EMBEDDED SYSTEMS CMPE-453

Department of Computer Engineering

Serial Communication-4

SPI (SERIAL PERIPHERAL INTERFACE)

- Very fast data transfer between 2 or more devices
  - SD card reader modules
  - RFID card reader modules
  - 2.4 GHz wireless transmitter/receivers
  - Communication with flash memory, sensors, real-time clock (RTC), analog-to-digital converters
- a clocked or synchronous protocol.
- Speed, starting from 1 megahertz
- 10 megahertz is pretty much standard. (a thousand times faster than UART)
- Up to 50–100 MHz
- Lines
  - clock line
  - two data lines (one for each direction)
  - a fourth line that's used to signal the start and end of a transaction

A TRADE OFF!

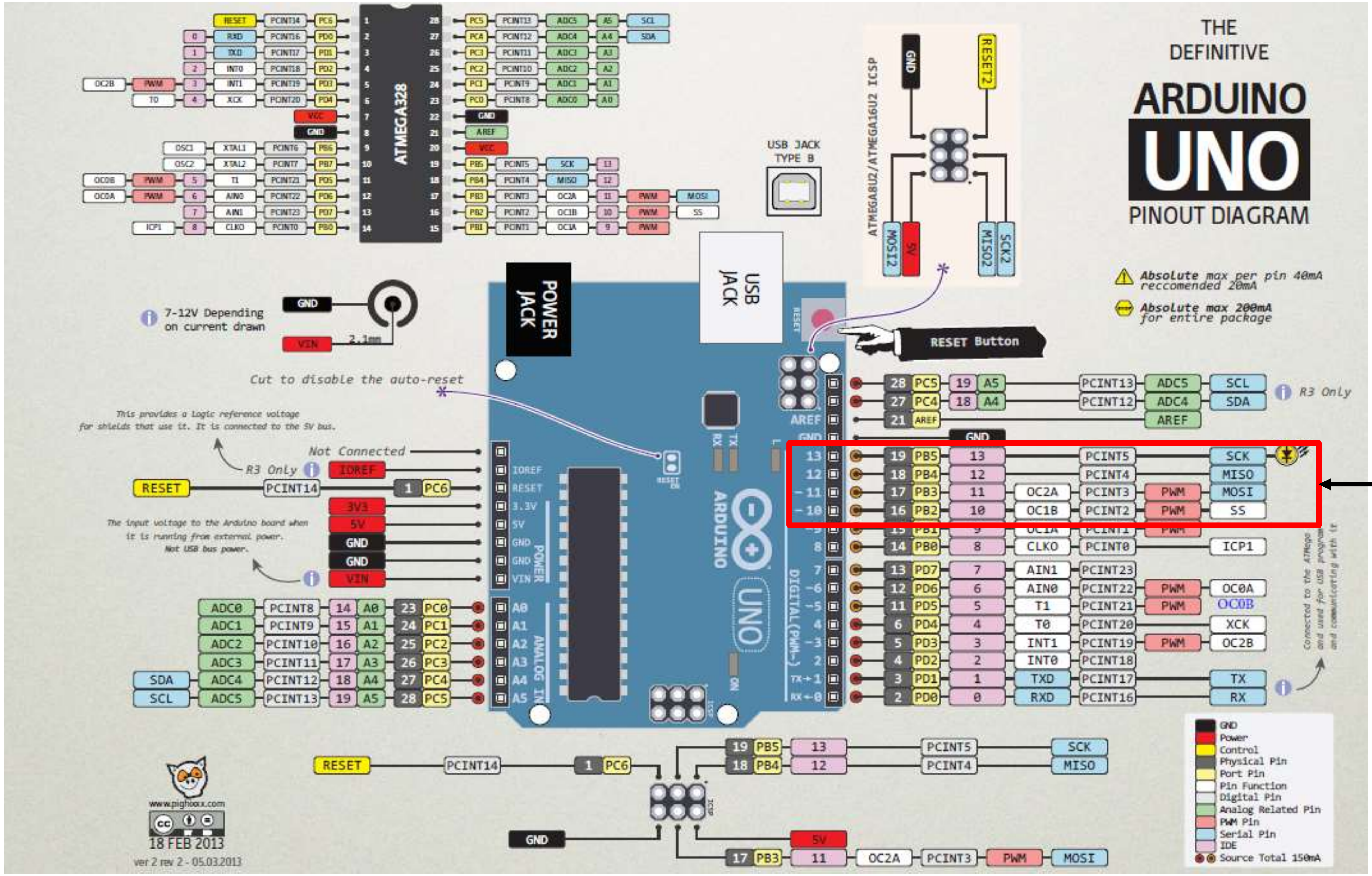
THE PINS

- **MISO** – Master In Slave Out.
  - Input pin for Master AVR
  - Output pin for Slave AVR
  - Data transfer from Slave to Master

- **MOSI** – Master Out Slave In.
  - Output pin for Master
  - Input pin for Slave
  - Data transfer from Master to Slave

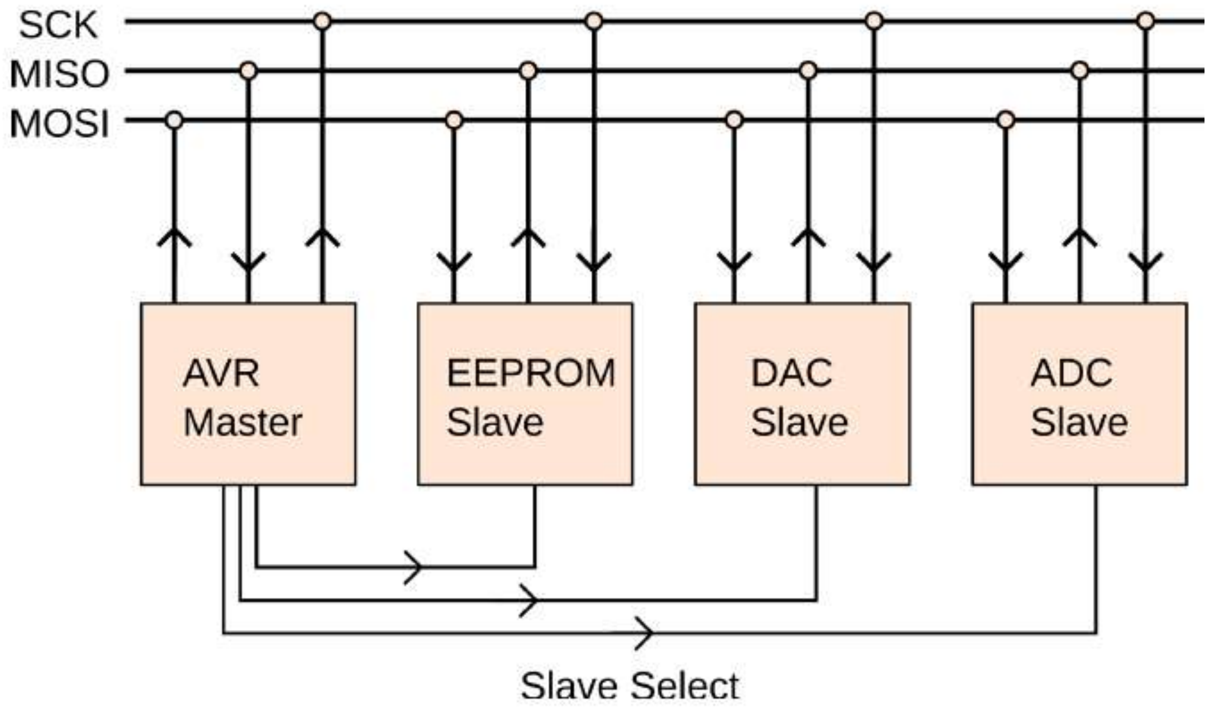
- **SCK** – This is the SPI clock line (since SPI is a synchronous communication).

- **SS** – Slave Select


THE DEFINITIVE
ARDUINO
UNO
PINOUT DIAGRAM
ATMEGA328
POWER JACK
USB JACK
USB JACK TYPE B
ATMEGA8U2/ATMEGA16U2 ICSP
RESET Button
7-12V Depending on current drawn
Cut to disable the auto-reset
This provides a logic reference voltage for shields that use it. It is connected to the 5V bus.
Not Connected
R3 Only
The input voltage to the Arduino board when it is running from external power. Not USB bus power.
Absolute max per pin 40mA reccomended 20mA
Absolute max 200mA for entire package
13 PB5 PCINT5 SCK
12 PB4 PCINT4 MISO
11 PB3 OC2A PCINT3 PWM MOSI
10 PB2 OC1B PCINT2 PWM SS
www.pighixxx.com
18 FEB 2013
ver 2 rev 2 - 05.03.2013

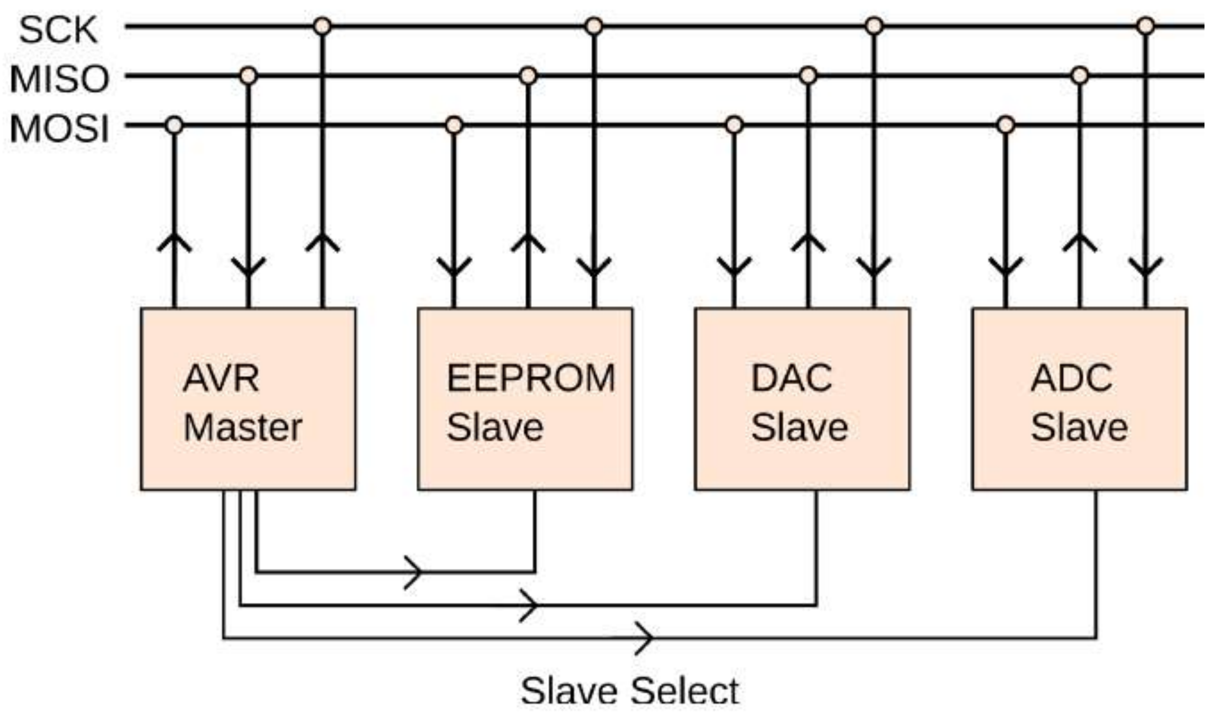
SPI Pins

SPI BUS



- A Master device is responsible for generating the clock signal (SCK) that all of the *slave* devices receive
- Both master and slave devices use the clock signal to synchronize their voltage signals on the MOSI and MISO
- The master sets the voltage on MOSI, the slave sets the voltage level on MISO
- the slave-select lines control which slave device is currently active.
- One slave-select line per slave device.
- On an SPI bus, all the chips can be connected to the same SCK, MISO, and MOSI wires, but each target chip also has to have its own chip select line.
- keeps SPI basically limited to small networks, or forces to use trickery like port expanders to address all of the devices.

SPI BUS



- One edge of the timing clock: bit is set up for output
- Next edge of the timing clock: one bit is read as input.
- With every clock tick, both the master and the slave transmit and receive a bit of data on their respective lines.
- no concept of separate "send" and "receive" actions in SPI—both are sending and receiving all the time

SPI REGISTERS

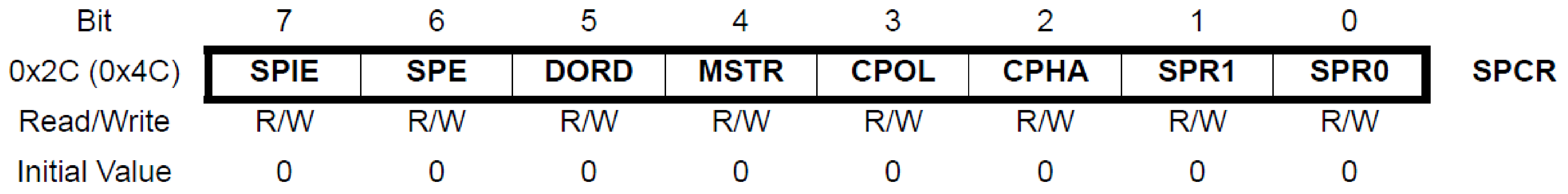
1. **SPCR – SPI Control Register** – This register is basically the master register i.e. it contains the bits to initialize SPI and control it.

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	**SPIE**	**SPE**	**DORD**	**MSTR**	**CPOL**	**CPHA**	**SPR1**	**SPR0**	**SPCR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7: SPIE – SPI Interrupt Enable
The SPI Interrupt Enable bit is used to enable interrupts in the SPI. Note that global interrupts must be enabled to use the interrupt functions. Set this bit to '1' to enable interrupts.

Bit 6: SPE – SPI Enable
The SPI Enable bit is used to enable SPI as a whole. When this bit is set to 1, the SPI is enabled or else it is disabled. When SPI is enabled, the normal I/O functions of the pins are overridden.

SPI REGISTERS

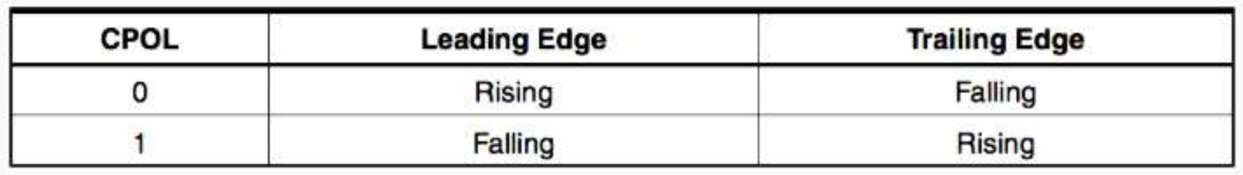


Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 5: DORD – Data Order
DORD stands for Data Order. Set this bit to 1 if you want to transmit LSB first, else set it to 0, in which case it sends out MSB first.

Bit 4: MSTR – Master/Slave Select
This bit is used to configure the device as Master or as Slave. When this bit is set to 1, the SPI is in Master mode (i.e. clock will be generated by the particular device), else when it is set to 0, the device is in SPI Slave mode.

Bit 3: CPOL – Clock Polarity
This bit selects the clock polarity when the bus is idle. Set this bit to 1 to ensure that SCK is HIGH when the bus is idle, otherwise set it to 0 so that SCK is LOW in case of idle bus.



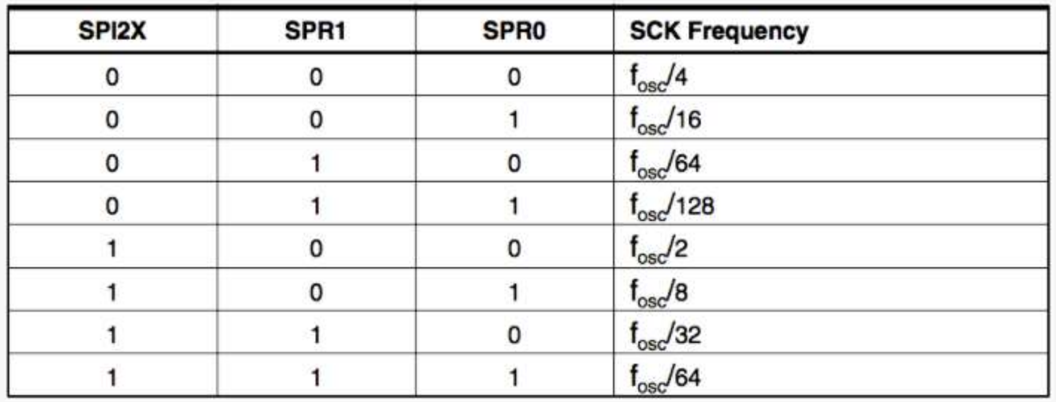
CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

CPOL Functionality

SPI REGISTERS

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 1,0: SPR1, SPR0 – SPI Clock Rate Select
These bits, along with the SPI2X bit in the SPSR register (discussed next), are used to choose the oscillator frequency divider, wherein the f_{OSC} stands for internal clock, or the frequency of the crystal in case of an external oscillator.

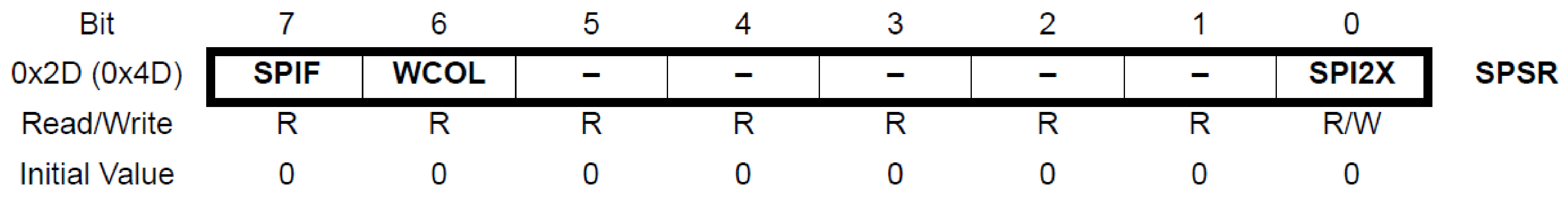


SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Frequency Divider

SPI REGISTERS

2. **SPSR -** The SPI Status Register (SPSR) : shows the status of the SPI bus, sets the interrupt flag



Bit	7	6	5	4	3	2	1	0	
0x2D (0x4D)	**SPIF**	**WCOL**	**–**	**–**	**–**	**–**	**–**	**SPI2X**	**SPSR**
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – SPIF: SPI Interrupt Flag
When a serial transfer is complete, the SPIF Flag is set.

Bit 0: SPI2x – SPI Double Speed Mode
The SPI double speed mode bit reduces the frequency divider from 4x to 2x, hence doubling the speed. Set this bit to 1 to enable SPI Double Speed Mode.

SPI REGISTERS

3. **SPDR -** The SPI Data Register: 8-bit read/write register. Reading the incoming data, and writing the data to transmit.

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	**MSB**							**LSB**	**SPDR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

RECALL !

bit 5 of SPCR – the DORD bit. When DORD is set to 1, then LSB, i.e. the 0th bit of the SPDR is transmitted first, and vice versa.

SPI DATA MODES

Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize

SPI Mode	Conditions	Leading Edge	Trailing eDge
0	CPOL=0, CPHA=0	Sample (rising)	Setup (falling)
1	CPOL=0, CPHA=1	Setup (rising)	Sample (falling)
2	CPOL=1, CPHA=0	Sample (falling)	Setup (rising)
3	CPOL=1, CPHA=1	Setup (falling)	Sample (rising)

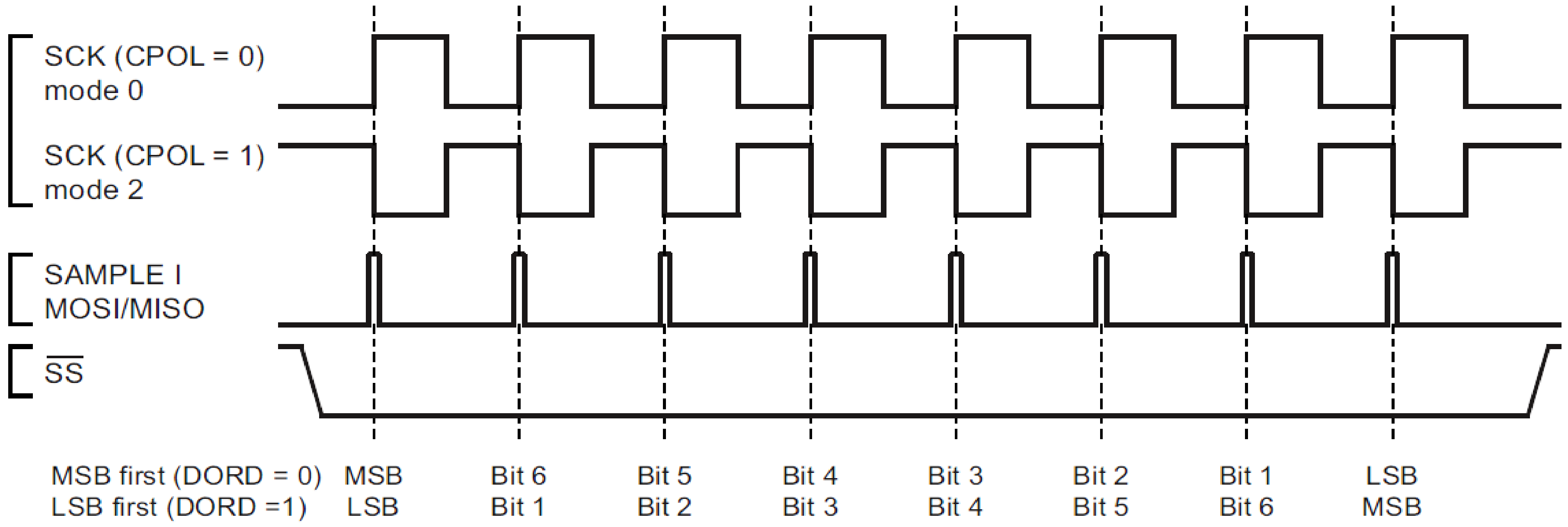
RECALL !

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	**SPIE**	**SPE**	**DORD**	**MSTR**	**CPOL**	**CPHA**	**SPR1**	**SPR0**	**SPCR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

SPI DATA MODES

SPI Mode	Conditions	Leading Edge	Trailing eDge
0	CPOL=0, CPHA=0	Sample (rising)	Setup (falling)
1	CPOL=0, CPHA=1	Setup (rising)	Sample (falling)
2	CPOL=1, CPHA=0	Sample (falling)	Setup (rising)
3	CPOL=1, CPHA=1	Setup (falling)	Sample (rising)

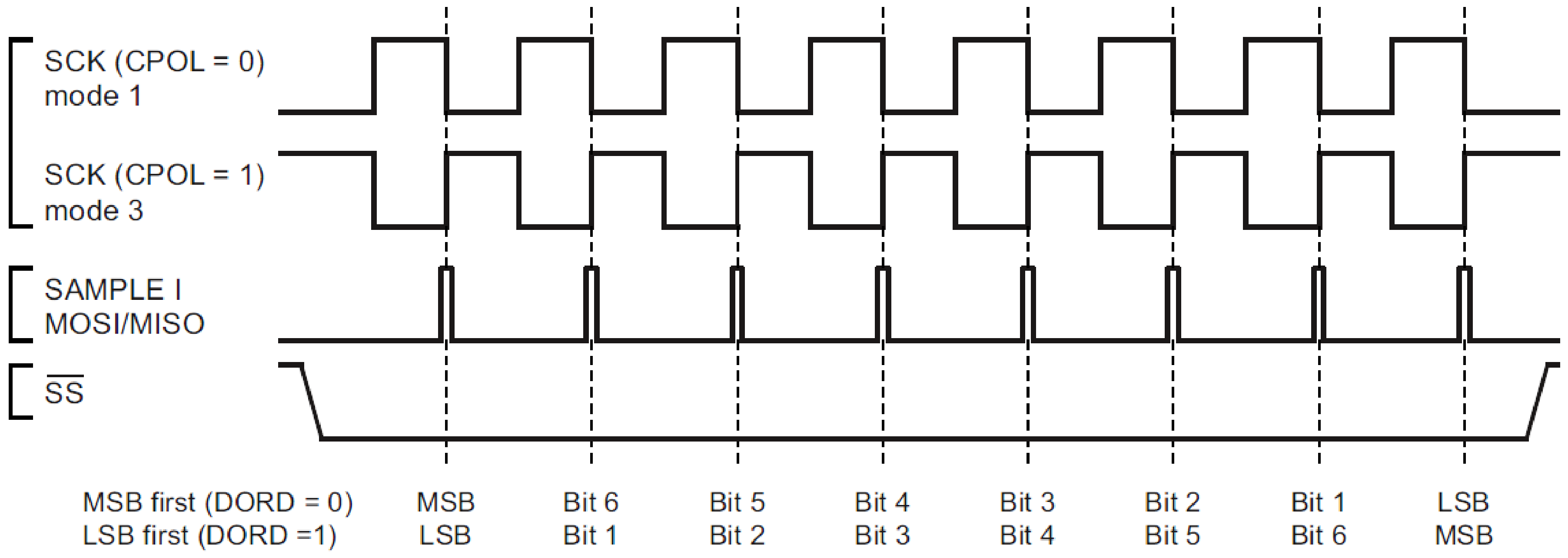
SPI Transfer Format with CPHA=0


SCK (CPOL = 0)
mode 0
SCK (CPOL = 1)
mode 2
SAMPLE I
MOSI/MISO
$\overline{\text{SS}}$
MSB first (DORD = 0) MSB Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 LSB
LSB first (DORD =1) LSB Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 MSB

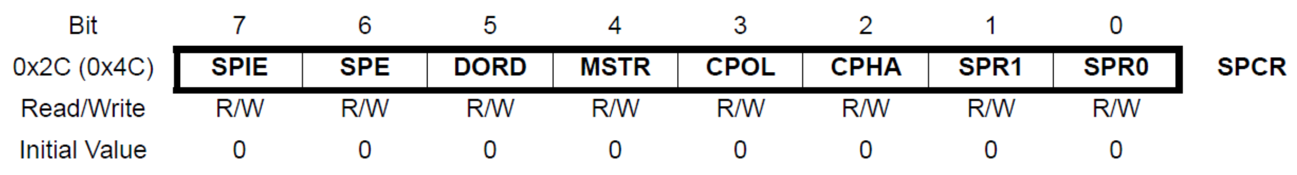
SPI DATA MODES

SPI Mode	Conditions	Leading Edge	Trailing eDge
0	CPOL=0, CPHA=0	Sample (rising)	Setup (falling)
1	CPOL=0, CPHA=1	Setup (rising)	Sample (falling)
2	CPOL=1, CPHA=0	Sample (falling)	Setup (rising)
3	CPOL=1, CPHA=1	Setup (falling)	Sample (rising)

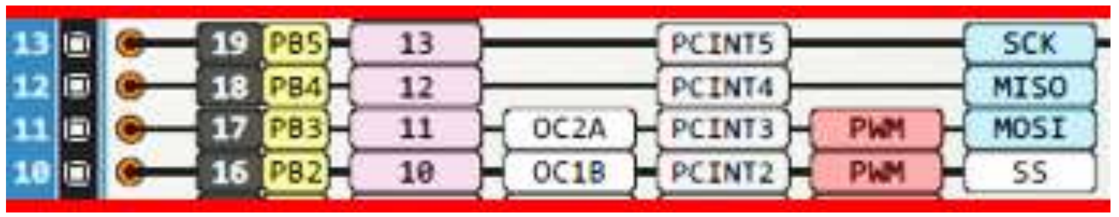
SPI Transfer Format with CPHA=1

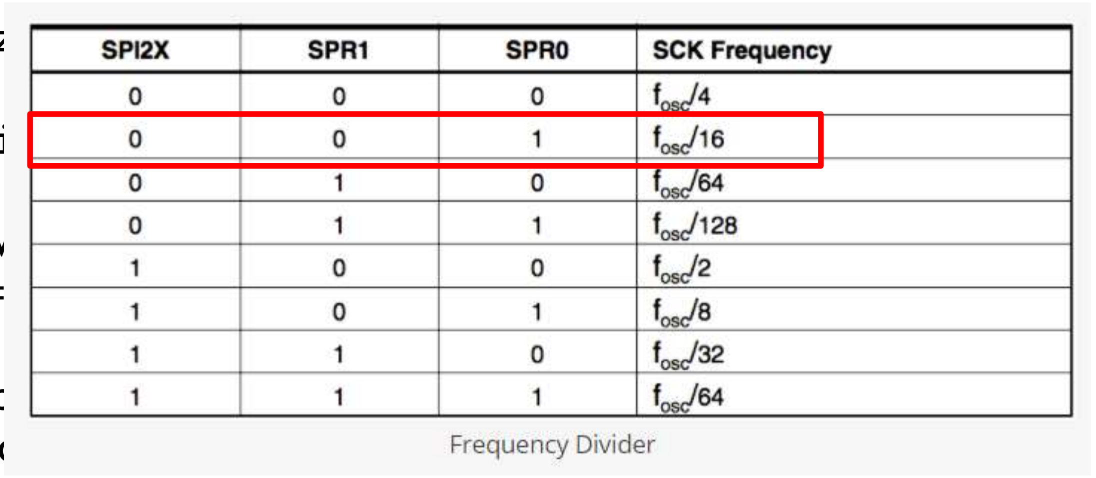


ENABLING SPI ON MASTER / SLAVE



Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	





SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Frequency Divider

```
// Initializ

void spi_i
{
   // Set M
   DDRB=

   // Enab
   // Presc

   SPCR=(1<<SPE)|(1<<MSTR)|(1<<SPR0)|(1<<SPIE);

   // Enable Global Interrupts
   sei();
}
```

```
// Initialize SPI Slave Device
void spi_init_slave (void)
{
   DDRB = (1<<4);     //MISO as OUTPUT
   SPCR = (1<<SPE);   //Enable SPI
}
```

SENDING AND RECEIVING DATA

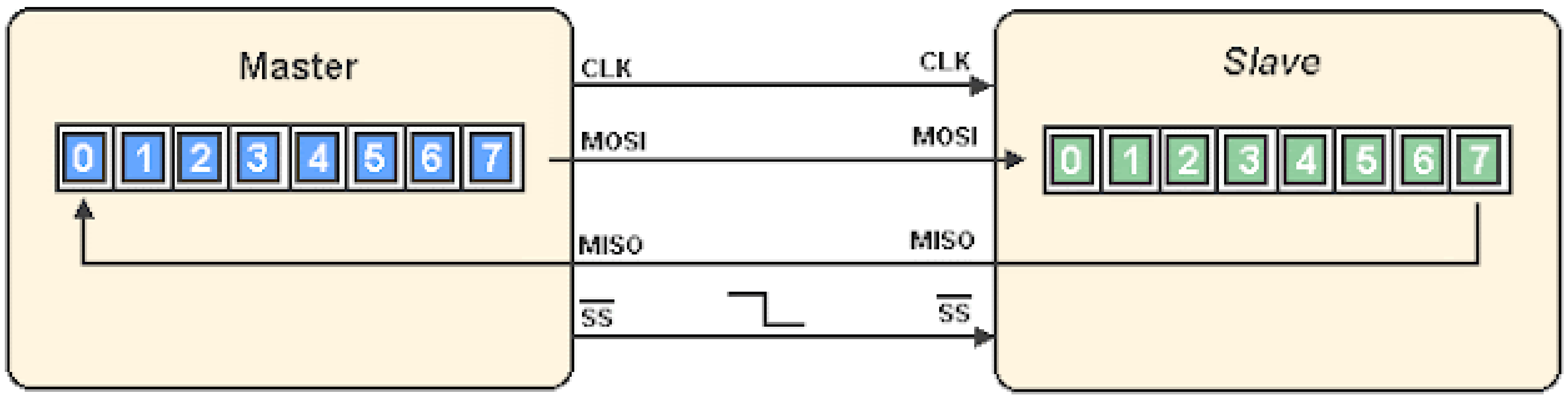
```
//Function to send and receive data for both master and slave
unsigned char spi_tranceiver (unsigned char data)
{
   // Load data into the buffer
   SPDR = data;

   //Wait until transmission complete
   while(!(SPSR & (1<<SPIF) ));

   // Return received data
   return(SPDR);
}
```

```
when SPSR.SPIF=0
  (!(SPSR & (1 << SPIF))) =
  (!(SPSR & (1 << 7))) =
  (!(SPSR & 0b10000000)) =
  (!(0)) = TRUE

when SPSR.SPIF=1
  (!(SPSR & (1 << SPIF))) =
  (!(SPSR & (1 << 7))) =
  (!(SPSR & 0b10000000)) =
  (!(128)) = FALSE
```


Master
0
1
2
3
4
5
6
7
CLK
MOSI
MISO
$\overline{SS}$
CLK
MOSI
MISO
$\overline{SS}$
Slave
0
1
2
3
4
5
6
7