Mechatronics Engineering

Tutorial #5

SPI Communication
Atmega328P ADC

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

Tutorial Contents

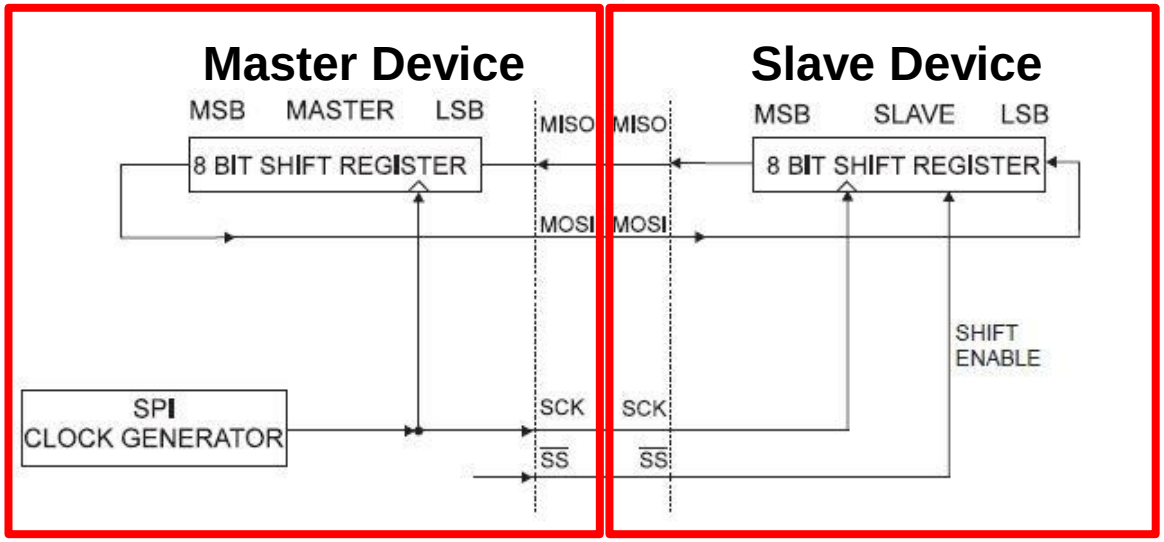
- SPI Communication
- Analog to Digital Converter in ATmega328P

Tutorial Contents

- **SPI Communication**
- Analog to Digital Converter in ATmega328P

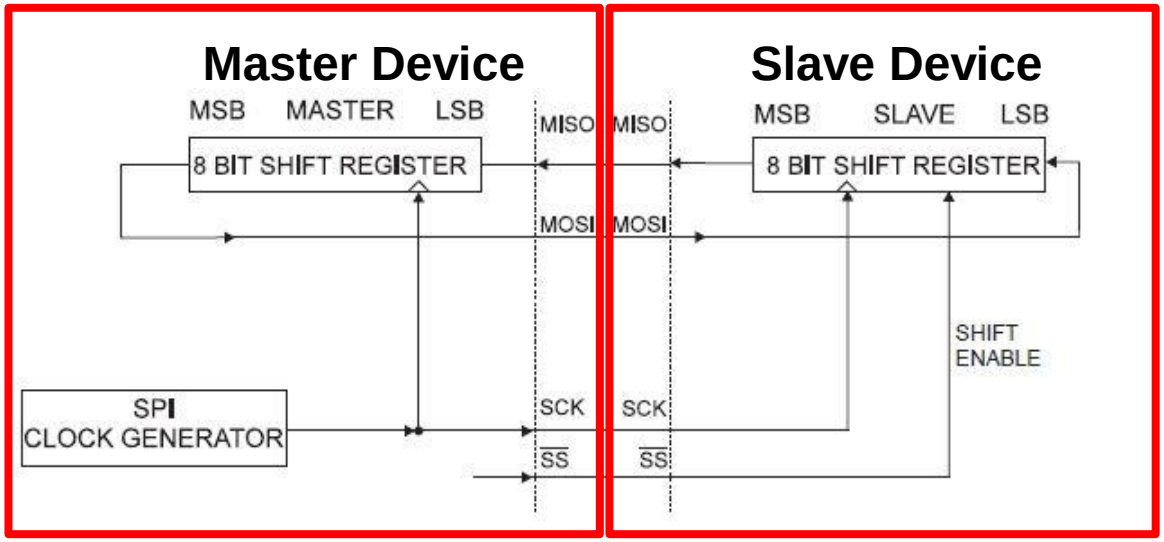
Serial Peripheral Interface (SPI):

- SPI is a synchronous communication protocol.
- When transmission is initiated, the data in the shift register of the slave and master devices are exchanged through the two pins: Master In Slave Out (**MISO**) and Master Out Slave In (**MOSI**) simultaneously.



Serial Peripheral Interface (SPI):

- Data transmission is synchronized using (**SCK**) clock signal, which is generated by the master device.
- To enable communication with a slave device, the Slave Select (**SS**) pin of the slave device must be pulled low.



ATmega328P SPI Register Description:

- SPI Registers:
  - SPDR: SPI Data Register
  - SPCR: SPI Control Register
  - SPSR: SPI Status Register

19.5.3 SPDR – SPI Data Register

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

The SPI Data Register is a read/write register used for data transfer between the Register File and the SPI Shift Register. Writing to the register initiates data transmission. Reading the register causes the Shift Register Receive buffer to be read.

ATmega328P SPI Register Description:

19.5.1 SPCR – SPI Control Register

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**

This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and the if the Global Interrupt Enable bit in SREG is set.

- **Bit 6 – SPE: SPI Enable**

When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**

When the DORD bit is written to one, the LSB of the data word is transmitted first.

When the DORD bit is written to zero, the MSB of the data word is transmitted first.

- **Bit 4 – MSTR: Master/Slave Select**

This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If $\overline{SS}$ is configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI Master mode.

ATmega328P SPI Register Description:

CPOL and CPHA bits must have the same settings for both the master and slave devices

- **Bit 3 – 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. Refer to Figure 19-3 and Figure 19-4 for an example. The CPOL functionality is summarized below:

Table 19-3. **CPOL Functionality**

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

- **Bit 2 – 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. Refer to Figure 19-3 and Figure 19-4 for an example. The CPOL functionality is summarized below:

Table 19-4. CPHA Functionality

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

- **Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0**

These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect on the Slave. The relationship between SCK and the Oscillator Clock frequency f_{osc} is shown in the following table:

ATmega328P SPI Register Description:

Table 19-5. Relationship Between SCK and the Oscillator Frequency

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	$f_{osc}/64$

For a slave ATmega328P to communicate successfully:

$$f_{sck} \leq \frac{f_{osc}}{4}$$

19.5.2 SPSR – SPI Status Register

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**

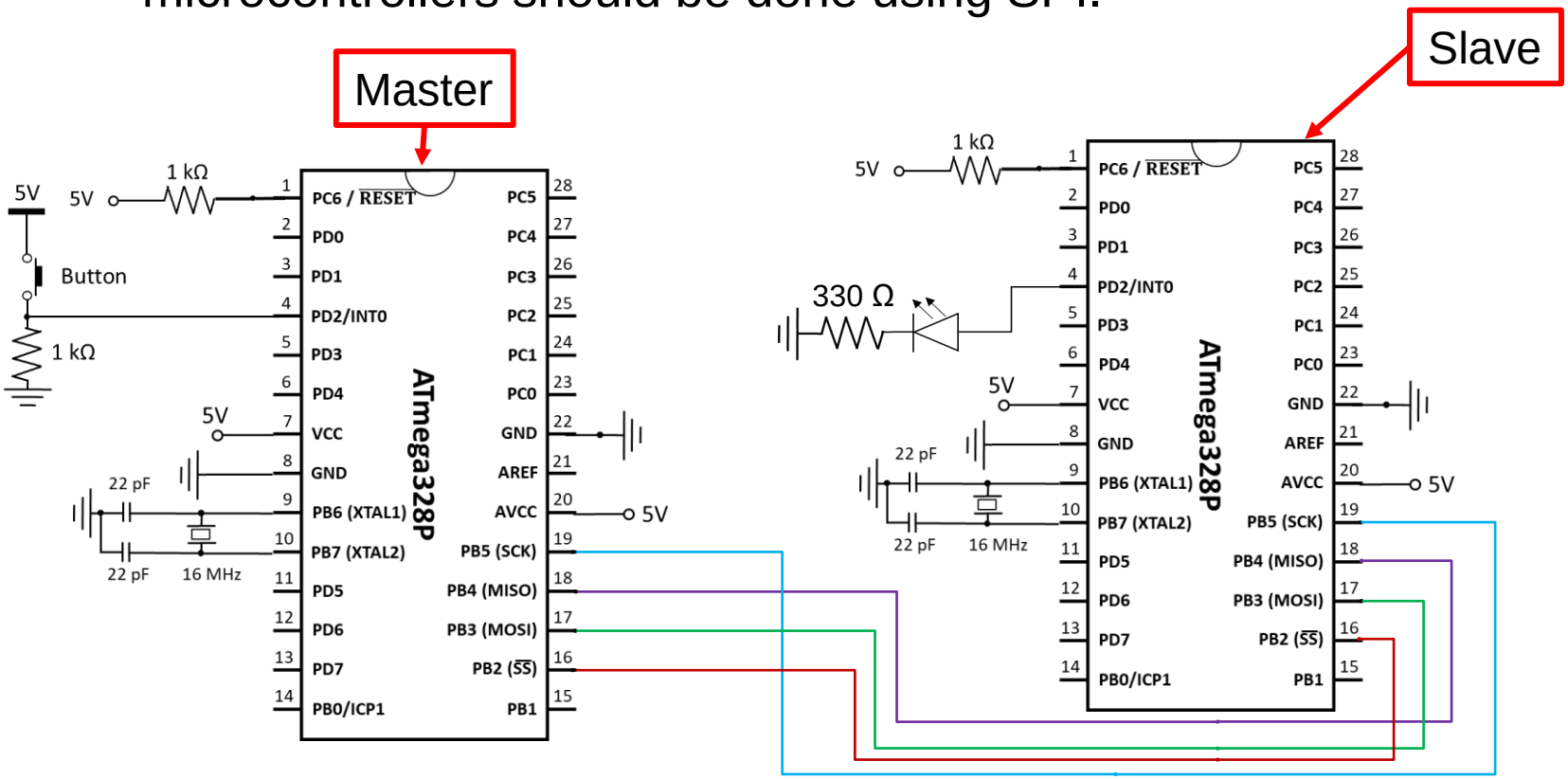
When a serial transfer is complete, the SPIF Flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If $\overline{SS}$ is an input and is driven low when the SPI is in Master mode, this will also set the SPIF Flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI Status Register with SPIF set, then accessing the SPI Data Register (SPDR).

- **Bit 6 – WCOL: Write COLlision Flag**

The WCOL bit is set if the SPI Data Register (SPDR) is written during a data transfer. The WCOL bit (and the SPIF bit) are cleared by first reading the SPI Status Register with WCOL set, and then accessing the SPI Data Register.

Problem 1:

- Write a C program to display the state of a push-button connected to one ATmega328P(Master) on an LED connected to another ATmega328P(Slave). Communication between the two microcontrollers should be done using SPI.



Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

Master Device Program:

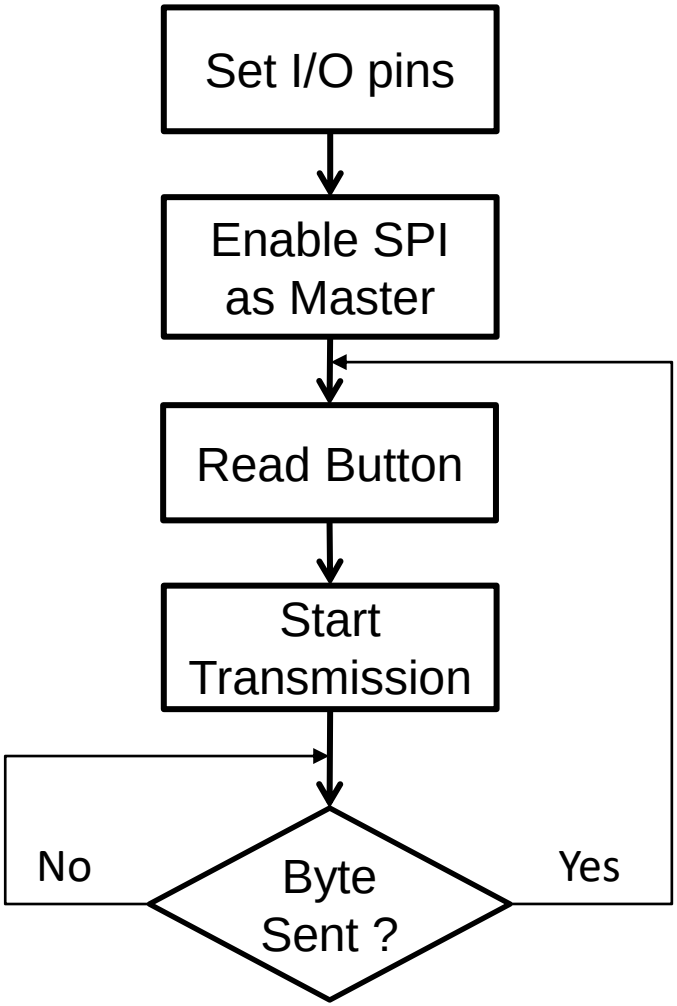
SPCR – SPI Control Register

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	

- Enable SPI (**SPE** = 1)
- Set device as master (**MSTR** = 1)
- Set $f_{sck} = \frac{f_{osc}}{16}$ (**SPR0** = 1, **SPI2X** = **SPR1** = 0)

- Register settings: **SPCR** = 0b01010001;

Master Device Program:



Pin Directions		
	Master	Slave
SCK (PB5)	Output	Input
MISO (PB4)	Input	Output
MOSI (PB3)	Output	Input
SS (PB2)	Output	Input
PD2	Button (In)	LED (Out)

Master Device Program:

Master Code:

Pin Directions		
	Master	Slave
SCK (PB5)	Output	Input
MISO (PB4)	Input	Output
MOSI (PB3)	Output	Input
SS (PB2)	Output	Input
PD2	Button (In)	LED (Out)

```
#include <avr/io.h>

int main(void)
{

    DDRB   = 0b00101100; // configure I/O pins of SPI
    DDRD  &= 0b11111011; // set PD2 as input for push-button
    SPCR   = 0b01010001; // Enable SPI as master device, fsck=fosc/16
    PORTB &= 0b11111011;
    while (1)
    {

        SPDR= (0b00000100 & PIND); // send the second bit of Port D (push-button)
        while(!(SPSR & 0b10000000)); // wait for transfer to be completed

    }
}
```

Slave Device Program:

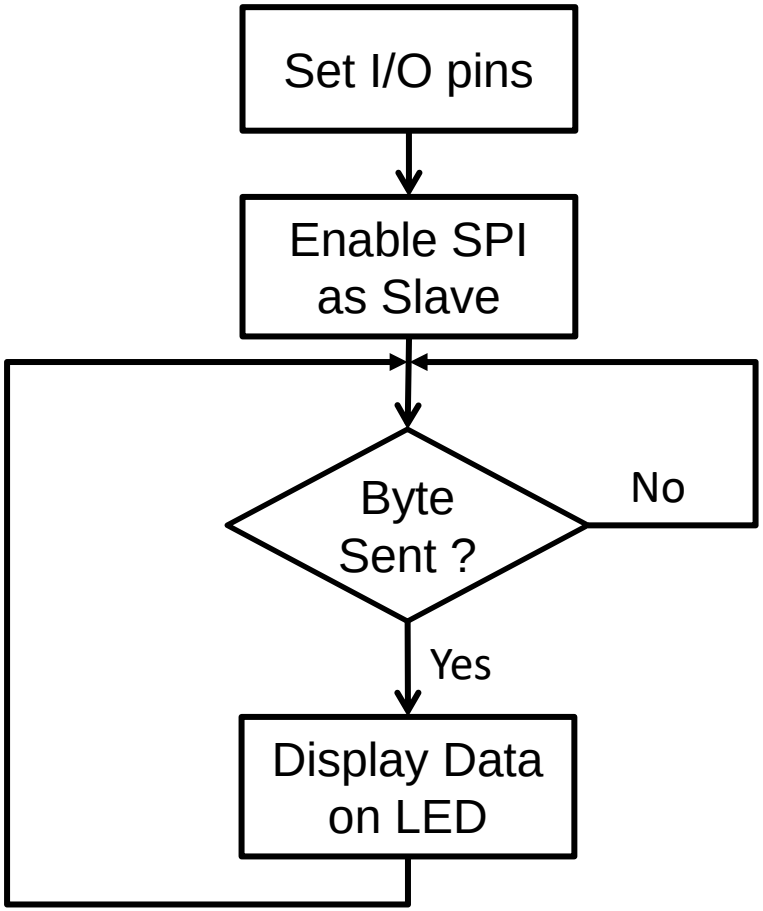
SPCR – SPI Control Register

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	

- Enable SPI (**SPE** = 1)
- Set device as slave (**MSTR** = 0)
- **SPR0** and **SPR1** do not matter (clock generated by master).

- Possible Register settings: **SPCR** = 0b01000001;

Slave Device Program:



Pin Directions		
	Master	Slave
SCK (PB5)	Output	Input
MISO (PB4)	Input	Output
MOSI (PB3)	Output	Input
SS (PB2)	Output	Input
PD2	Button (In)	LED (Out)

Slave Device Program:

Code:

Pin Directions		
	Master	Slave
SCK (PB5)	Output	Input
MISO (PB4)	Input	Output
MOSI (PB3)	Output	Input
SS (PB2)	Output	Input
PD2	Button (In)	LED (Out)

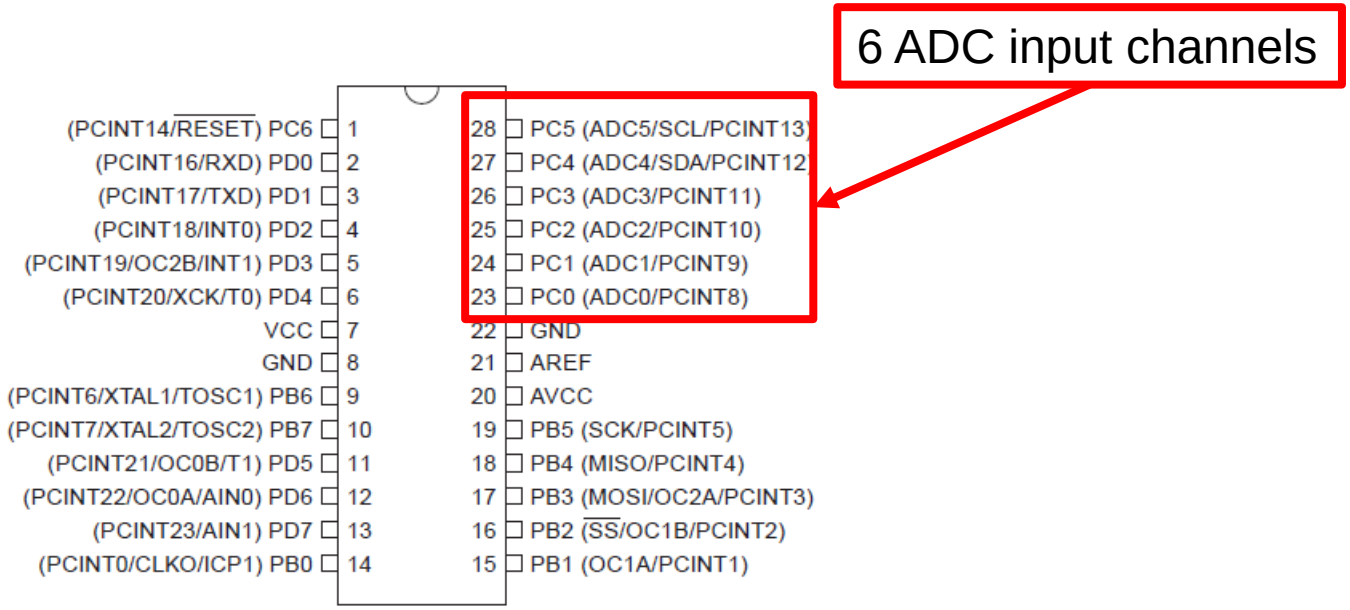
```
#include <avr/io.h>

int main()
{

    DDRB   = 0b00010000; // configure I/O pins of SPI
    DDRD  |= 0b00000100; // set PD2 as output for LED
    SPCR   = 0b01000001; // Enable SPI for slave device
    while (1)
    {

        while(!(SPSR & 0b10000000)); // wait for data to be received
        PORTD=SPDR; // display data on PortD

    }
}
```

Tutorial Contents

- SPI Communication
- **Analog to Digital Converter in ATmega328P**

Atmega328P Analog to Digital Converter(ADC):

- ATmega328P ADC:
  - 10-bit resolution ADC
  - 6 Multiplexed ADC input channels
  - ADC supply voltage pin (**AVCC**) must not differ than **Vcc** by ($\pm 0.3\ V$)



ATmega328P ADC Register Description:

- ADC Registers:
  - ADCL: ADC Data Register Low
  - ADCH: ADC Data Register High
  - ADMUX: ADC Multiplexer Selection Register
  - ADCSRA: ADC Control and Status Register A
  - ADCSRB: ADC Control and Status Register B
  - DIDR0: Digital Input Disable Register 0

ATmega328P ADC Register Description:

ADC data registers are left or right adjusted based on ADLAR bit from ADMUX register

24.9.3 ADCL and ADCH – The ADC Data Register

24.9.3.1 ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

24.9.3.2 ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

When an ADC conversion is complete, the result is found in these two registers.

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

ATmega328P ADC Register Description:

24.9.1 ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – REFS[1:0]: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in Table 24-3. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

Table 24-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal V_{ref} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V Voltage Reference with external capacitor at AREF pin

- **Bit 5 – ADLAR: ADC Left Adjust Result**

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions. For a complete description of this bit, see "ADCL and ADCH – The ADC Data Register" on page 259.

ATmega328P ADC Register Description:

- **Bits 3:0 – MUX[3:0]: Analog Channel Selection Bits**

The value of these bits selects which analog inputs are connected to the ADC. See Table 24-4 for details. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set).

Table 24-4. Input Channel Selections

MUX3...0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8(1)
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (V_{BG})
1111	0V (GND)

Note: 1. For Temperature Sensor.

ADC6 & ADC7 not supported on the 28 pin MCU package

ATmega328P ADC Register Description:

24.9.2 ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

- **Bit 5 – ADATE: ADC Auto Trigger Enable**

When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB.

ATmega328P ADC Register Description:

- **Bit 4 – ADIF: ADC Interrupt Flag**

This bit is set when an ADC conversion completes and the Data Registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-Write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

- **Bit 3 – ADIE: ADC Interrupt Enable**

When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

- **Bits 2:0 – ADPS[2:0]: ADC Prescaler Select Bits**

These bits determine the division factor between the system clock frequency and the input clock to the ADC.

Table 24-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

The ADC clock should be between $50kHz$ and $200kHz$ for maximum accuracy. This is achieved using the ADC clock pre-scaler

$$f_{ADC} = \frac{f_{clk}}{DF}$$

DF: Division Factor

ATmega328P ADC Register Description:

24.9.4 ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7, 5:3 – Reserved**

These bits are reserved for future use. To ensure compatibility with future devices, these bits must be written to zero when ADCSRB is written.

- **Bit 2:0 – ADTS[2:0]: ADC Auto Trigger Source**

If ADATE in ADCSRA is written to one, the value of these bits selects which source will trigger an ADC conversion. If ADATE is cleared, the ADTS[2:0] settings will have no effect. A conversion will be triggered by the rising edge of the selected Interrupt Flag. Note that switching from a trigger source that is cleared to a trigger source that is set, will generate a positive edge on the trigger signal. If ADEN in ADCSRA is set, this will start a conversion. Switching to Free Running mode (ADTS[2:0]=0) will not cause a trigger event, even if the ADC Interrupt Flag is set.

Table 24-6. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

ATmega328P ADC Register Description:

24.9.5 DIDR0 – Digital Input Disable Register 0

Bit	7	6	5	4	3	2	1	0	
(0x7E)	–	–	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	DIDR0
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

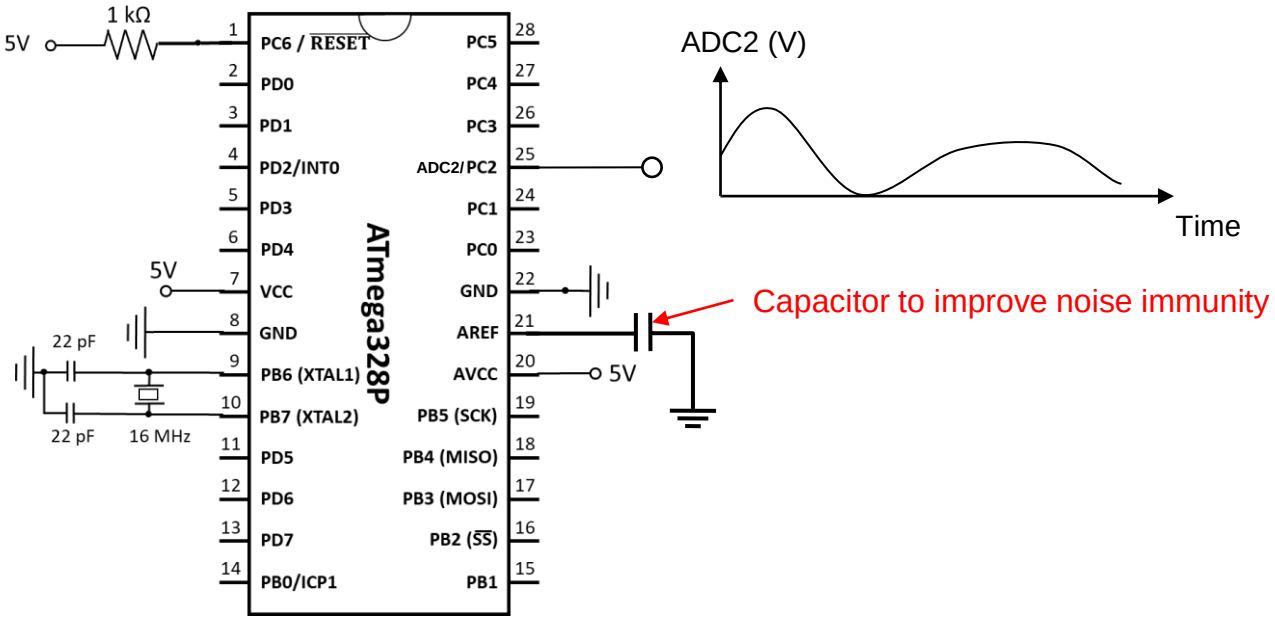
- **Bit 5:0 – ADC5D...ADC0D: ADC5...0 Digital Input Disable**

When this bit is written logic one, the digital input buffer on the corresponding ADC pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the ADC5...0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

Note that ADC pins ADC7 and ADC6 do not have digital input buffers, and therefore do not require Digital Input Disable bits.

Problem 2:

- Write a C program to read an analog signal connected to ADC channel 2 and display the most significant 8 bits of the conversion result on PORTD. Use AVCC as a reference voltage



Problem 2:

ADC settings:

- **ADMUX** Register:

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX

  - Use AVCC as reference voltage **AREF**.(**REFS**[1:0]= 01)
  - Left Adjust the ADC conversion result. (**ADLAR** = 1)
  - Select ADC channel 2. (**MUX**[3:0]= 0010)
- **ADCSRA** Register:

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA

  - Enable ADC: (**ADEN** = 1)
  - Disable Auto Trigger Mode. (**ADATE** = 0)
  - Disable ADC interrupt: (**ADIE** = 0)
  - The ADC clock should be between $50kHz$ and $200kHz$ for maximum accuracy. This is achieved using the ADC clock pre-scaler:

$$f_{ADC} = f_{osc}/(prescaler)$$

  - A pre-scaler of 128 will result in $f_{ADC} = 125\ kHz$. (**ADPS** [2:0] = 111)
- **ADCSRB** Register:

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB

  - Register has no effect since auto trigger mode is disabled.
- **DIDR0** Register: disable digital input buffer by setting all bits in this register to 1.

Problem 2:

Code:

```
#include <avr/io.h>

int main()
{
    DDRC &= 0b11111011; // set ADC2 as input
    DDRD  = 0b11111111; // set PORTD as output
    ADMUX = 0b00100010;
    ADCSRA= 0b10000111;
    ADCSRB= 0;
    DIDR0 = 0b11111111; // disable digital input buffer

    while(1)
    {
        ADCSRA|=0b01000000; // start conversion ADSC = 1
        while((ADCSRA & 0b01000000)); // wait until conversion ends ADSC returns to 0
        PORTD = ADCH; // display 8 most significant bits on PORTD
    }

}
```