words of data from SPI RX registers and then trigger SPI for next transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

pu32Buf [out]

A buffer pointer. This buffer is used for storing the data got from SPI bus.

Include

Driver\DrvSPI.h

Return Value

TRUE: The data stored in pu32Buf is valid.

FALSE: The data stored in pu32Buf is invalid.

Example

/* Read two words of data from SPI0 RX registers to au32DestinationData[u32DataCount] and au32DestinationData[u32DataCount+1]. And then trigger SPI for next transfer. */

DrvSPI_BurstRead(eDRVSPI_PORT0, &au32DestinationData[u32DataCount]);

DrvSPI_BurstWrite

Prototype

BOOL

DrvSPI_BurstWrite(

E_DRVSPI_PORT eSpiPort,

uint32_t *pu32Buf

);

Description

Write two words of data to SPI TX register and then trigger SPI to start a transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

pu32Buf [in]

A buffer pointer. The data stored in this buffer will be transmitted through the SPI bus.

Include

Driver\DrvSPI.h

Return Value

TRUE: The data stored in pu32Buf has been transferred.

FALSE: The SPI is in busy. The data stored in pu32Buf has not been transferred.

Example

/* Write two words of data stored in au32SourceData[u32DataCount] and

au32SourceData[u32DataCount+1] to SPI0 TX registers. And then trigger SPI for next transfer. */

DrvSPI_BurstWrite(eDRVSPI_PORT0, & au32SourceData[u32DataCount]);

DrvSPI_DumpRxRegister

Prototype

uint32_t

DrvSPI_DumpRxRegister(

E_DRVSPI_PORT eSpiPort,

uint32_t *pu32Buf,

uint32_t u32DataCount

);

Description

Read data from RX registers. This function will not trigger a SPI data transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

pu32Buf [out]

A buffer pointer. This buffer is used for storing the data got from SPI RX registers.

u32DataCount [in]

The count of data read from RX registers. The maximum number is 2.

Include

Driver\DrvSPI.h

Return Value

The count of data actually read from Rx registers.

Example

/* Read one word of data from SPI0 RX buffer and store to au32DestinationData[u32DataCount] */

DrvSPI_DumpRxRegister(eDRVSPI_PORT0, &au32DestinationData[u32DataCount], 1);

DrvSPI_SetTxRegister

Prototype

```
uint32_t
DrvSPI_SetTxRegister(
E_DRVSPI_PORT eSpiPort,
uint32_t *pu32Buf,
uint32_t u32DataCount
);
```

Description

Write data to TX registers. This function will not trigger a SPI data transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

pu32Buf [in]

A buffer stores the data which will be written to TX registers.

u32DataCount [in]

The count of data written to TX registers.

Include

Driver\DrvSPI.h

Return Value

The count of data actually written to SPI TX registers.

Example

/* Write one word of data stored in u32Buffer to SPI0 TX register. */

DrvSPI_SetTxRegister(eDRVSPI_PORT0, &u32Buffer, 1);

DrvSPI_SetGo

Prototype

```
void
DrvSPI_SetGo(
E_DRVSPI_PORT eSpiPort
```

);

Description

In master mode, call this function can start a SPI data transfer. In slave mode, executing this function means that the slave is ready to communicate with a master.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Trigger a SPI data transfer */

DrvSPI_SetGo(eDRVSPI_PORT0);

DrvSPI_GetJoyStickIntType

Prototype

E_DRVSPI_JOYSTICK_INT_FLAG

DrvSPI_GetJoyStickIntType(

E_DRVSPI_PORT eSpiPort

);

Description

Get interrupt flag of JOYSTICK mode.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Return Value

eDRVSPI_JOYSTICK_DATA_READY: 8-byte data available in the buffer.

eDRVSPI_JOYSTICK_CS_ACTIVE: Chip Select is actived.

eDRVSPI_JOYSTICK_CS_DEACT: Chip Select is de-actived.

eDRVSPI_JOYSTICK_NONE: None.

Example

/* Get interrupt flag of JOYSTICK mode of SPI0. */

DrvSPI_GetJoyStickIntType(eDRVSPI_PORT0);

DrvSPI_SetJoyStickStatus

Prototype

void

DrvSPI_SetJoyStickStatus(

E_DRVSPI_PORT eSpiPort,

BOOL bReady

);

Description

Set the JoyStick status to ready or not ready.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

bReady [in]

TRUE -- The SPI is ready to transfer data.

FALSE -- The SPI is not ready to transfer data.

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Set the JoyStick status of SPI0 to ready */

DrvSPI_SetJoyStickStatus(eDRVSPI_PORT0, TRUE);

DrvSPI_GetJoyStickMode

Prototype

E_DRVSPI_JOYSTICK_RW_MODE

DrvSPI_GetJoyStickMode(

E_DRVSPI_PORT eSpiPort

);

Description

Get the JoyStick operation mode.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Return Value

eDRVSPI_JOYSTICK_TRANSMIT_MODE: Master writes data to slave.

eDRVSPI_JOYSTICK_RECEIVE_MODE: Master read data from slave.

Example

/* Get the JoyStick operation mode of SPI0. */

E_DRVSPI_JOYSTICK_RW_MODE eJoyStickRwMode;

eJoyStickRwMode =DrvSPI_GetJoyStickMode(eDRVSPI_PORT0);

DrvSPI_StartPDMA

Prototype

void

DrvSPI_StartPDMA(

E_DRVSPI_PORT eSpiPort,

E_DRVSPI_DMA_MODE eDmaMode,

BOOL bEnable

);

Description

Configure the DMA settings.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDmaMode [in]

Specify the DMA mode.

eDRVSPI_TX_DMA: DMA-Transmitting

eDRVSPI_RX_DMA: DMA-Receiving

bEnable [in]

TRUE: Enable DMA

FALSE: Disable DMA

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Enable the SPI0 DMA-Receiving function */

DrvSPI_StartPDMA(eDRVSPI_PORT0, eDRVSPI_RX_DMA, TRUE);

DrvSPI_SetFIFOMode

Prototype

void

DrvSPI_SetFIFOMode(

E_DRVSPI_PORT eSpiPort,

BOOL bEnable

);

Description

The SPI controller supports a dual buffer mode when SPI->CNTRL.FIFO is set as 1. In normal mode, software can only update the transmitted data when the current transmission is done. In FIFO mode, the next transmitted data can be written into the SPI_TX buffer at any time when in master mode or the GO_BUSY bit is set in slave mode. This data will load into the transmit buffer when the current transmission done.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

bEnable [in]

TRUE: Enable FIFO mode

FALSE: Disable FIFO mode

Include

Driver\DrvSPI.h

Return Value

None

Example

/* Enable the SPI FIFO mode */

DrvSPI_SetFIFOMode(eDRVSPI_PORT0, TRUE);

DrvSPI_IsRxEmpty

Prototype

BOOL

DrvSPI_IsRxEmpty(

E_DRVSPI_PORT eSpiPort

);

Description

Check the empty status of the Rx buffer of the specified SPI port.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Return Value

TRUE: Rx buffer empty.

FALSE: Rx buffer is not empty.

Example

```
/* Check SPI Rx empty status */
DrvSPI_IsRxEmpty(eDRVSPI_PORT0);
```

DrvSPI_IsRxFull

Prototype

```
BOOL
DrvSPI_IsRxFull(
E_DRVSPI_PORT eSpiPort
);
```

Description

Check the full status of the Rx buffer of the specified SPI port.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Return Value

TRUE: Rx buffer full.

FALSE: Rx buffer is not full.

Example

```
/* Check SPI Rx full status */
DrvSPI_IsRxFull(eDRVSPI_PORT0);
```

DrvSPI_IsTxEmpty

Prototype

```
BOOL
DrvSPI_IsTxEmpty(
E_DRVSPI_PORT eSpiPort
);
```

Description

Check the empty status of the Tx buffer of the specified SPI port.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Return Value

TRUE: Tx buffer empty.

FALSE: Tx buffer is not empty.

Example

```
/* Check SPI Tx empty status */
DrvSPI_IsTxEmpty(eDRVSPI_PORT0);
```

DrvSPI_IsTxFull

Prototype

```
BOOL
DrvSPI_IsTxFull(
E_DRVSPI_PORT eSpiPort
);
```

Description

Check the full status of the Tx buffer of the specified SPI port.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

Include

Driver\DrvSPI.h

Return Value

TRUE: Tx buffer full.

FALSE: Tx buffer is not full.

Example

/* Check SPI Tx full status */

DrvSPI_IsTxFull(eDRVSPI_PORT0);

DrvSPI_GetVersion

Prototype

uint32_t

DrvSPI_GetVersion(void);

Description

Get the version number of SPI driver.

Parameters

None

Include

Driver\DrvSPI.h

Return Value

Version number:

31:24	23:16	15:8	7:0
00000000	MAJOR_NUM	MINOR_NUM	BUILD_NUM

Example

printf(“Driver version:%x\n”, DrvSPI_GetVersion());

2. Revision History

Version	Date	Description
1.00.01	Mar. 2011	Preliminary SPI Driver User Guide of ISD9160