ماژول SPI میکرو LPC1768

طریقه کانفیگ:

۱. روشن کردن ماژول با یک کردن بیت PCSPI در رجیستر PCONP
۲. کلاک مورد نظر برای ماژول از طریق رجیستر PCLKSEL0 و بیت های PCLK_SPI انتخاب گردد
۳. پین های مورد نیاز از طریق رجیستر PINSEL انتخاب گردند

PINSEL0[31:30] P0.15 SCK : کلاک است که مستر آنرا تولید و برای اسلیو ارسال می کند

PINSEL1[1:0] P0.16 SSEL :

یک سیگنال Active low است که مشخص می کند کدام اسلیو انتخاب شده است این پین باید قبل از اغاز انتقال داده ها low شود و در طی فرایند ارسال اطلاعات low بماند

PINSEL1[3:2] P0.17 MISO (Master In Slave Out)

PINSEL1[5:4] P0.18 MOSI(Master Out Slave In)

برای این پینها باید فانکشن ۳ انتخاب گردد در رجیستر PINMODE مود آنها ۰ انتخاب گردد تا PULL-UP داخلی برایشان فعال گردد و در رجیستر PINMODE_OD برای آنها ۰ انتخاب گردد که یعنی OPEN DRAIN مد نظر نیست.

نکته : ماژول SSP0 به عنوان جایگزین SPI در نظر گرفته شده است و در آن واحد تنها یکی از این دو ماژول قابل استفاده هستند.

حداکثر بیت ریت دیتا 1/8 کلاک ریت ماژول می باشد و این ماژول می تواند۸ بیت تا حداکثر ۱۶ بیت را در هر ترانسفر جابجا کند.

۵ رجیستر کنترل و وضعیت در رابطه با این ماژول وجود دارند که در ادامه به بررسی آنها می پردازیم :

SPI Control Register (S0SPCR): قبل از ترانسفر اطلاعات باید تنظیم شود و کنترل ماژول را برعهده دارد

SPI Status Register (S0SPSR) : بیت های این رجیستر فقط خواندنی هستند و وضعیت ماژول را نشان می دهند.

SPI Data Register (S0SPDR) : اطلاعاتی که باید ارسال شوند در این رجیستر نوشته می شوند و همچنین بایت دریافتی اطلاعات نیز در این رجیستر ذخیره می شود

SPI Clock Counter Register (S0SPCCR) : در وضعیت مستر کنترل کلاک ریت را بر عهده دارد

SPI Interrupt Flag.(S0SPINT): اینتراپت فلگ ماژول

مراحل تنظیم ماژول در مد مستر :

۱. تنظیم کلاک
۲. تنظیم رجیستر کنترل

مراحل ارسال داده در مد مستر:

۱. چک کردن کانفیگ ماژول(اختیاری)
۲. نوشتن دیتا در رجیستر دیتا
۳. انتظار برای یک شدن بیت SPIF در رجیستر SPI Status Register که نشانه پایان ارسال است
۴. خواندن رجیستر status
۵. خواندن دیتای دریافتی
۶. رفتن به مرحله ۲ در صورت وجود اطلاعات دیگر برای ارسال

شرایط بروز خطا (Exception conditions):

Read Overrun:

این خطا موقعی رخ می دهد که بافر داخلی خواندن ماژول هنوز حاوی اطلاعاتی است که توسط پروسسور خوانده نشده اند و خواندن بایت بعدی اطلاعات به پایان می رسد در اینجا بافر داخلی حاوی اطلاعات صحیحی است که باید با اطلاعات جدید رونویسی شود که در این حالت دیتای جدید از بین می رود و بیت ROVR در رجیستر SPI Status Register یک می شود.

Write Collision :

اگر در تایم فریم ارسال اطلاعات که رجیستر دیتا پر است یعنی از زمان آغاز ارسال تا زمان یک شدن SPIF و سپس خواندن رجیستر status عمل نوشتن در رجیستر دیتا اتفاق بیفتد دیتای جدید از بین می رود و بیت WCOL در Status Register

یک می شود.

Mode Fault:

اگر در زمانی که ماژول در مد مستر است سیگنال SSEL اکتیو شود که نشان می دهد یک مستر دیگر ما را انتخاب کرده تا slave باشیم باعث یک شدن بیت MODF در Status Register می شود.

Slave Abort:

اگر ماژول در مد Slaveدر حال ارسال اطلاعات باشد و سیگنال SSEL غیر فعال شود پیش می آید و بیت ABRT در Status Register یک می شود.

پیکر بندی رجیستر ها :

SPI Clock Counter Register (S0SPCCR - 0x4002 000C):

این رجیستر در مد مستر کلاک ماژول را کنترل می کند. که باید یک عدد زوج بزرگتر از ۸ باشد کلاک ریت ماژول به صورت زیر محاسبه می شود :

Spi clock = PCLK_SPI/SPCCR0

که در PCLK_SPI در رجیستر PCLKSEL0 و بیتهای PCLK_SPI تعیین می شود.

SPI Interrupt Register (S0SPINT - 0x4002 001C)

بیت صفر این رجیستر SPIF می باشد که پرچم اینتراپت ماژول SPI استو توسط ماژول یک می شود

نکته : با یک بودن SPIE و یک شدن یکی از بیتهای SPIF , WCOL این بیت یک می شود اما تنها در صورتی که بیت فعال ساز اینتراپت یک باشد و اینتراپت SPI در NVIC فعال باشد باعث بروز اینتراپت می شود

SPI Status Register (S0SPSR - 0x4002 0004)

31:8	7	6	5	4	3	2:0
RESERVED	SPIF	WCOL	POVR	MODF	ABRT	RESERVED

SPI Control Register (S0SPCR - address 0x4002 0000)

31:12	8	7	6	5	4	3	2	1:0
RESERVED	BITS	SPIE	LSBF	MSTR	CPOL	CPHA	BIT ENABLE	RESERVED

BIT ENABLE

- ۰ : دیتا ۸ بیتی است
- ۱ : تعداد بیتهای دیتا در بیتهای ۱۱:۸ مشخص شده است.

CPHA(Clock Phase control bit)

مشخص کننده ارتباط بین کلاک و دیتا در انتقال هاست و مشخص می کند که کی ترانسفر از SLAVE شروع شود و کی پایان یابد.

• : دیتا در نخستین لبه کلاک SCK نمونه برداری می شود و انتقال دیتا با فعال و غیر فعال شدن SSEL آغاز شده و پایان می پذیرد.

۱ :دیتا در دومین لبه کلاک SCK نمونه برداری می شود یک انتقال در اولین لبه SCK شروع شده و در آخرین لیه نمونه برداری پایان می پذیرد در حالی که SSELفعال است

CPOL Clock polarity control

0 SCK is active high.

1 SCK is active low.

MSTR Master mode select

0 The SPI operates in Slave mode.
1 The SPI operates in Master mode.

LSBF

مشخص می کند که در هنگام انتقال یک بایت از کدام طرف ارسال شود

0 SPI data is transferred MSB (bit 7) first.

2 SPI data is transferred LSB (bit 0) first.

SPIE

بیت فعال ساز وقفه که با یک کردن این بیت وقفه پورت فعال می شود و هرگاه بیت های SPIF یا MODF یک شوند باعث بروز وقفه می شود.

BITS

تعداد بیتهای ارسالی را مشخص می کند :

1000 8 bits per transfer
1001 9 bits per transfer
1010 10 bits per transfer
1011 11 bits per transfer
1100 12 bits per transfer
1101 13 bits per transfer
1110 14 bits per transfer
1111 15 bits per transfer
0000 16 bits per transfer

SPI Data Register (S0SPDR - 0x4002 0008)

دیتایی که باید ارسال شود در این رجیستر قرار می گیرد همچنین دیتای دریافتی هم در این رجیستر قرار می گیرد

31:16	15:8	7:0
RESERVED	DataHigh	DataLow

نحوه برنامه نویسی ماژول SPI در CMSIS

اول از همه طبق معمول باید هدر فایل مربوطه رو اول فایلتون اضافه کنید.

```
#include "lpc17xx_spi.h"
```

دو ساختار داده مرتبط با این ماژول در CMSIS وجود دارد:

SPI_CFG_Type

ساختار پیکر بندی ماژول است و حاوی عناصر زیر است:

uint32_t Databit

باید SPI_DATABIT_x باشد که X تعداد بیت مورد نظر برای هر انتقال است

uint32_t CPHA

SPI_CPHA_FIRST: first clock edge

SPI_CPHA_SECOND: second clock edge

uint32_t CPOL

SPI_CPOL_HI: high level

SPI_CPOL_LO: low level

uint32_t Mode

SPI_MASTER_MODE: Master mode

SPI_SLAVE_MODE: Slave mode

uint32_t DataOrder

SPI_DATA_MSB_FIRST: MSB first

SPI_DATA_LSB_FIRST: LSB first

uint32_t ClockRate

سرعت انتقال مورد نظر است

SPI_DATA_SETUP_Type

این ساختار دیتای ارسالی یا دریافتی را پیکر بندی می کند .

void * tx_data

اشاره گر به بافری است که دیتای ارسالی در آن قرار دارد

void * rx_data

اشاره گر به بافری است که دیتای دریافتی باید در آن قرار گیرد

uint32_t length

طول دیتای ارسالی یا دریافتی

uint32_t counter

شمارشگر دیتای ارسالی یا دریافتی

uint32_t status

وضیت فعلی فعالیت ماژول SPI

SPI_TRANSFER_Type

یک ENUM است که نحوه ارسال را تعیین می کند.

```
typedef enum {
               SPI_TRANSFER_POLLING = 0,
               SPI_TRANSFER_INTERRUPT
            } SPI_TRANSFER_Type;
```

توابع مورد استفاده :

void SPI_Init (LPC_SPI_TypeDef * SPIx , SPI_CFG_Type * SPI_ConfigStruct)

ماژول SPI را با استفاده از پارامترهای ست شده در SPI_ConfigStruct پیکربندی و فعال می کند

پارامترها :

SPIx : باید LPC_SPI باشد

SPI_ConfigStruct : اشاره گری به ساختار از نوع SPI_CFG_Type

Void SPI_DeInit (LPC_SPI_TypeDef *SPIx)

ماژول SPI را غیر فعال نموده و رجیستر ها را به صورت دیفالت مقدار دهی می کند.

Void SPI_SetClock (LPC_SPI_TypeDef *SPIx, uint32_t target_clock)

کلاک ریت ماژول را تنظیم می کند

پارامترها:

SPIx که باید LPC_SPI باشد

target_clock کلاک مورد نظر بر حسب هرتز(HZ)

*Void SPI_ConfigStructInit (SPI_CFG_Type *SPI_InitStruct)*

ساختار SPI_InitStruct را که به عنوان پارامتر به این تابع ارسال شده است را با مقادیر پیش فرض پر می کند که شامل :

- CPHA = SPI_CPHA_FIRST
- CPOL = SPI_CPOL_HI
- ClockRate = 1000000
- DataOrder = SPI_DATA_MSB_FIRST
- Databit = SPI_DATABIT_8
- Mode = SPI_MASTER_MODE

*Void SPI_SendData (LPC_SPI_TypeDef *SPIx, uint16_t Data)*

یک واحد دیتا را از طریق ماژول SPI ارسال می کند

SPIx که باید LPC_SPI باشد

Data باید طول ۸ تا ۱۶ بیت داشته باشد با توجه به آنچه قبلا تنظیم شده است.

*uint16_t SPI_ReceiveData (LPC_SPI_TypeDef *SPIx)*

یک واحد دیتا را از طریق ماژول SPI دریافت می کند

SPIx که باید LPC_SPI باشد

مقدار بازگشتی این تابع دیتای دریافتی است که ۱۶ بیت طول دارد

*int32_t SPI_ReadWrite (LPC_SPI_TypeDef *SPIx, SPI_DATA_SETUP_Type *dataCfg, SPI_TRANSFER_Type xfType)*

از این تابع برای ارسال یا دریافت چندین بایت از طریق پورت SPI استفاده می شود.

SPIx که باید LPC_SPI باشد

DataCfg : اشاره گری به یک ساختار از نوع SPI_DATA_SETUP_Type می باشد که اطلاعات مورد نیاز در مورد کانفیگ و دیتای انتقالی را در خود دارد.

XfType : نوع انتقال را معین می کند

- SPI_TRANSFER_POLLING: Polling mode
- SPI_TRANSFER_INTERRUPT: Interrupt mode

مقدار بازگشتی در مود POOLING تعداد بایت ترانسفر شده است و در مود INTERRUPT همیشه 0 بر می گرداند یا در صورت بروز اشکال 1-

توجه شود که این تابع در هر دو مود MASTER یا SLAVE قابل استفاده است.

*void SPI_IntCmd (LPC_SPI_TypeDef *SPIx, FunctionalState NewState)*

این تابع برای فعال یا غیر فعال کردن وقفه پورت SPI بکار گرفته می شود

SPIx که باید LPC_SPI باشد

NewState که باید ENALBE یا DISALBE باشد

*IntStatus SPI_GetIntStatus (LPC_SPI_TypeDef *SPIx)*

برای چک کردن وضعیت فلگ اینتراپت بکار می رود

SPIx که باید LPC_SPI باشد

مقدار بازگشتی با توجه به وضعیت ماژول SET or RESET می باشد.

*void SPI_ClearIntPending (LPC_SPI_TypeDef *SPIx)*

اینتراپت فلگ مربوط به ماژول SPI را پاک می کند

SPIx که باید LPC_SPI باشد

*uint8_t SPI_GetDataSize (LPC_SPI_TypeDef *SPIx)*

برای هر انتقال تعداد بیتهای ارسالی را بر می گرداند که می تواند ۸ تا ۱۶ بیت باشد

Int32_t SPI_GetStatus (LPC_SPI_TypeDef *SPIx)

مقدار فعلیSPI Status register را بر می گرداندکه این مقدار بازگشتی باید بوسیله تابع SPI_CheckStatus چک شود تا وضعیت فلگهای مختلف مشخص شود.

FlagStatus SPI_CheckStatus (uint32_t inputSPIStatus ,uint8_t SPIStatus)

چک می کند که آیا بیت فلگ مورد نظر در SPI STATUS REGISTER فعال است یا نه

inputSPIStatus : مقدار بازگشتی تابع SPI_GetStatus به عنوان ورودی به این تابع ارسال می شود

SPIStatus : باید یکی از مقادیر زیر باشد برای چک کردن فلگ معادل

- SPI_STAT_ABRT: Slave abort.
- SPI_STAT_MODF: Mode fault.
- SPI_STAT_ROVR: Read overrun.
- SPI_STAT_WCOL: Write collision.
- SPI_STAT_SPIF: SPI transfer complete.

مقدار بازگشتی این تابع SET or RESET است که بیانگر ۰ یا ۱ بودن فلگ است.

تهیه و تنظیم:

محسن خاشعی

اسفند ماه ۱۳۹۰