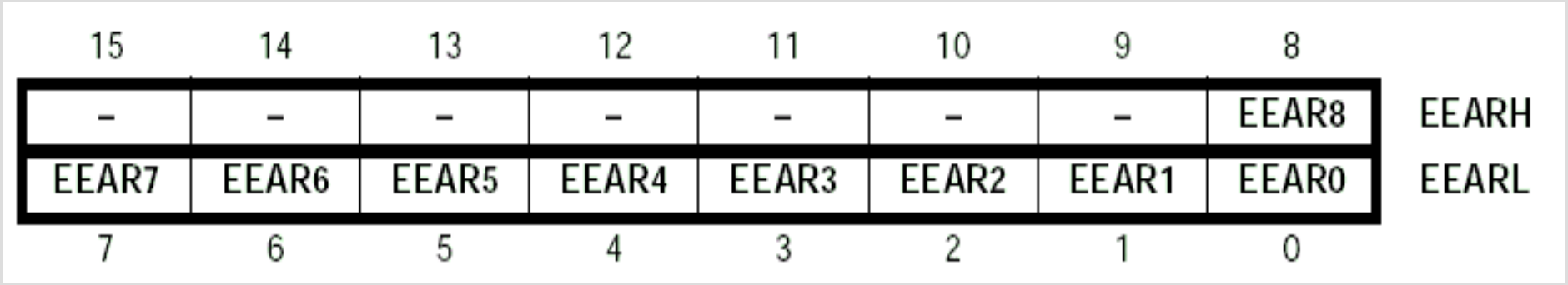
EEPROM

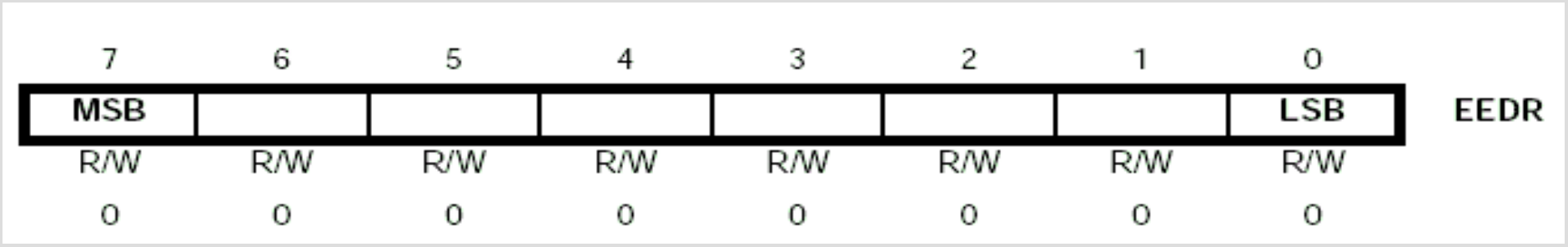
- EEPROM (Electrically Erasable Programmable Read Only Memory) is one of the three memory types of AVRs .
- EEPROM is able to retain its contents when there is no supply voltage.
- You can also change the EEPROM contents on runtime, so, EEPROM is useful to store information like calibration values, ID numbers, etc.

- To write in the EEPROM, you need to specify the data you want to write and the address at which you want to write this data.
- In order to prevent unintentional EEPROM writes (for instance, during power supply power up/down), a specific write procedure must be followed
- The write process is not instantaneous, it takes between 2.5 to 4 ms. For this reason, your software must check if the EEPROM is ready to write a new byte (maybe a previous write operation is not finished yet).

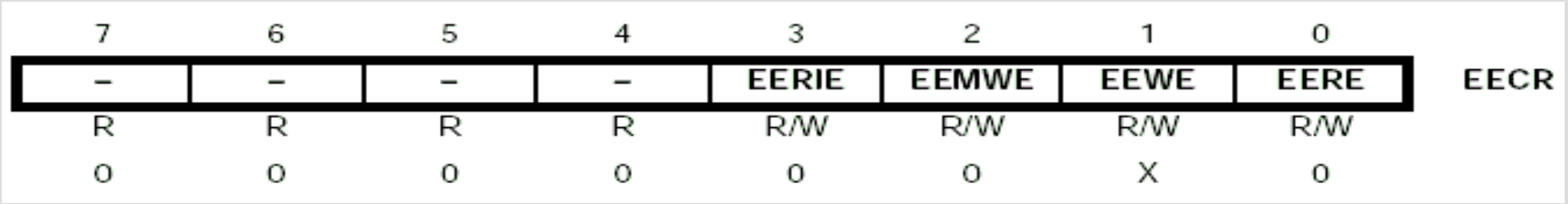
Registers



15	14	13	12	11	10	9	8	
–	–	–	–	–	–	–	EEAR8	EEARH
EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
7	6	5	4	3	2	1	0	



7	6	5	4	3	2	1	0	
MSB							LSB	EEDR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	



7	6	5	4	3	2	1	0	
–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
R	R	R	R	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	X	0	

- The EEPROM Control Register (EECR) is used to control the operation of the EEPROM
- The EERE (EEPROM Read Enable) bit is used to read the EEPROM.
- In order to issue an EEPROM write, you must first set the EEMWE (EEPROM Master Write Enable) bit, and then set the EEWE (EEPROM write enable) bit. If you don't set EEMWE first, setting EEWE will have no effect.
- The EEWE bit is also used to know if the EEPROM is ready to write a new byte. While the EEPROM is busy, EEWE is set to one, and is cleared by hardware when the EEPROM is ready. So, your program can poll this bit and wait until is cleared before writing the next byte.

The following is a code snippet for writing the data 0xAA in address 0x10 :

```
cli                                 ; disable interrupts
 EEPROM_write:                      ;if write enable bit is cleared, EEPROM is

        sbic EECR, EEWE     ;ready to be written to  else loop until EEWE
        rjmp EEPROM_write   ;cleared

        ldi r16, 0x10               ;load r16 with address (0x10)
        out EEAR, r16       ;and write it to the address register

        ldi r16, 0xAA               ;load with the data we want to write (0xAA)
        out EEDR, r16       ;and write it to the data register

        sbi EECR; EEMWE             ;set master write enable bit
        sbi EECR, EEWE      ; set write enable bit

sei                                 ; and allow interrupts again
```

Eeeprom read

- To read a data from the EEPROM, you must first check that the EEPROM is not busy by polling the EEWE bit

- Then you set the EEAR register with the address you want to read, and then set the EERE bit in the EECR register.

- After that, the requested data is found in the EEDR register.

The following is a code snippet for reading the data stored in address 0x10. The read data is stored in r16.

```
EEPROM_read:

sbic EECR, EEWE   ;check if the EEPROM is busy writing a
      `                    ;byte
rjmp EEPROM_read

ldi r16, 0x10              ;load address register with 0x10
out EEAR, r16

sbi EECR, EERE     ;set read enable bit
in r16, EEDR       ; and get the data from address 0x10
```

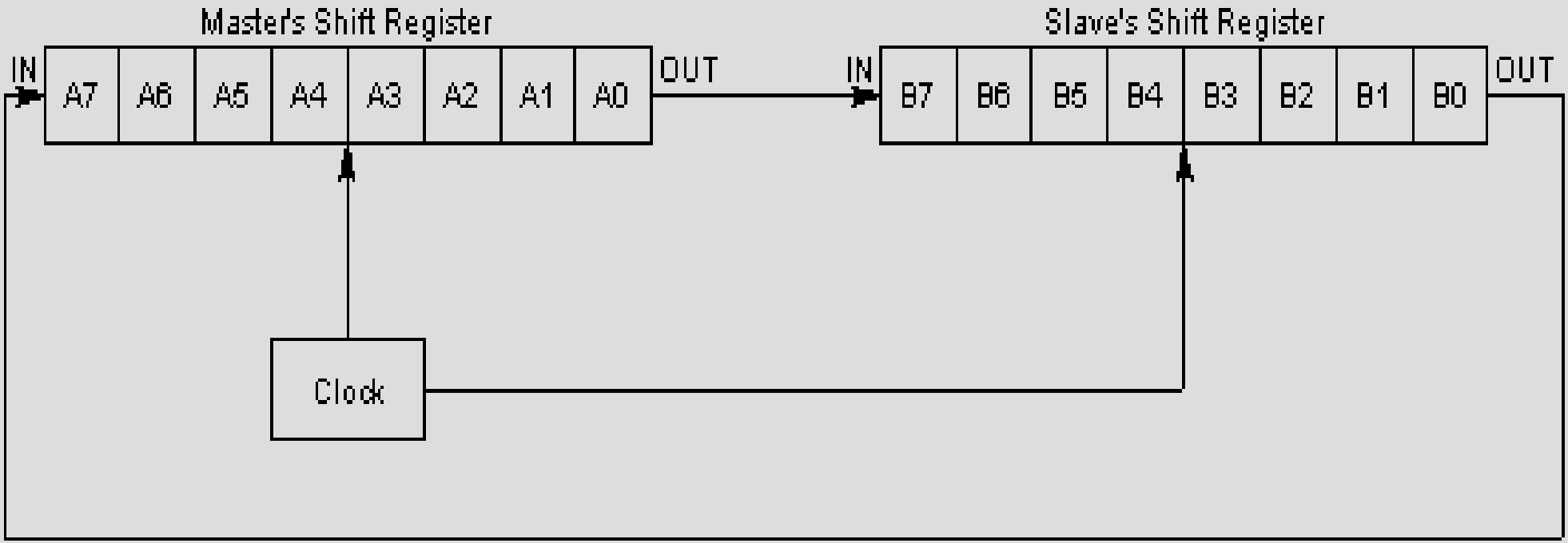
The Serial Peripheral Interface (SPI)

- The SPI (Serial Peripheral Interface) is a peripheral used to communicate between the AVR and other devices, like others AVRs, external EEPROMs, DACs, ADCs, etc.

- With this interface, you have one Master device which initiates and controls the communication, and one or more slaves who receive and transmit to the Master.

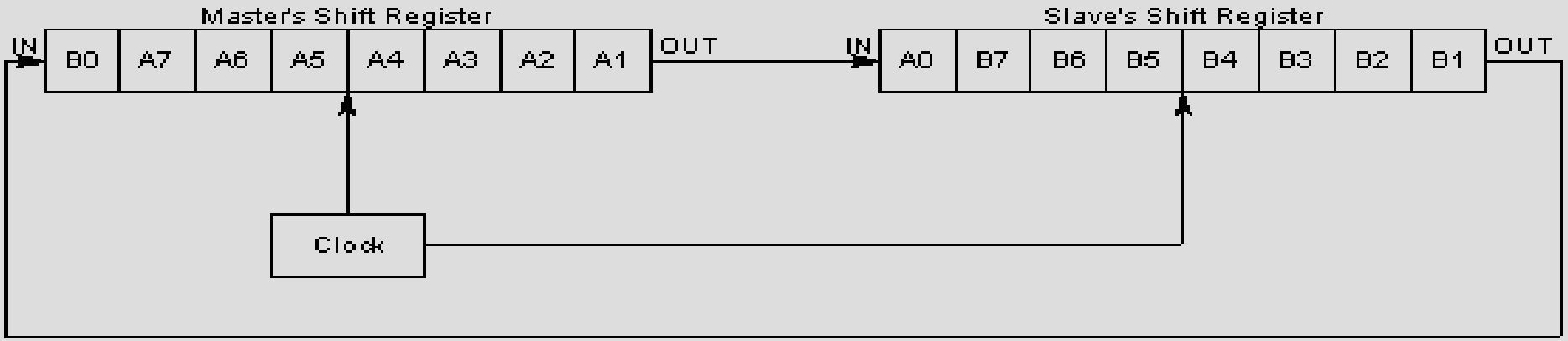
- The core of the SPI is an 8-bit shift register in both the Master and the Slave, and a clock signal generated by the Master.
- Let's say the Master wants to send a byte of data (call it A) to the Slave and at the same time receive another byte of data from the Slave (call it B).
- Before starting the communication, the Master places A in its shift register, and the Slave places B in its shift register.

Then the Master generates 8 clock pulses, and the contents of the Master's shift register are transferred to the Slave's shift register and vice versa

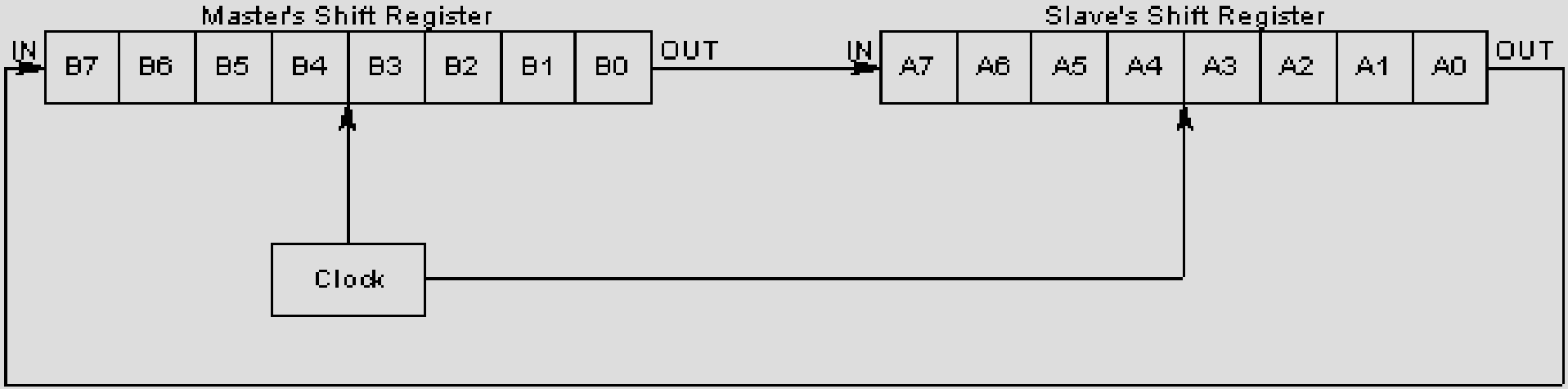


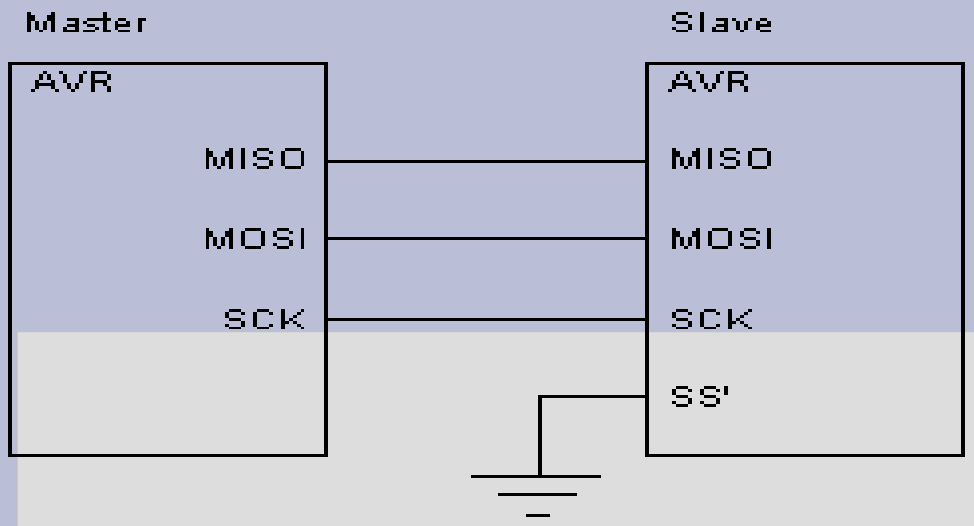
So, at the end of the clock pulses, the Master has completely received B, and the Slave has received A. As you can see, the transmission and reception occurs at the same time, so it is a full duplex data transfer

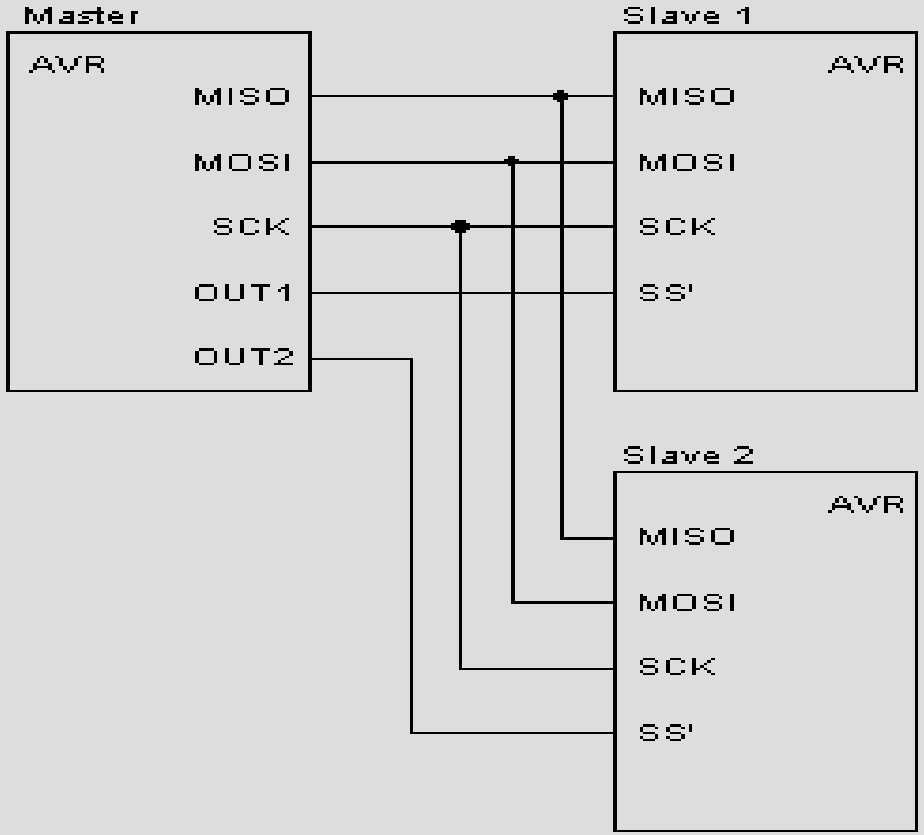
Master generates the first clock pulse:



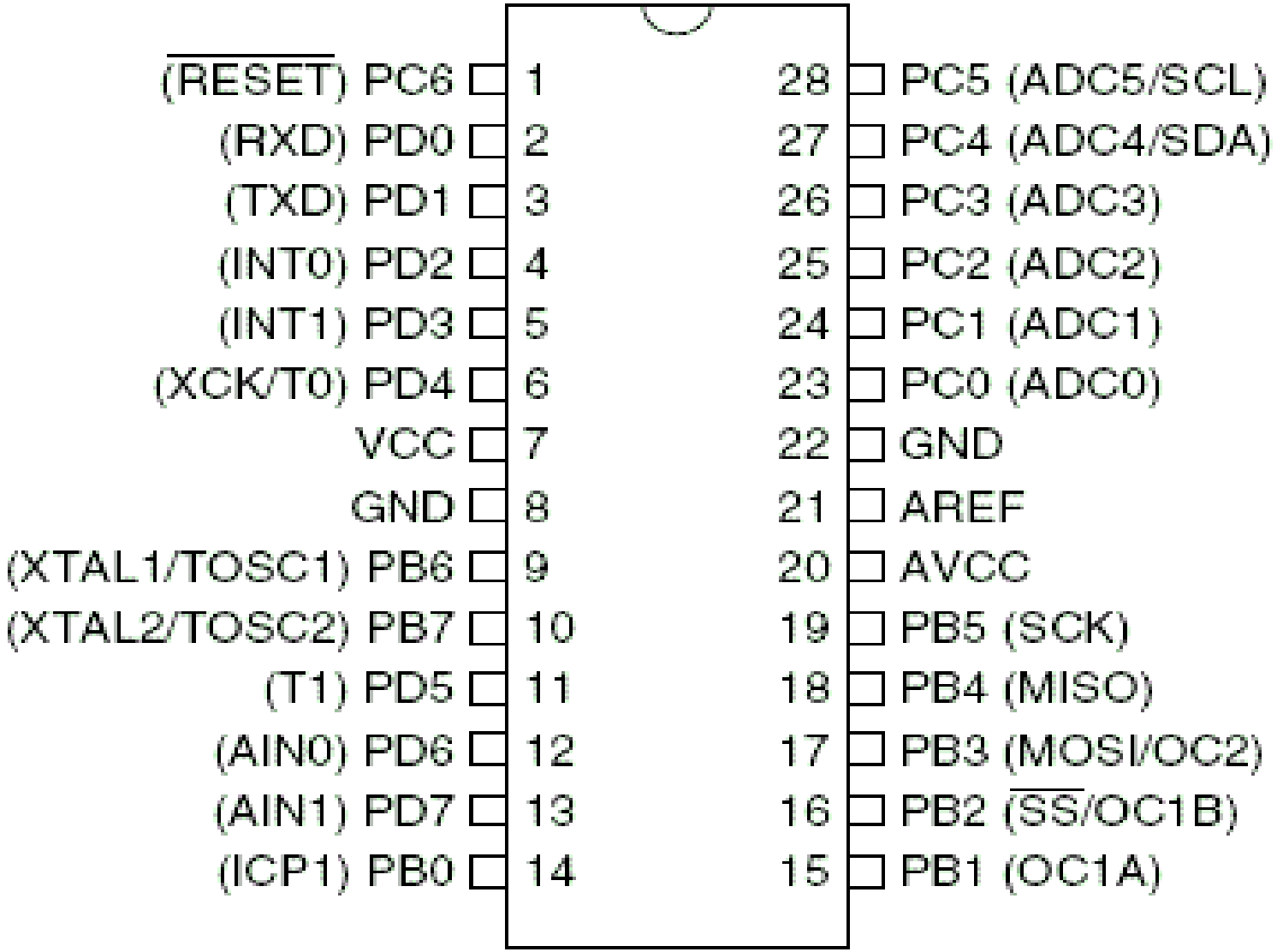
Master generates the last clock pulse:




Master
AVR
MISO
MOSI
SCK
Slave
AVR
MISO
MOSI
SCK
SS'


Master
AVR
MISO
MOSI
SCK
OUT1
OUT2
Slave 1
AVR
MISO
MOSI
SCK
SS'
Slave 2
AVR
MISO
MOSI
SCK
SS'

PDIP


(RESET) PC6 1
(RXD) PD0 2
(TXD) PD1 3
(INT0) PD2 4
(INT1) PD3 5
(XCK/T0) PD4 6
VCC 7
GND 8
(XTAL1/TOSC1) PB6 9
(XTAL2/TOSC2) PB7 10
(T1) PD5 11
(AIN0) PD6 12
(AIN1) PD7 13
(ICP1) PB0 14
28 PC5 (ADC5/SCL)
27 PC4 (ADC4/SDA)
26 PC3 (ADC3)
25 PC2 (ADC2)
24 PC1 (ADC1)
23 PC0 (ADC0)
22 GND
21 AREF
20 AVCC
19 PB5 (SCK)
18 PB4 (MISO)
17 PB3 (MOSI/OC2)
16 PB2 (SS/OC1B)
15 PB1 (OC1A)

- In an AVR, four signals (pins) are used for the SPI: MISO, MOSI, SCK and SS' (SS' means SS complemented).

- MISO (Master In Slave Out): the input of the Master's shift register, and the output of the Slave's shift register.

- MOSI (Master Out Slave In): the output of the Master's shift register, and the input of the Slave's shift register.

- SCK (Serial Clock): In the Master, this is the output of the clock generator. In the Slave, it is the input clock signal.

- SS' (Slave Select): Since in an SPI setup you can have several slaves at the same time, you need a way to select which Slave you want to communicate to.

- If SS' is held in a high state, all Slave SPI pins are normal inputs, and will not receive incoming SPI data.

- On the other hand, if SS' is held in a low state, the SPI is activated. The software of the Master must control the SS'-line of each Slave.

- If the SPI-device is configured as a Master, the behavior of the SS' pin depends on the configured data direction of the pin.

- If SS' is configured as an output, the pin does not affect the SPI.

- f SS' is configured as an input, it must be held high to ensure Master SPI operation.

- If the SS' pin is driven low, the SPI system interprets this as another Master selecting the SPI as a Slave and starting to send data to it.

- Having two SPI Masters is quite unusual

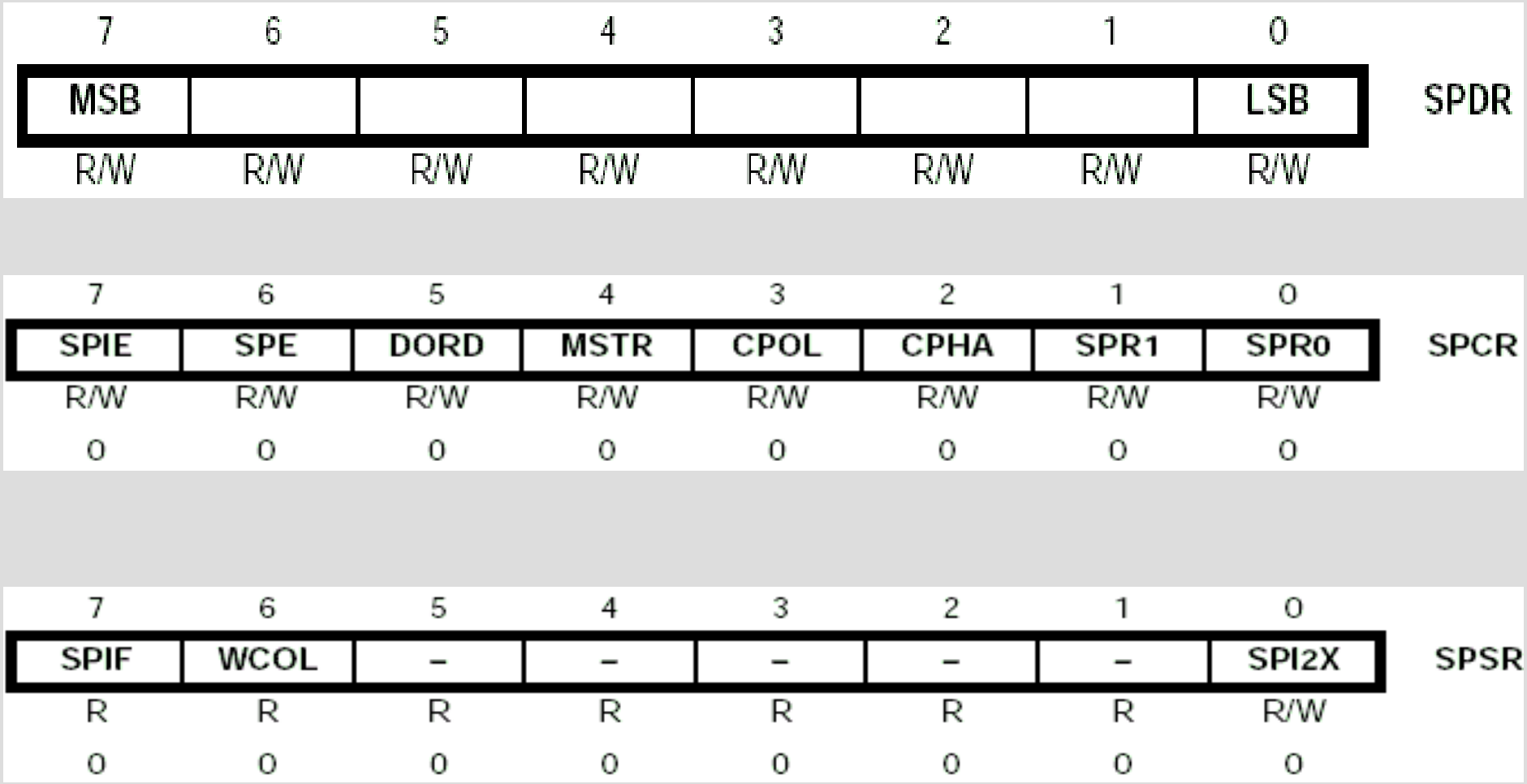
A word of caution about the SPI pin names. MISO, MOSI, SCK and SS' are the names used by AVRs. Other devices may use a different set of names. You must check the data sheet of the particular device you are using to get them right.

What are the data directions of the SPI pins?
It depends on the particular pin and on whether the SPI is set as a Master or Slave

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

Registers

[SPCR] [SPSR] [SPDR]



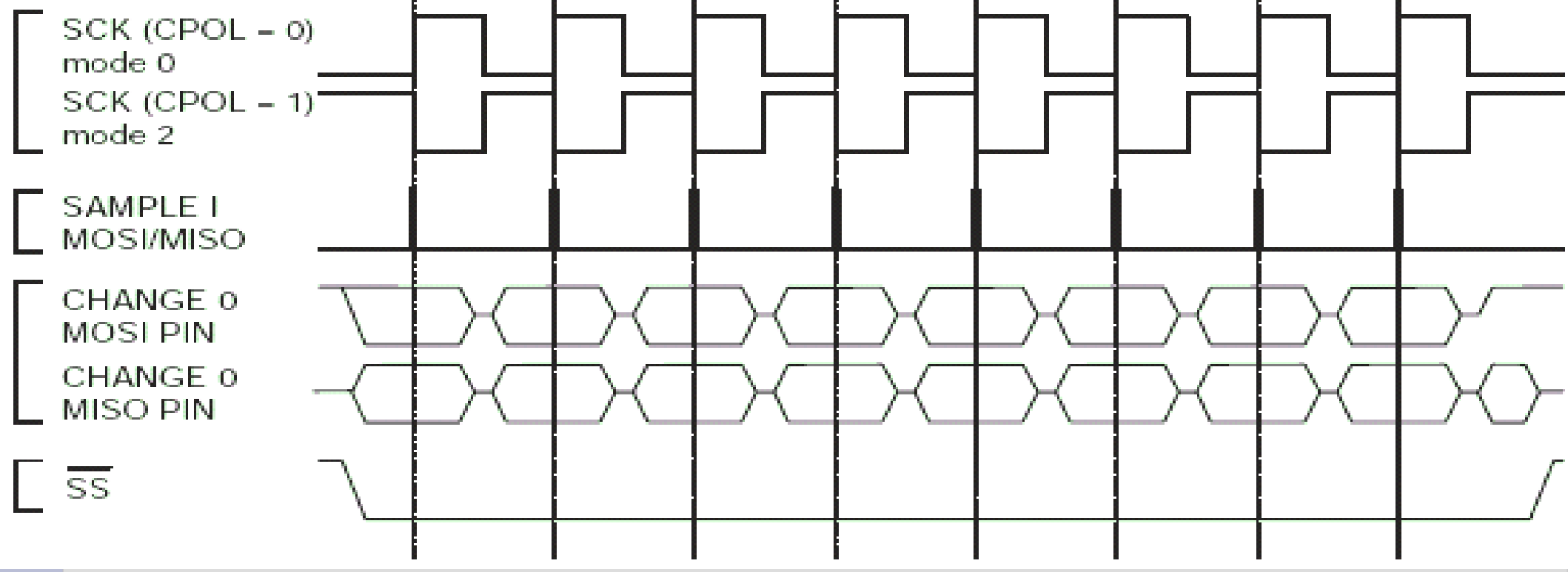
These two bits control the SCK rate of the device configured as a master. SPR1 and SPR0 have no effect on the slave.

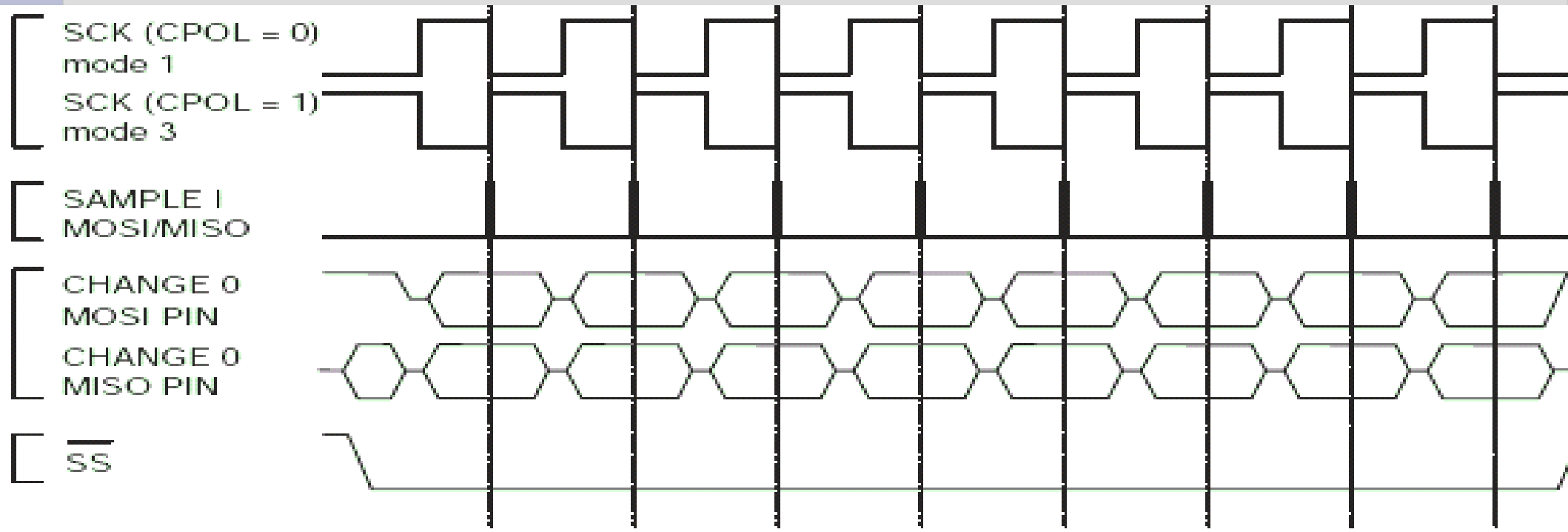
SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

-

- CPOL: Clock Polarity-> When this bit is written to one, SCK is high when idle. When CPOL is written to zero,SCK is low when idle.

- CPHA: Clock Phase-> The settings of the clock phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK.


SCK (CPOL – 0)
mode 0
SCK (CPOL – 1)
mode 2
SAMPLE I
MOSI/MISO
CHANGE 0
MOSI PIN
CHANGE 0
MISO PIN
SS


SCK (CPOL = 0)
mode 1
SCK (CPOL = 1)
mode 3
SAMPLE I
MOSI/MISO
CHANGE 0
MOSI PIN
CHANGE 0
MISO PIN
SS

SPI is used to talk to a variety of peripherals

- Sensors: temperature, pressure, ADC, touchscreens
- Control devices: audio codecs, digital potentiometers, DAC
- Communications: Ethernet, USB, USART, CAN, IEEE 802.15.4
- Memory: flash and EEPROM
- Real-time clocks
- LCD displays, sometimes even for managing image data
- Any MMC or SD card

For example program of SPI
reffer to page number 124-132
of Data Sheet

master side algo

- ->>Set MOSI as output.
- ->> Set SCK as output.
- ->> Set SS' as output.
- ->>Set SPI as a Master, with interrupt disabled,MSB first, SPI mode 3 and clock frequency .
- ->> Initiate data transfer.
- ->> Wait for transmission to complete.
- ->>The received data is placed in register.

Slave side algo

- Set MISO as an output.
- Set SPI as a Slave, with interrupt disabled,
- MSB first and SPI mode 3.
- Send data on Master request.
- Wait for reception to complete.
- The received data is placed in register.

watchdog timer

A watchdog timer is a piece of hardware, often built into a microcontroller that can cause a processor reset when it judges that the system has hung, or is no longer executing the correct sequence of code

Kicking the dog

- The process of restarting the watchdog timer's counter is sometimes called "kicking the dog."

7	6	5	4	3	2	1	0	
–	–	–	WDCE	WDE	WDP2	WDP1	WDP0	WDTCR
R	R	R	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Table 22. Watchdog Timer Prescale Select

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 3.0V	Typical Time-out at V_{CC} = 5.0V
0	0	0	16K (16,384)	14.8 ms	14.0 ms
0	0	1	32K (32,768)	29.6 ms	28.1 ms
0	1	0	64K (65,536)	59.1 ms	56.2 ms
0	1	1	128K (131,072)	0.12 s	0.11 s
1	0	0	256K (262,144)	0.24 s	0.22 s
1	0	1	512K (524,288)	0.47 s	0.45 s
1	1	0	1,024K (1,048,576)	0.95 s	0.9 s
1	1	1	2,048K (2,097,152)	1.9 s	1.8 s

Disabling Watchdog

```
WDT_off:
   ; Reset WDT
   wdr
   in r16, WDTCR
   ; Write logical one to WDCE and WDE
   ori  r16, (1<<WDCE)|(1<<WDE)
   out  WDTCR, r16
   ; Turn off WDT
   ldi  r16, (0<<WDE)
   out  WDTCR, r16
   ret
```

Default Values

M103C	WDTON	Safety Level	WDT Initial State	How to Disable the WDT	How to Change Time-out
Unprogrammed	Unprogrammed	1	Disabled	Timed sequence	Timed sequence
Unprogrammed	Programmed	2	Enabled	Always enabled	Timed sequence
Programmed	Unprogrammed	0	Disabled	Timed sequence	No restriction
Programmed	Programmed	2	Enabled	Always enabled	Timed sequence