CP316
Serial Communication-SPI

Terry Sturtevant

Wilfrid Laurier University

February 13, 2018

Serial Communication -SPI

Serial Communication -SPI

- Serial Peripheral Interface

Serial Communication -SPI

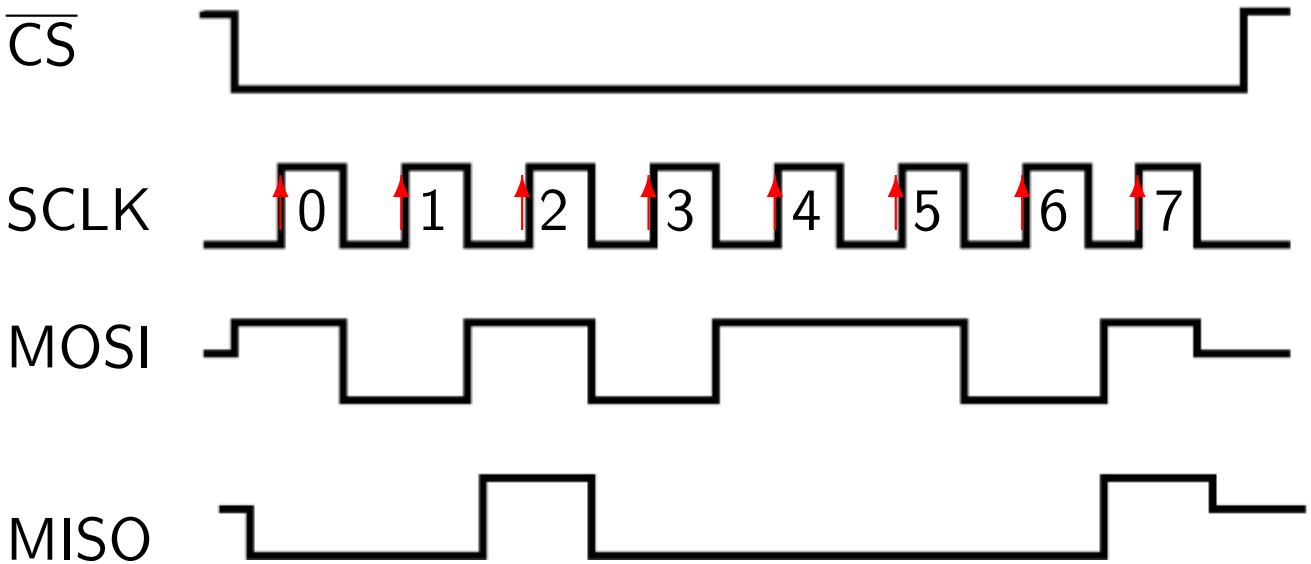
- Serial Peripheral Interface
- Master/slave communication

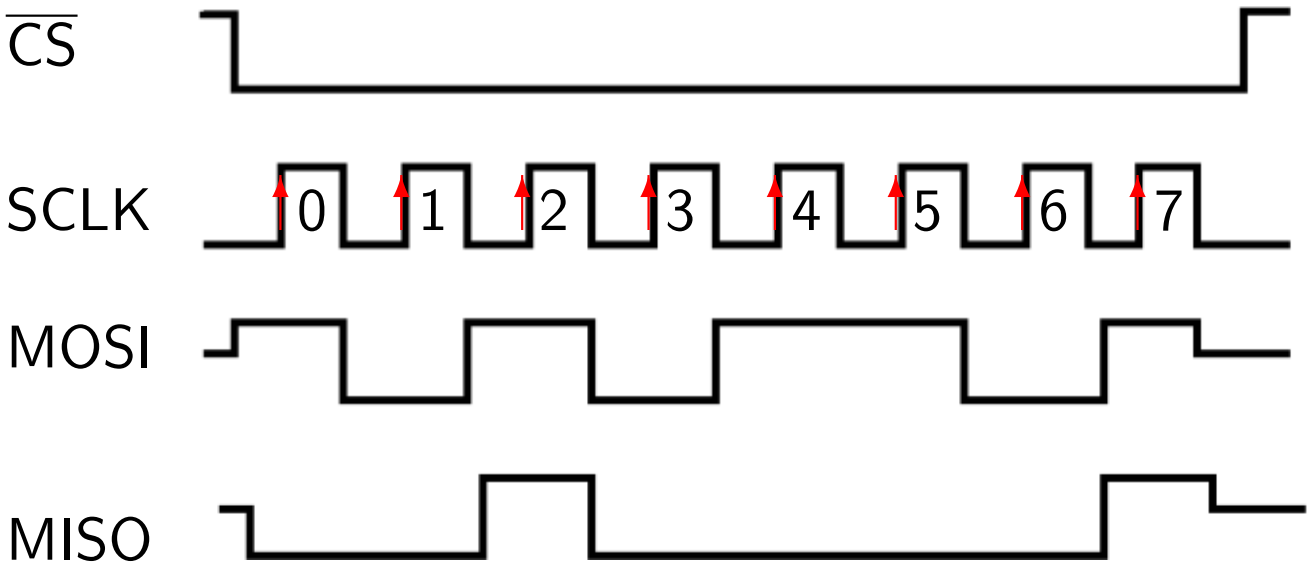
Serial Communication -SPI

- Serial Peripheral Interface
- Master/slave communication
- Uses 3 signals (and Ground),
  MISO, MOSI, SCLK
  and chip selects for each slave device

Serial Communication -SPI

- Serial Peripheral Interface
- Master/slave communication
- Uses 3 signals (and Ground),
  MISO, MOSI, SCLK
  and chip selects for each slave device
- Synchronous, so master controls clock rate


$\overline{CS}$
SCLK
0
1
2
3
4
5
6
7
MOSI
MISO



SPI transfers can happen in both directions simultaneously.

Introduction

Introduction

QwikFlash modules

Introduction

QwikFlash modules

ramifications???

Introduction

QwikFlash modules

ramifications???

interrupts; transmit and receive

Introduction

QwikFlash modules

ramifications???

interrupts; transmit and receive

$\rightarrow$ Sections 6.4.5 to 6.4.7

Introduction

QwikFlash modules

ramifications???

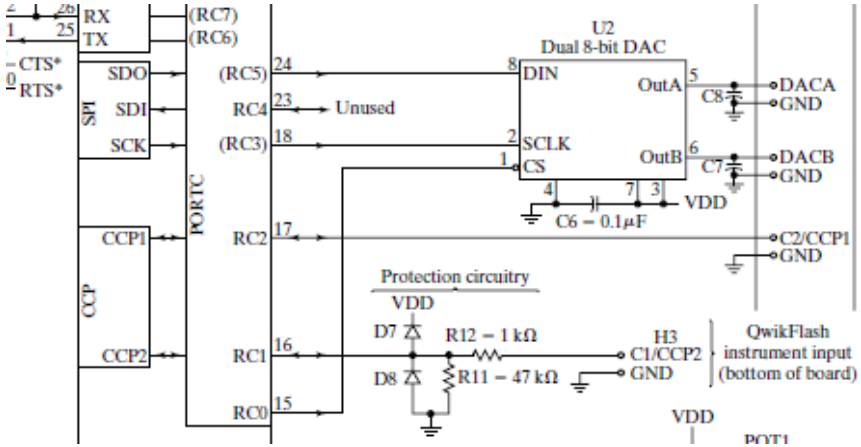
interrupts; transmit and receive

→ Sections 6.4.5 to 6.4.7

→ Section 8.2

QwikFlash SPI connections

QwikFlash SPI connections



Master Synchronous Serial Port (MSSP) module

Master Synchronous Serial Port (MSSP) module

2 modes; **SPI** and I^2C

Master Synchronous Serial Port (MSSP) module

2 modes; **SPI** and I^2C

→ Section 15.0 -15.2

SPI

SPI

3 wires +chip select, master-slave

SPI

3 wires +chip select, master-slave

overview

SPI

3 wires +chip select, master-slave

overview

→ Section 10.2

SPI

3 wires +chip select, master-slave

overview

→ Section 10.2

SPI registers

SPI

3 wires +chip select, master-slave

overview

→ Section 10.2

SPI registers

→ Section 10.3.1

SPI

3 wires +chip select, master-slave

overview

→ Section 10.2

SPI registers

→ Section 10.3.1

operation

SPI

3 wires +chip select, master-slave

overview

→ Section 10.2

SPI registers

→ Section 10.3.1

operation

→ Sections 10.3.2 to 10.3.5

SPI

3 wires +chip select, master-slave

overview

→ Section 10.2

SPI registers

→ Section 10.3.1

operation

→ Sections 10.3.2 to 10.3.5

→ Section 15.3

SPI

3 wires +chip select, master-slave

overview

→ Section 10.2

SPI registers

→ Section 10.3.1

operation

→ Sections 10.3.2 to 10.3.5

→ Section 15.3

QwikFlash MAX522 DAC

SPI summary

SPI summary

3 wires (+ ground), one-to-many

SPI summary

3 wires (+ ground), one-to-many

SCLK (from master)

SPI summary

3 wires (+ ground), one-to-many

SCLK (from master)

SDO (serial data out)

SPI summary

3 wires (+ ground), one-to-many

SCLK (from master)

SDO (serial data out)

SDI (serial data in)

SPI summary

3 wires (+ ground), one-to-many

SCLK (from master)

SDO (serial data out)

SDI (serial data in)

$\overline{CS}$ for each device, generated by master

SPI summary

3 wires (+ ground), one-to-many

SCLK (from master)

SDO (serial data out)

SDI (serial data in)

$\overline{CS}$ for each device, generated by master

data transmission rate set by SCLK

SPI summary

3 wires (+ ground), one-to-many

SCLK (from master)

SDO (serial data out)

SDI (serial data in)

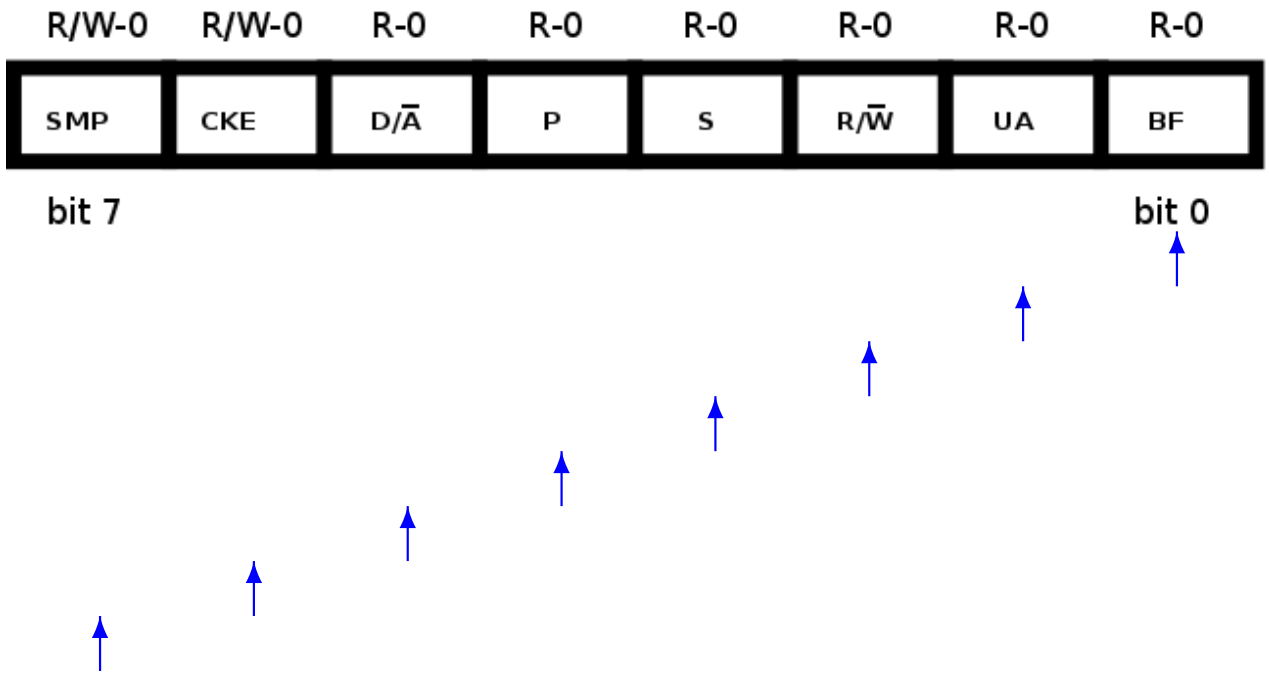
$\overline{CS}$ for each device, generated by master

data transmission rate set by SCLK

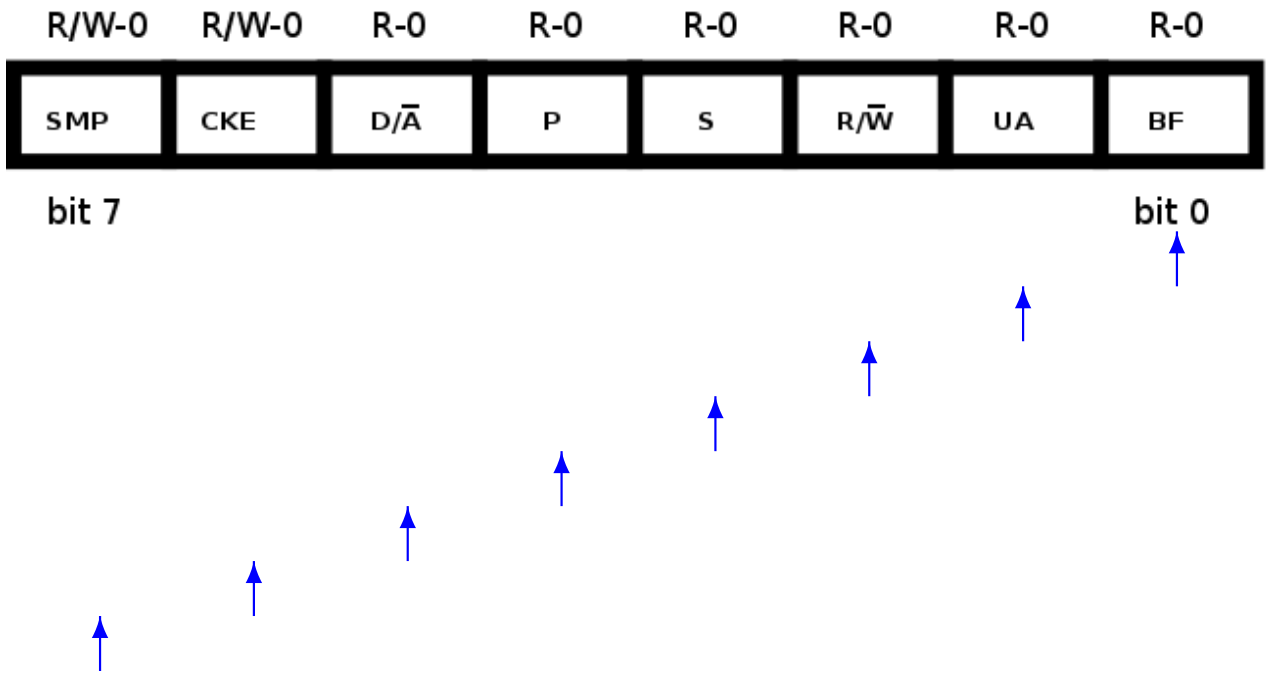
packets are single characters

SSPSTAT

SSPSTAT



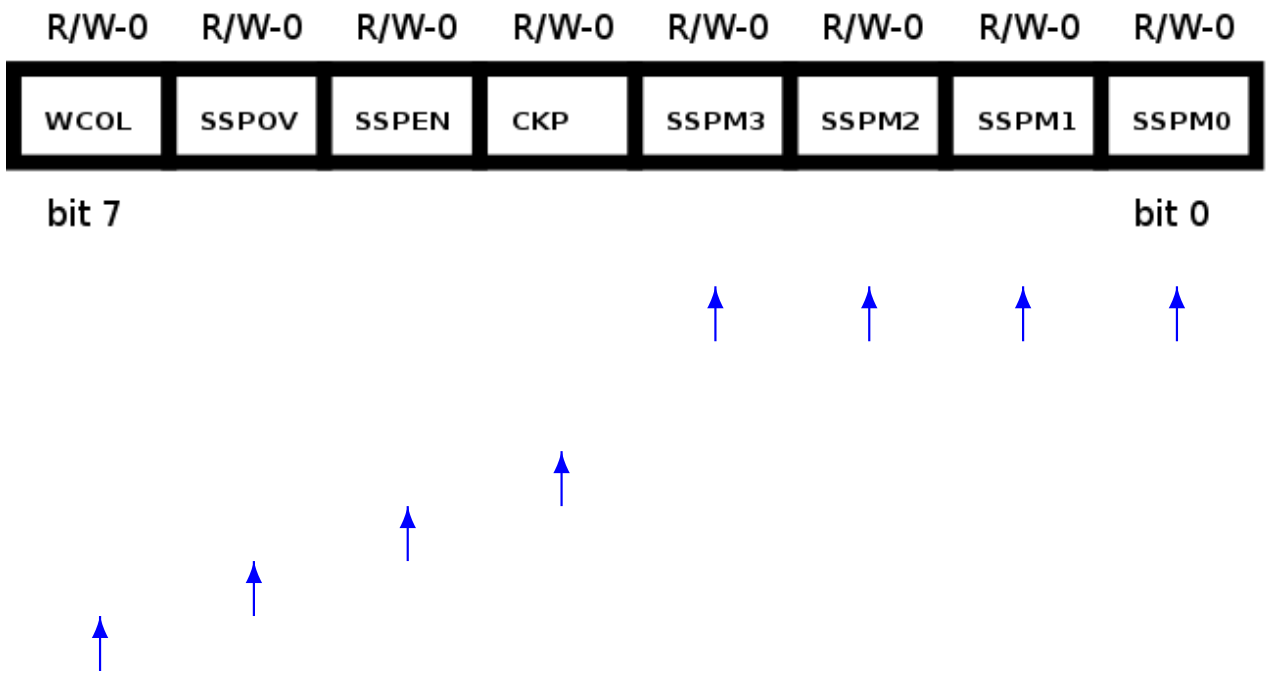
SSPSTAT



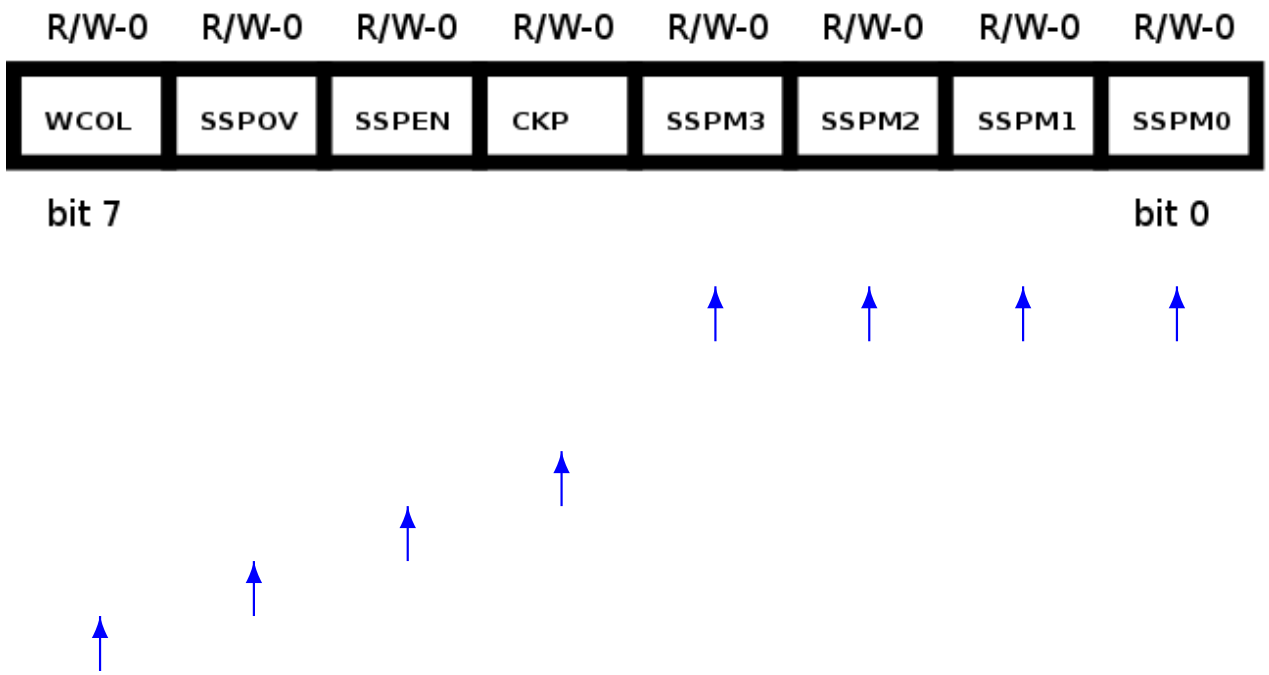
Bits in SSPSTAT register

SSPCON1

SSPCON1


R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
WCOL
SSPOV
SSPEN
CKP
SSPM3
SSPM2
SSPM1
SSPM0
bit 7
bit 0

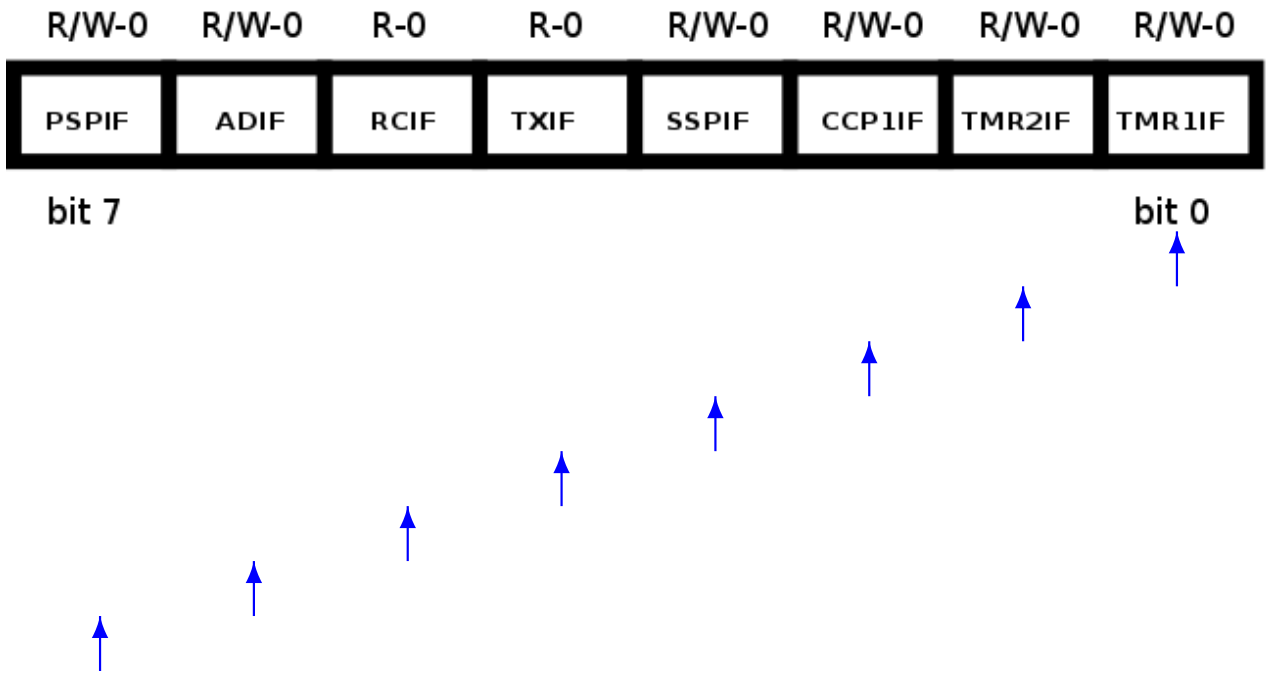
SSPCON1



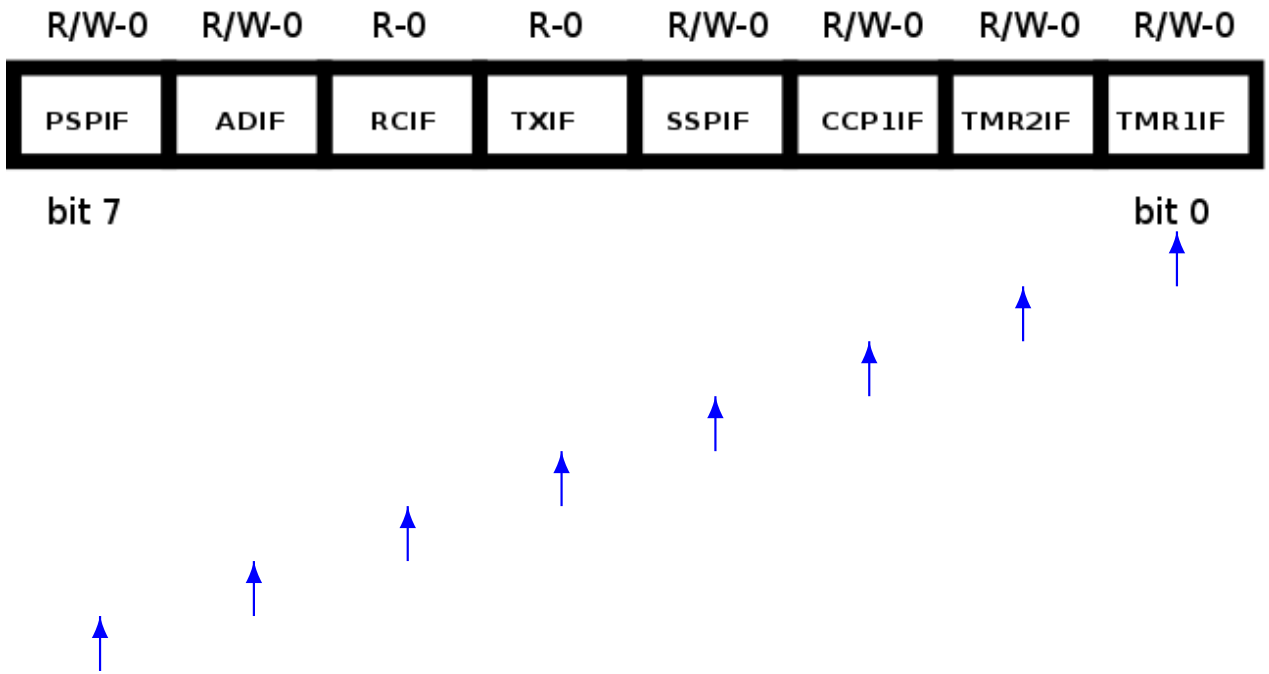
Bits in SSPCON1 register

PIR1

PIR1

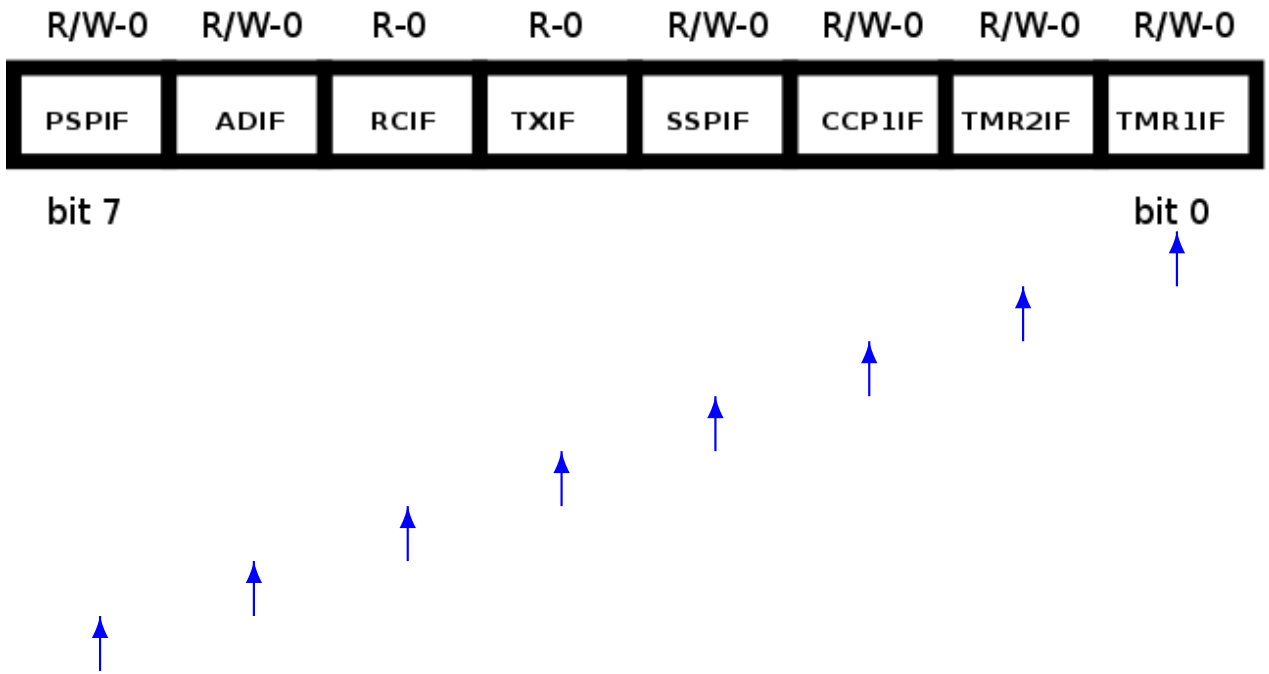

R/W-0
R/W-0
R-0
R-0
R/W-0
R/W-0
R/W-0
R/W-0
PSPIF
ADIF
RCIF
TXIF
SSPIF
CCP1IF
TMR2IF
TMR1IF
bit 7
bit 0

PIR1



Bits in PIR1 register

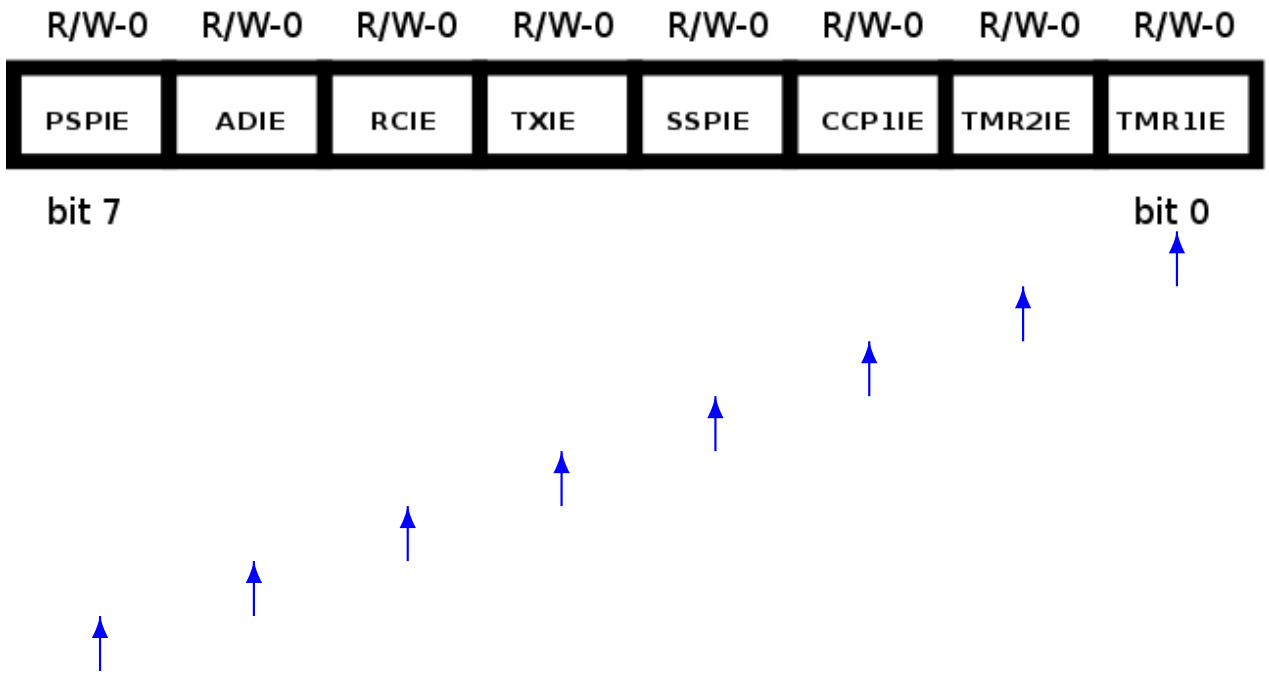
PIR1



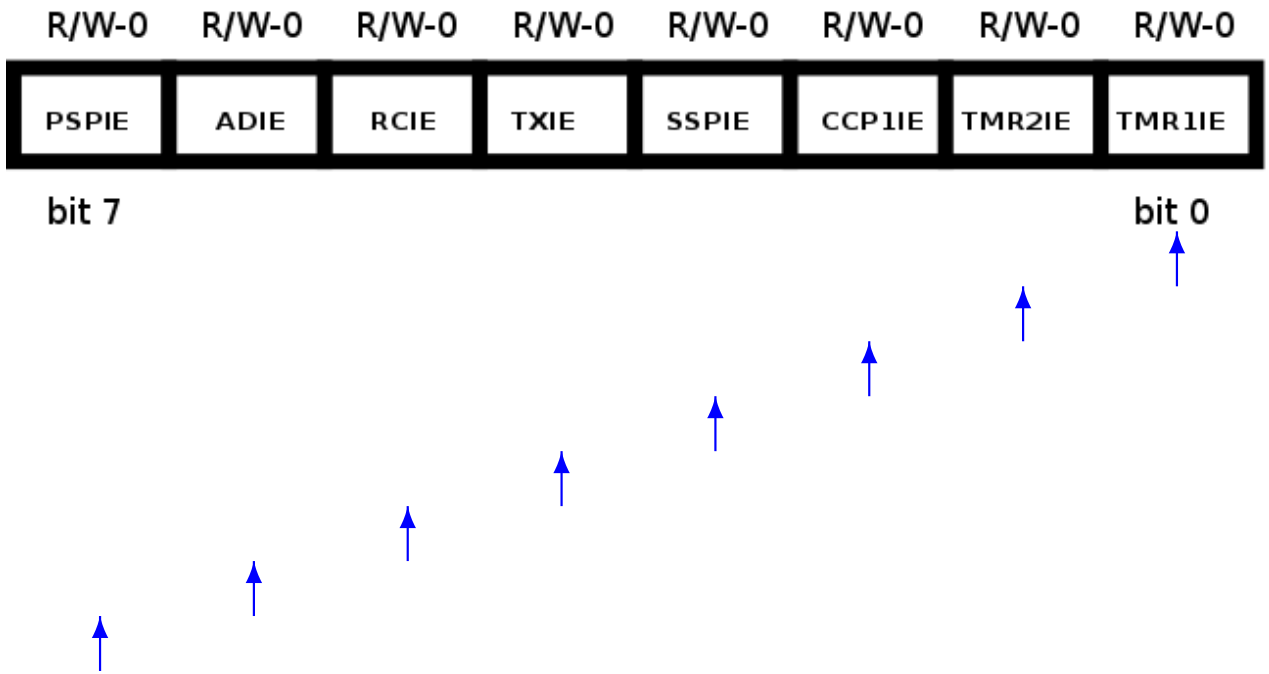
Bits in PIR1 register - Note SSPIF

PIE1

PIE1

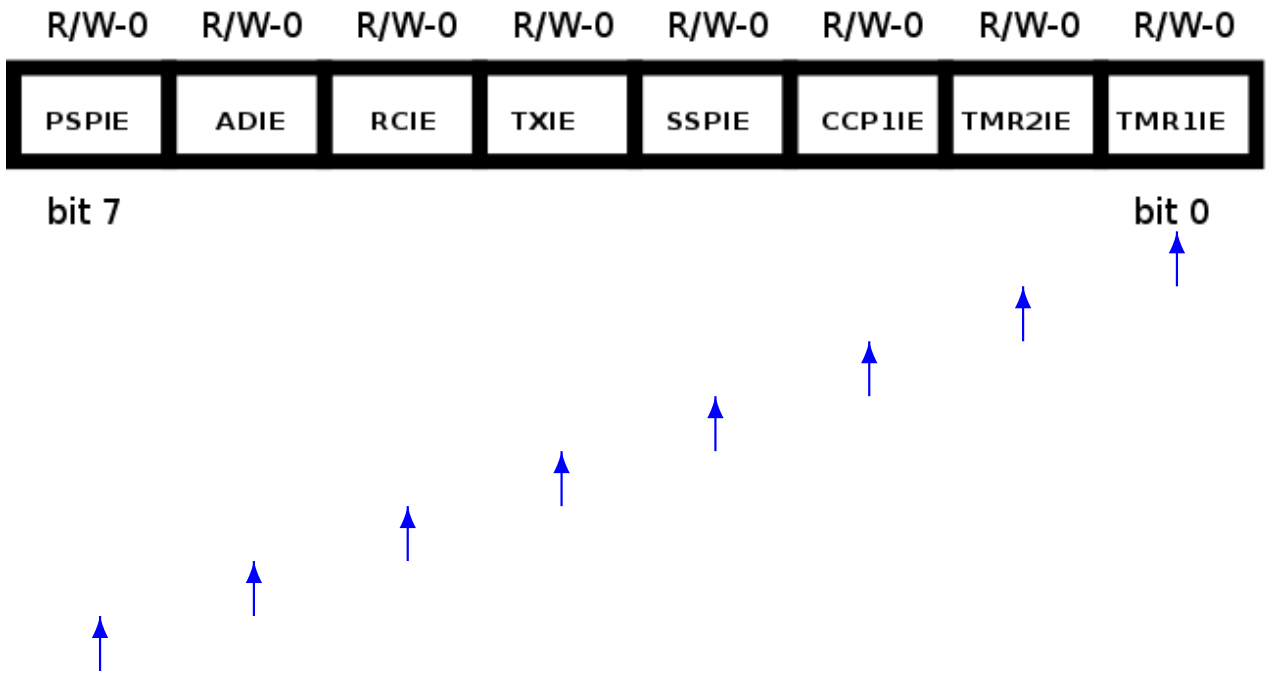

R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
PSPIE
ADIE
RCIE
TXIE
SSPIE
CCP1IE
TMR2IE
TMR1IE
bit 7
bit 0

PIE1



Bits in PIE1 register

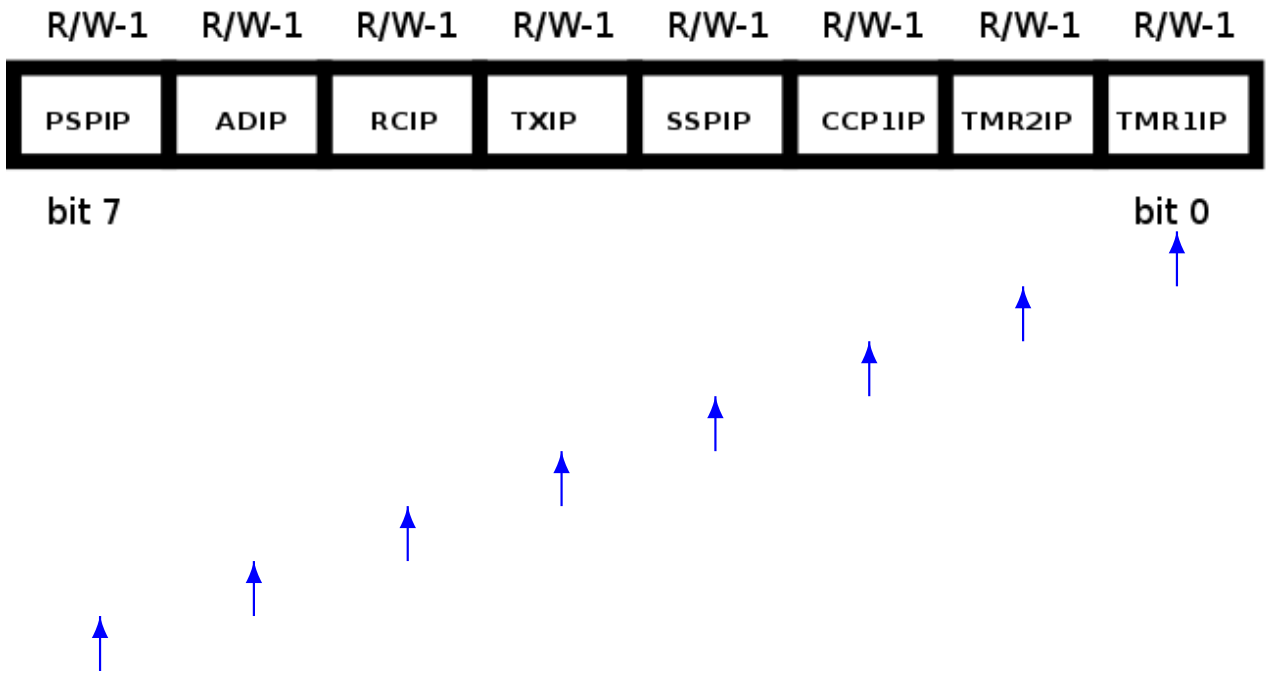
PIE1



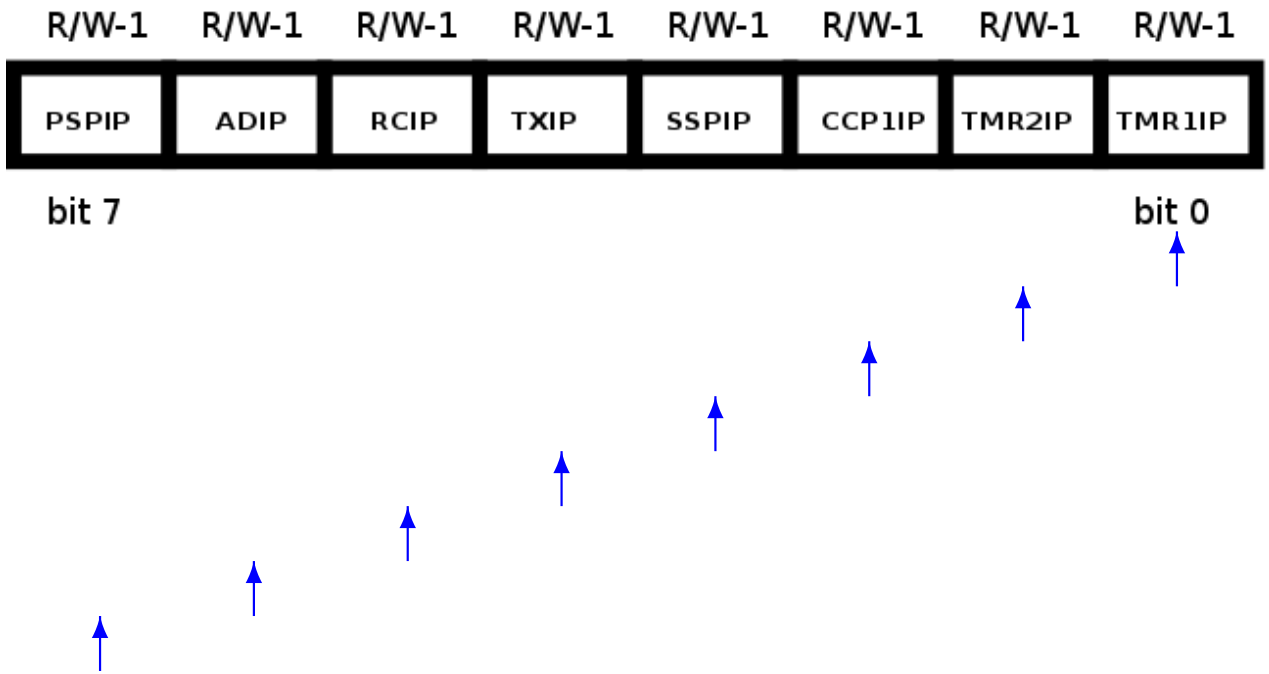
Bits in PIE1 register - Note SSPIE

IPR1

IPR1

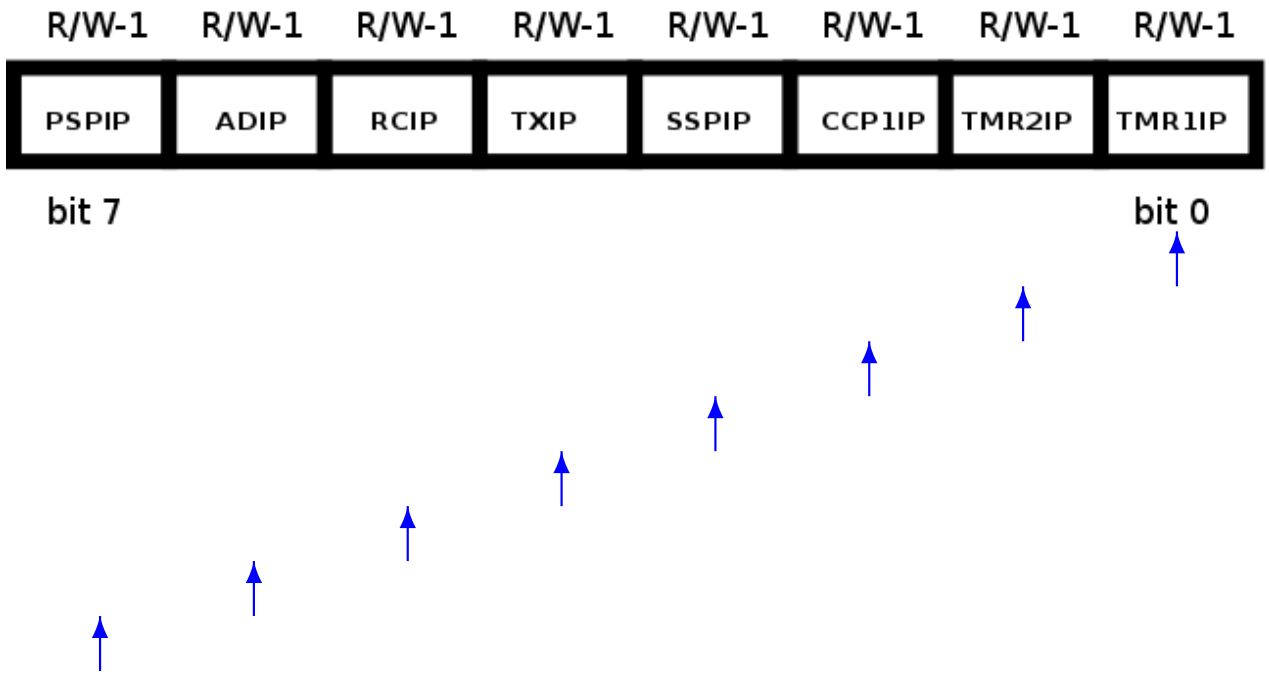

R/W-1
R/W-1
R/W-1
R/W-1
R/W-1
R/W-1
R/W-1
R/W-1
PSPIP
ADIP
RCIP
TXIP
SSPIP
CCP1IP
TMR2IP
TMR1IP
bit 7
bit 0

IPR1



Bits in IPR1 register

IPR1



Bits in IPR1 register - Note SSPIP

Bit-bashing

Bit-bashing

overview

Bit-bashing

overview

reasons

Bit-bashing

overview

reasons

NIB

Code

Code

PORT configuration

Code

PORT configuration

$\rightarrow$ macro or subroutine?

Code

PORT configuration

$\rightarrow$ macro or subroutine?

Initialization

Code

PORT configuration

$\rightarrow$ macro or subroutine?

Initialization

$\rightarrow$ macro or subroutine?

Code

PORT configuration

$\rightarrow$ macro or subroutine?

Initialization

$\rightarrow$ macro or subroutine?

Write to device

Code

PORT configuration

→ macro or subroutine?

Initialization

→ macro or subroutine?

Write to device

→ macro or subroutine?

Code

PORT configuration

$\rightarrow$ macro or subroutine?

Initialization

$\rightarrow$ macro or subroutine?

Write to device

$\rightarrow$ macro or subroutine?

Read from device

Code

PORT configuration

→ macro or subroutine?

Initialization

→ macro or subroutine?

Write to device

→ macro or subroutine?

Read from device

→ macro or subroutine?

Initialization

Initialization

1. configure port (**TRISA**)

Initialization

1. configure port (**TRISA**)

   don't inadvertently alter other bits

Initialization

1. configure port (**TRISA**)

   don't inadvertently alter other bits
2. set MSSP mode and other parameters (**SSPCON1**)

Initialization

1. configure port (**TRISA**)

   don't inadvertently alter other bits
2. set MSSP mode and other parameters (**SSPCON1**)

   SPI, master, enable, etc.

Initialization

1. configure port (**TRISA**)

   don't inadvertently alter other bits
2. set MSSP mode and other parameters (**SSPCON1**)

   SPI, master, enable, etc.
3. set other parameters (**SSPSTAT**)

Initialization

1. configure port (**TRISA**)

   don't inadvertently alter other bits
2. set MSSP mode and other parameters (**SSPCON1**)

   SPI, master, enable, etc.
3. set other parameters (**SSPSTAT**)

   SMP, CKE

Initialization

1. configure port (**TRISA**)

   don't inadvertently alter other bits
2. set MSSP mode and other parameters (**SSPCON1**)

   SPI, master, enable, etc.
3. set other parameters (**SSPSTAT**)

   SMP, CKE
4. configure interrupts (if desired) (**IPR1**)

Initialization

1. configure port (**TRISA**)

   don't inadvertently alter other bits
2. set MSSP mode and other parameters (**SSPCON1**)

   SPI, master, enable, etc.
3. set other parameters (**SSPSTAT**)

   SMP, CKE
4. configure interrupts (if desired) (**IPR1**)

   assuming interrupts are enabled globally

Sending and Receiving

Sending and Receiving

Sending and receiving happen simultaneously.

Sending and Receiving

Sending and receiving happen simultaneously.

1. check flag to see that buffer is empty (**SSPSTAT**)

Sending and Receiving

Sending and receiving happen simultaneously.

1. check flag to see that buffer is empty (**SSPSTAT**)

   otherwise there is a character *being* transmitted already

Sending and Receiving

Sending and receiving happen simultaneously.

1. check flag to see that buffer is empty (**SSPSTAT**)

   otherwise there is a character *being* transmitted already
2. assert $\overline{CS}$ (**LATC**)

Sending and Receiving

Sending and receiving happen simultaneously.

1. check flag to see that buffer is empty (**SSPSTAT**)

   otherwise there is a character *being* transmitted already
2. assert $\overline{CS}$ (**LATC**)
3. place value in buffer (**SSPBUF**)

Sending and Receiving

Sending and receiving happen simultaneously.

1. check flag to see that buffer is empty (**SSPSTAT**)

   otherwise there is a character *being* transmitted already
2. assert $\overline{CS}$ (**LATC**)
3. place value in buffer (**SSPBUF**)
4. enable interrupt (if using) (**PIE1**)

Sending and Receiving

Sending and receiving happen simultaneously.

1. check flag to see that buffer is empty (**SSPSTAT**)

   otherwise there is a character *being* transmitted already
2. assert $\overline{CS}$ (**LATC**)
3. place value in buffer (**SSPBUF**)
4. enable interrupt (if using) (**PIE1**)

   assuming interrupts are enabled globally

Sending and Receiving

Sending and receiving happen simultaneously.

1. check flag to see that buffer is empty (**SSPSTAT**)

   otherwise there is a character *being* transmitted already
2. assert $\overline{CS}$ (**LATC**)
3. place value in buffer (**SSPBUF**)
4. enable interrupt (if using) (**PIE1**)

   assuming interrupts are enabled globally
5. wait for flag to see that buffer is empty (**SSPSTAT**)

Sending and Receiving

Sending and receiving happen simultaneously.

1. check flag to see that buffer is empty (**SSPSTAT**)

   otherwise there is a character *being* transmitted already
2. assert $\overline{CS}$ (**LATC**)
3. place value in buffer (**SSPBUF**)
4. enable interrupt (if using) (**PIE1**)

   assuming interrupts are enabled globally
5. wait for flag to see that buffer is empty (**SSPSTAT**)

   (if not using interrupts)

Sending and Receiving

Sending and receiving happen simultaneously.

1. check flag to see that buffer is empty (**SSPSTAT**)

   otherwise there is a character *being* transmitted already
2. assert $\overline{CS}$ (**LATC**)
3. place value in buffer (**SSPBUF**)
4. enable interrupt (if using) (**PIE1**)

   assuming interrupts are enabled globally
5. wait for flag to see that buffer is empty (**SSPSTAT**)

   (if not using interrupts)
6. de-assert $\overline{CS}$ (**LATC**)

Sending and Receiving

Sending and receiving happen simultaneously.

1. check flag to see that buffer is empty (**SSPSTAT**)

   otherwise there is a character *being* transmitted already
2. assert $\overline{CS}$ (**LATC**)
3. place value in buffer (**SSPBUF**)
4. enable interrupt (if using) (**PIE1**)

   assuming interrupts are enabled globally
5. wait for flag to see that buffer is empty (**SSPSTAT**)

   (if not using interrupts)
6. de-assert $\overline{CS}$ (**LATC**)
7. when no more characters to send,

Sending and Receiving

Sending and receiving happen simultaneously.

1. check flag to see that buffer is empty (**SSPSTAT**)
   otherwise there is a character *being* transmitted already
2. assert $\overline{CS}$ (**LATC**)
3. place value in buffer (**SSPBUF**)
4. enable interrupt (if using) (**PIE1**)
   assuming interrupts are enabled globally
5. wait for flag to see that buffer is empty (**SSPSTAT**)
   (if not using interrupts)
6. de-assert $\overline{CS}$ (**LATC**)
7. when no more characters to send,
   disable transmit interrupt (if using) (**PIE1**)