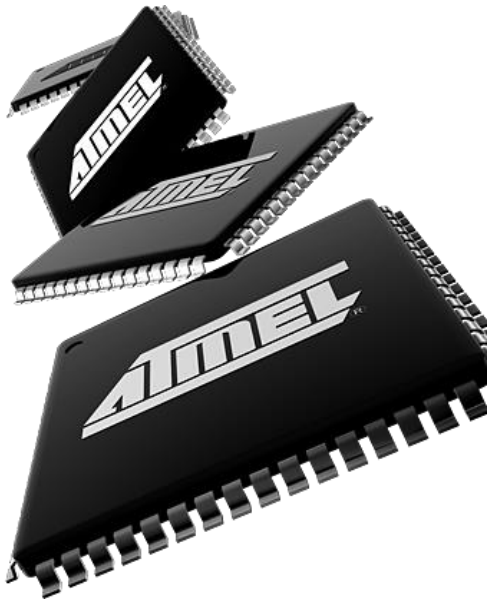
Principles and Applications of Microcontrollers

Yan-Fu Kuo
Dept. of Biomechatronics Engineering
National Taiwan University

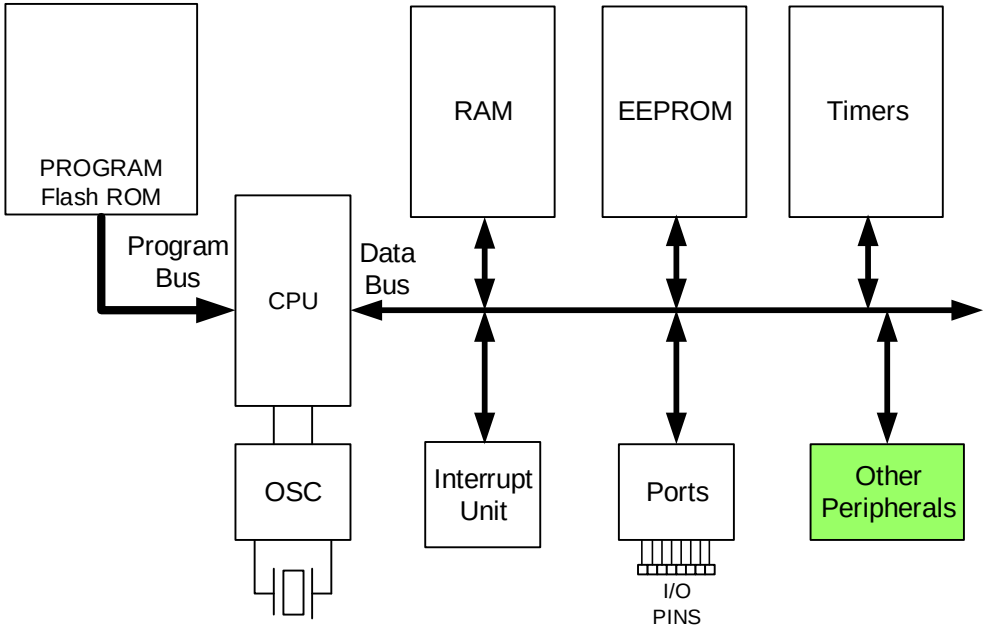
Today:

- Serial communication



Outline

- Considerations
- Wiring
- Baudrate and data framing
- AVR USART programming
- Synchronous data transmission
  - SPI
  - I^2C
- Getting started



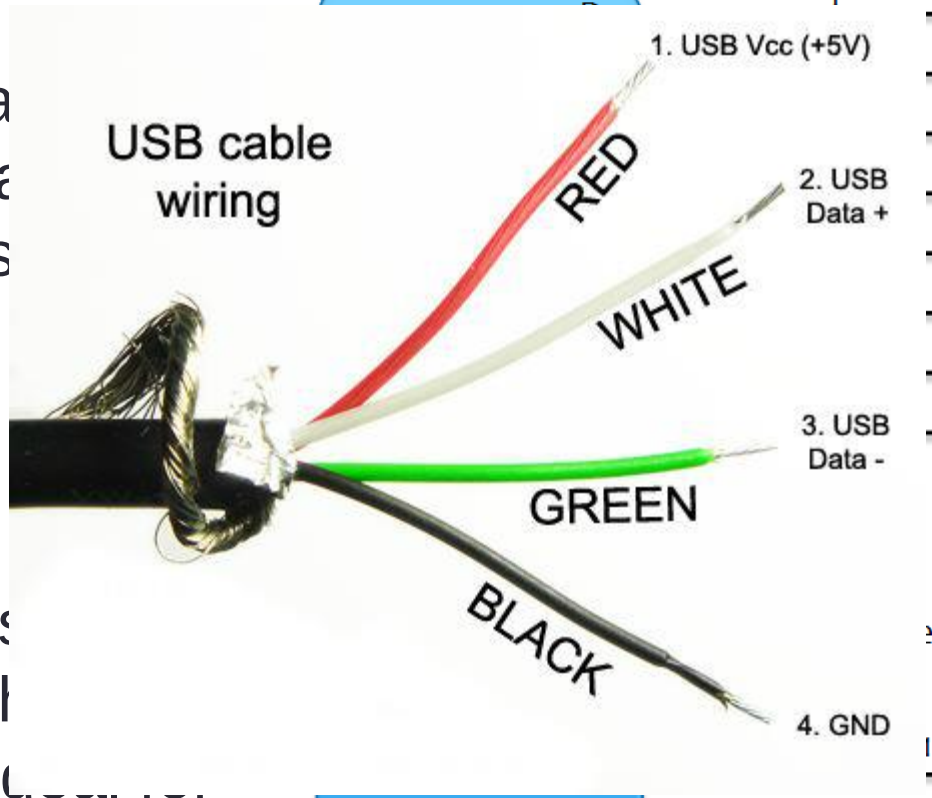
Considerations

Outline (Cont'd)

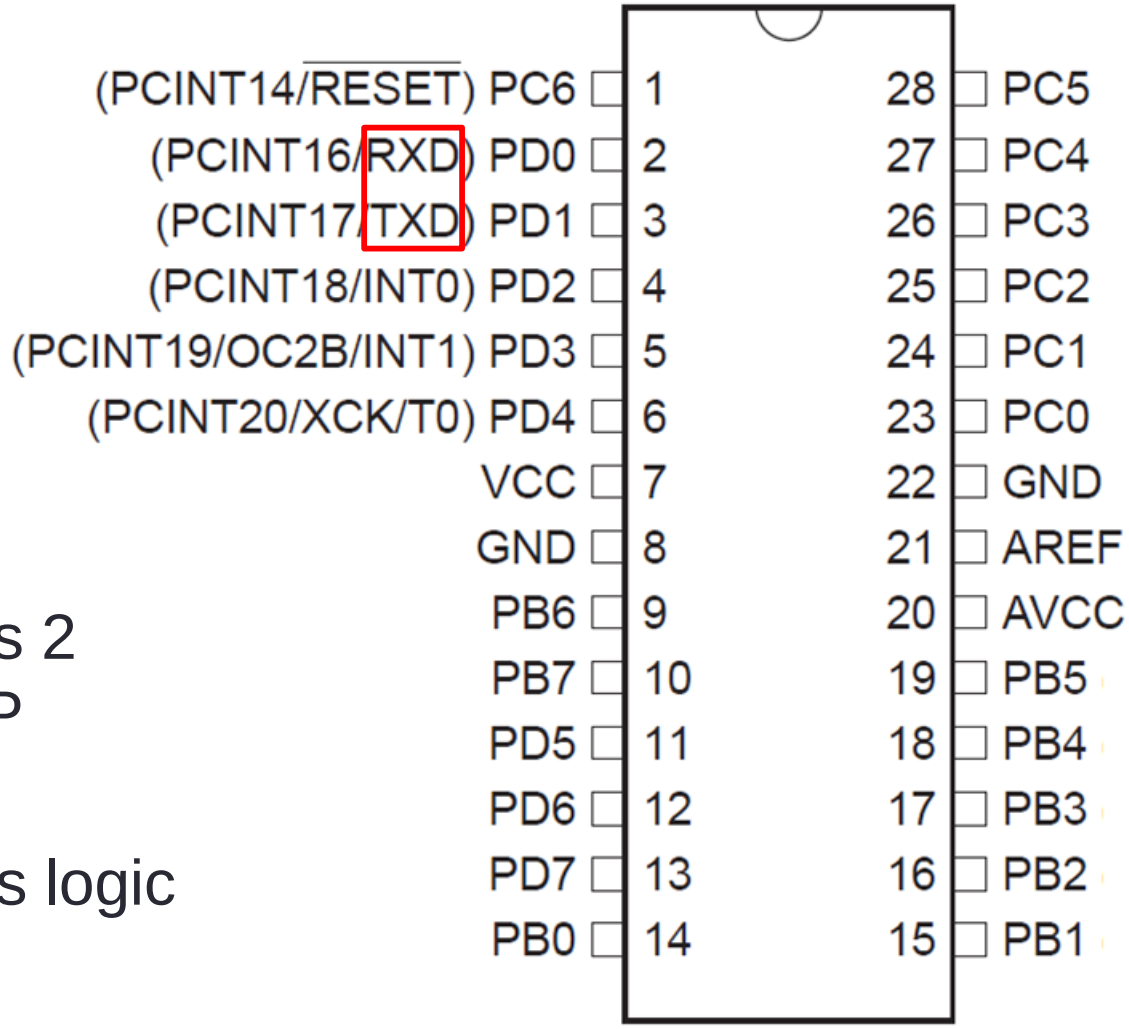
- Considerations
- Wiring
- Baudrate and data framing
- AVR USART programming
- Synchronous data transmission
  - SPI
  - I^2C
- Getting started

Parallel vs. Serial

- **Parallel** – tra
  byte of data a
  → faster, eas
- **Serial** – trans
  bit after anotl
  → cheaper, ideal for
  long distance transfer



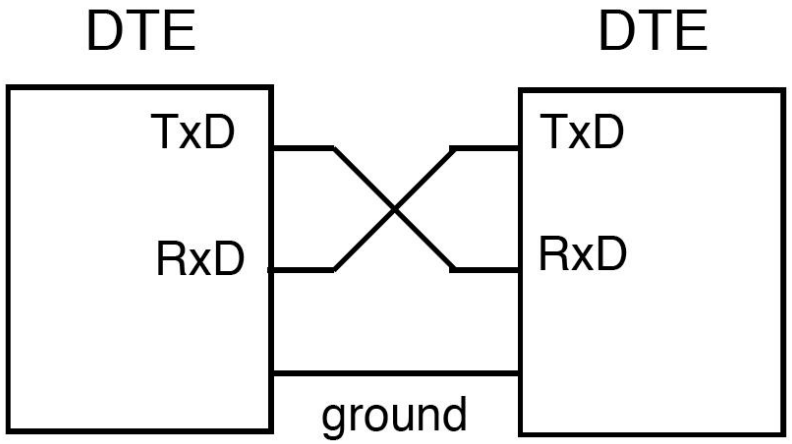
RxD and TxD Pins in ATmega328P

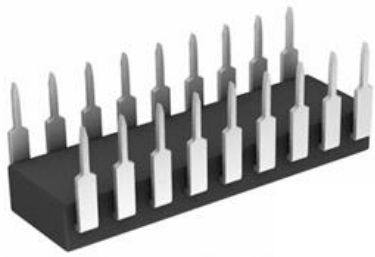


- TxD and RxD are pins 2 and 3 in ATmega328P
- These pins are TTL compatible (i.e., 5V as logic HIGH)

Null Modem Connection

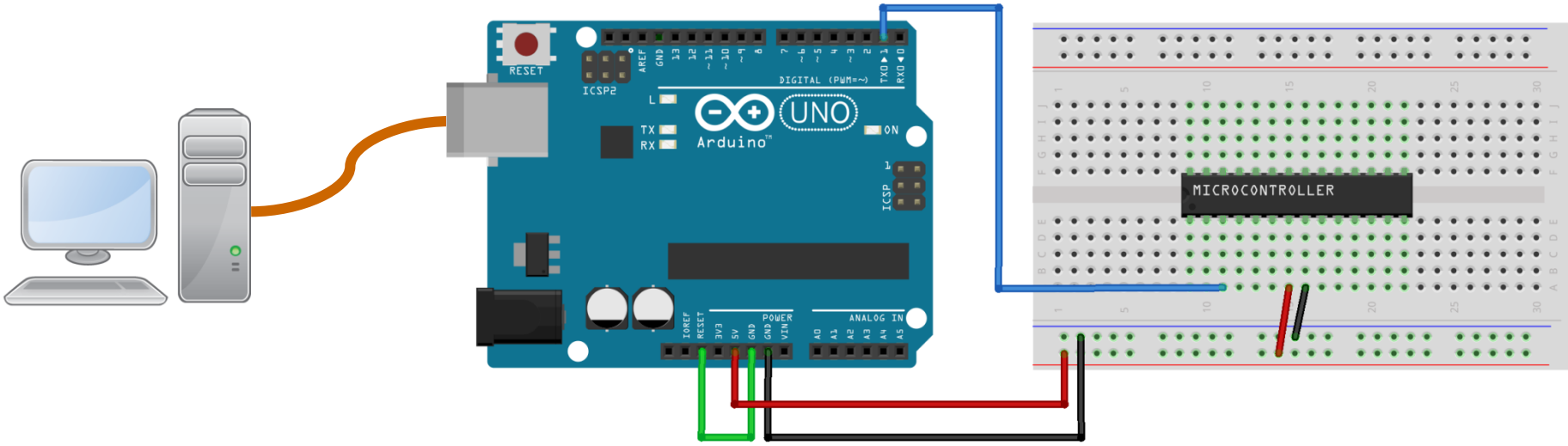
- The transmit data (TxD) and receive data (RxD) lines are crosslinked
- Used for transmitting data between 2 data terminal equipment (DTE) devices (e.g., MCU and computer)





Arduino as An RX/TX Adapter

- Wiring RESET to GND turns Arduino into an adapter
- Wiring: Arduino Tx to AVR Tx, and Arduino Rx to AVR Rx
- Display readings on Arduino serial monitor



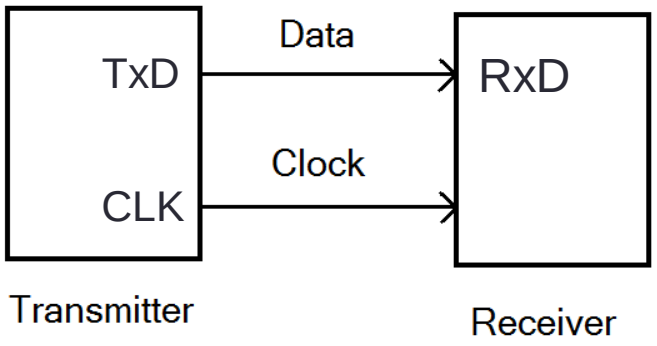
Outline (Cont'd)

- Considerations
- Wiring
- Baudrate and data framing
- AVR USART programming
- Synchronous data transmission
  - SPI
  - I^2C
- Getting started

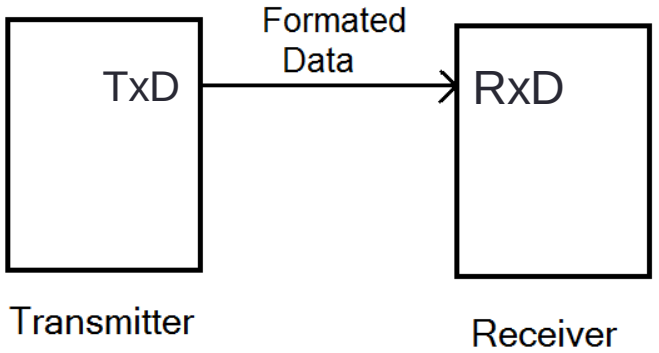
Synchronous vs. Asynchronous

- Synchronous:
  - Clock pulse is transmitted during data transmission
- Asynchronous:
  - Clock pulse is not transmitted
  - The two sides generate clock pulses at “the same” speed

Synchronous



Asynchronous

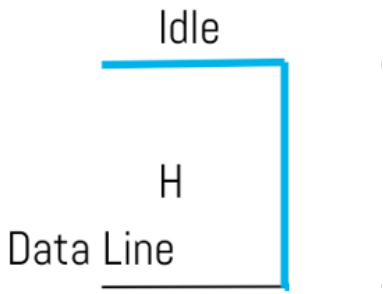


Baud Rate

- Number of symbols transferred per second
- The same as bit rate (bps) for USART
- Baud rate is NOT data rate
- Both sides of communication should use the same baud rate
- Standard baud rates: 110, 150, 300, 600, 900, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57800

Data Framing

- Framing determines what bits to read/write:
  - Start bit: a logic LOW
  - Data: 5-9 bits
  - Parity bit: optional
  - Stop bit: a logic HIGH

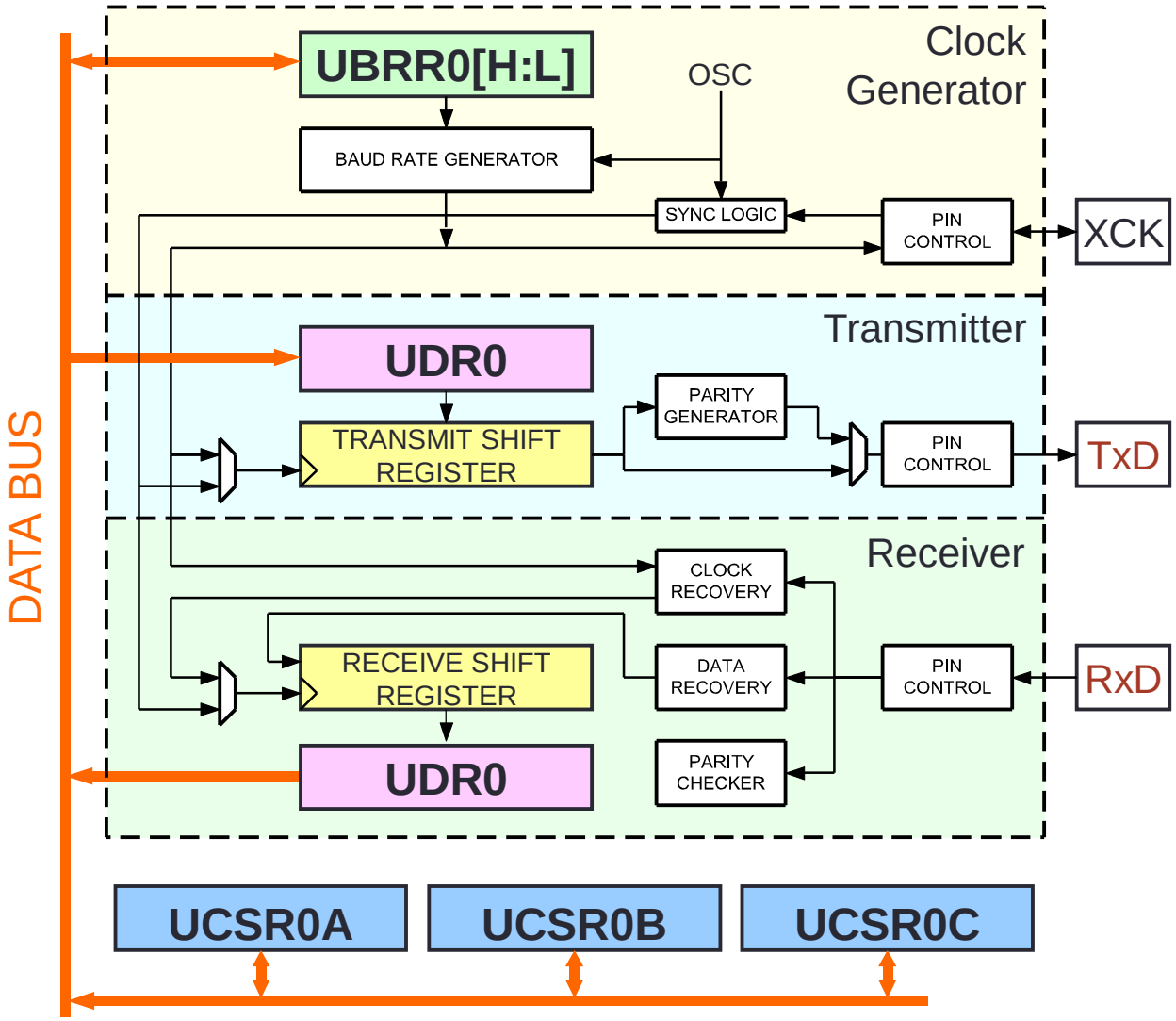


Outline (Cont'd)

- Considerations
- Wiring
- Baudrate and data framing
- AVR USART programming
- Synchronous data transmission
  - SPI
  - I^2C
- Getting started

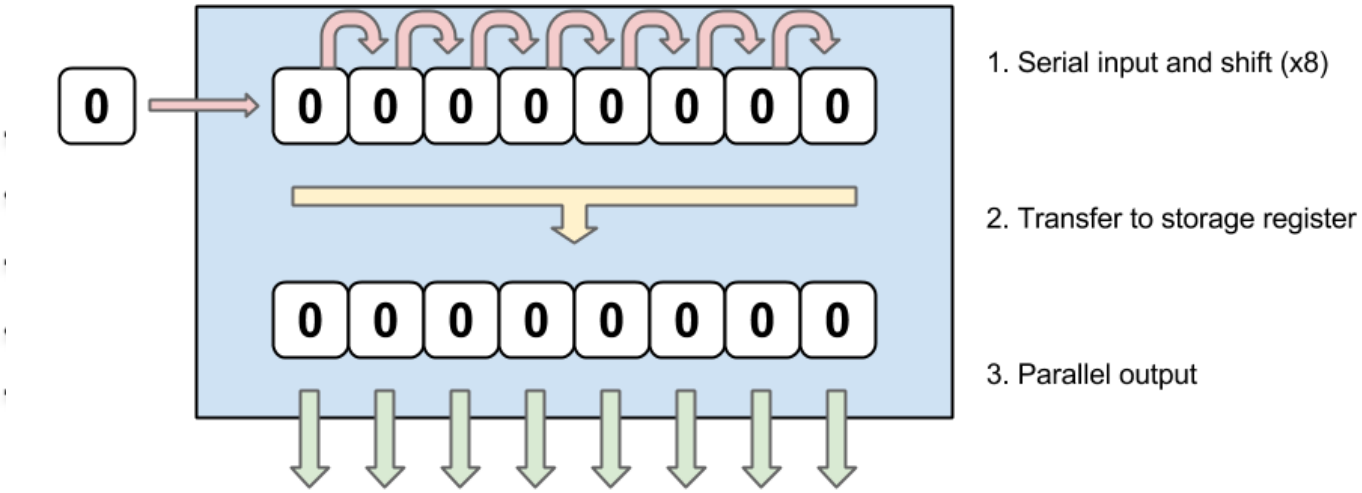
USART in AVR

- 5 registers
  - UDR0
  - UBRR0H
  - UBRR0L
  - UCSR0A
  - UCSR0B
  - UCSR0C



Shift Register: Buffer for Conversion

- A shift register is required for converting serial to/from parallel
- Data is push onto the beginning of the register, one bit at a time
- Serial-to-parallel shift register (SIPO)
- Parallel-to-serial shift register (PISO)

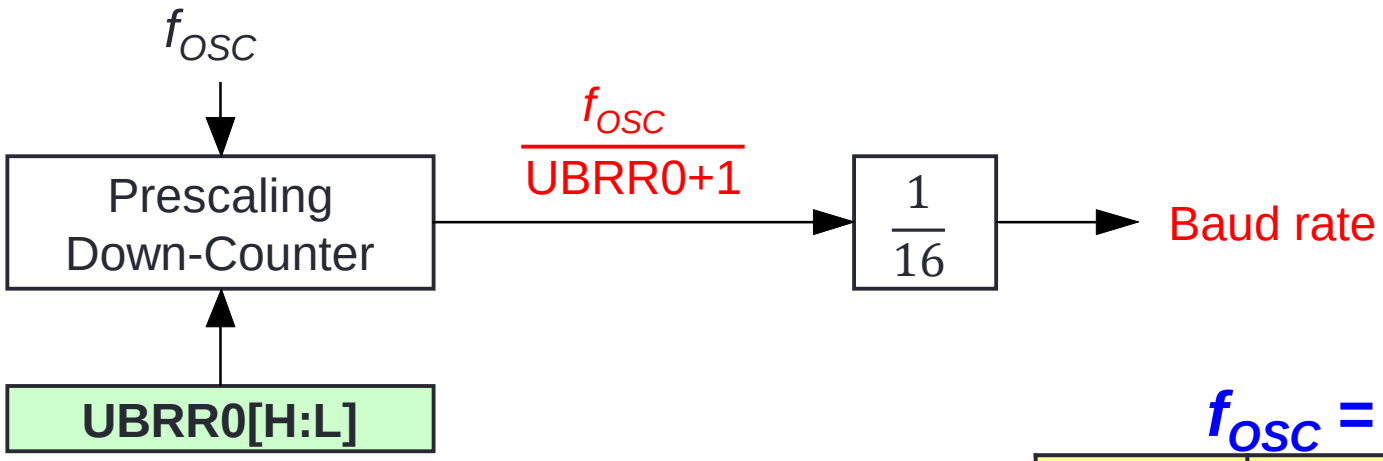


USART Registers

UDR0	• Shift (data) register
UBRR0H	• Baud rate setup
UBRR0L	
UCSR0A	• Status
UCSR0B	• Mode control
UCSR0C	• Asynchronous/synchronous, parity, stop bit, and data size

UBRR0: Baud Rate Setup

- Set the value of UBRR0 to set baud rate



UBRR0H

-	-	-	-	[11]	[10]	[9]	[8]
[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]

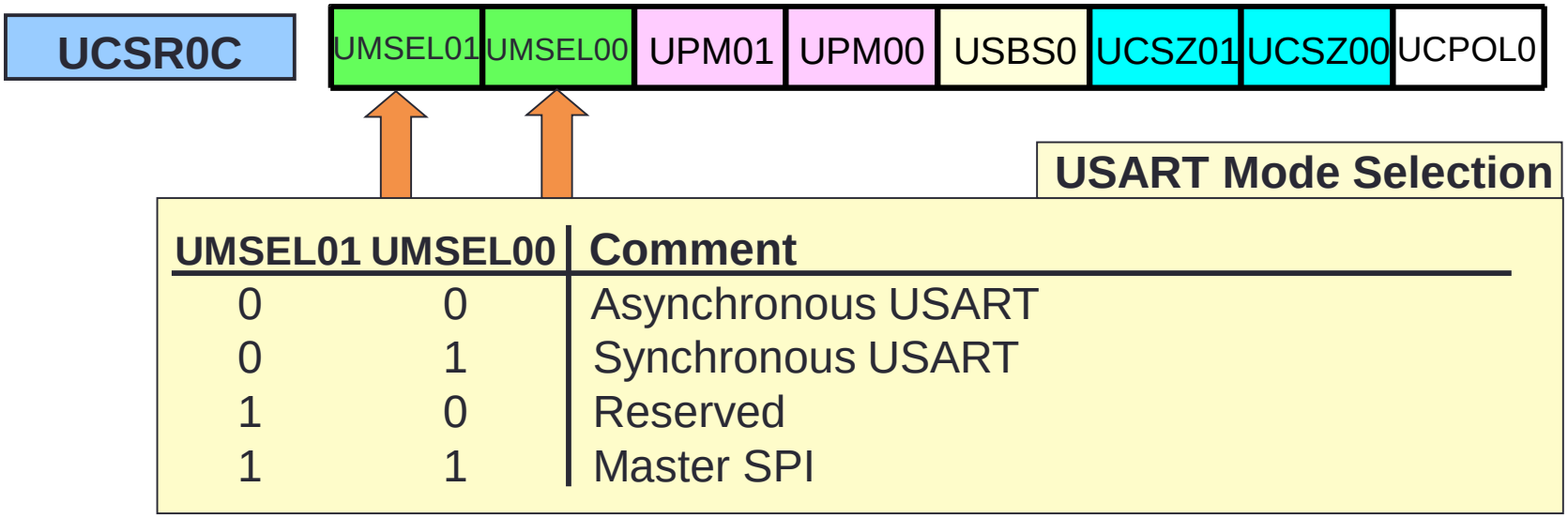
UBRR0L

f_{OSC} = 8 Mhz

Baud	UBRR0	Error (%)
2400	207	0.2
4800	103	0.2
9600	51	0.2
14.4k	34	-0.8
19.2k	25	0.2

UCSR0C: Asynchronous or Synchronous

- Asynchronous or synchronous
- Parity
- Stop bit(s) size
- Character size



UMSEL01	UMSEL00	Comment
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	Reserved
1	1	Master SPI

UCSR0C: Parity, Stop Bit, and Data Size

Number of Data Bits Selection

UCSZ01	UCSZ00	Comment
0	0	5-bit
0	1	6-bit
1	0	7-bit
1	1	8-bit

Stop Bit Selection

USBS0	Comment
0	1-bit
1	2-bit

UCSR0C

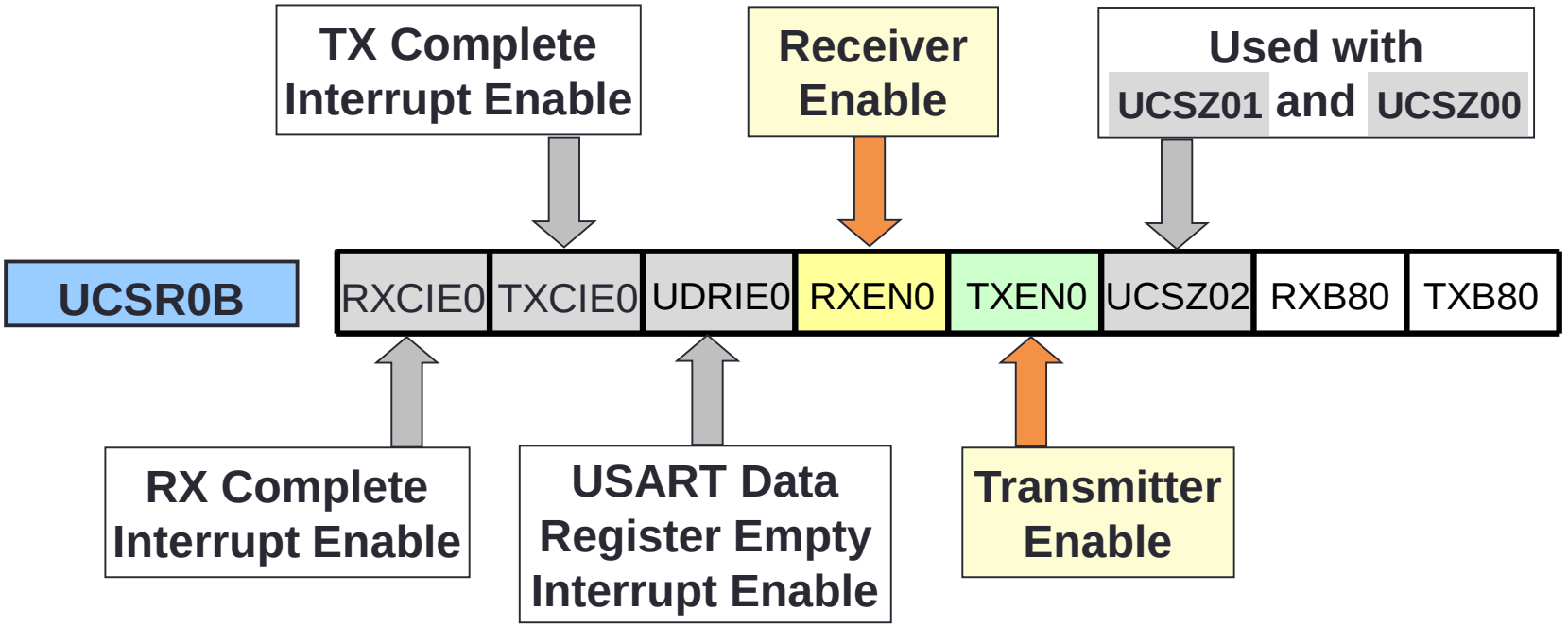
UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0

Parity Mode Selection

UPM01	UPM00	Comment
0	0	Disabled
0	1	Reserved
1	0	Enabled, even parity
1	1	Enabled, odd parity

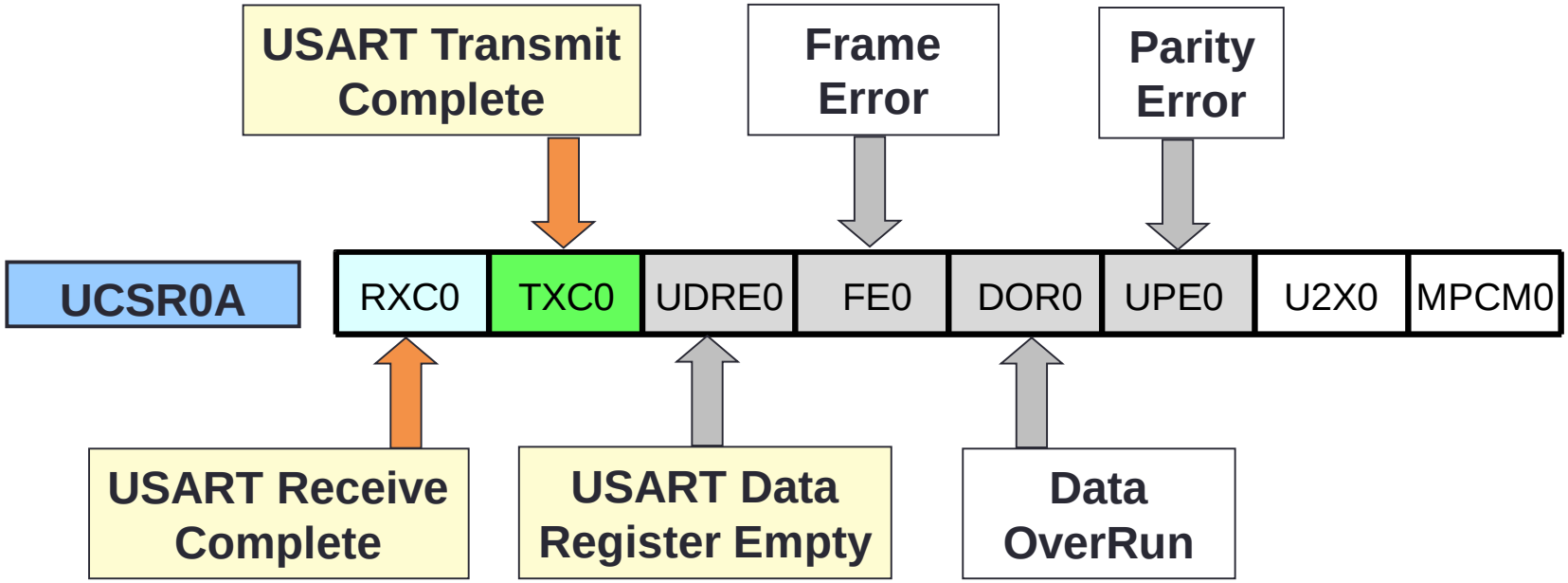
UCSR0B: USART Mode Control

- Interrupt enable
- Receiver and transmitter enable



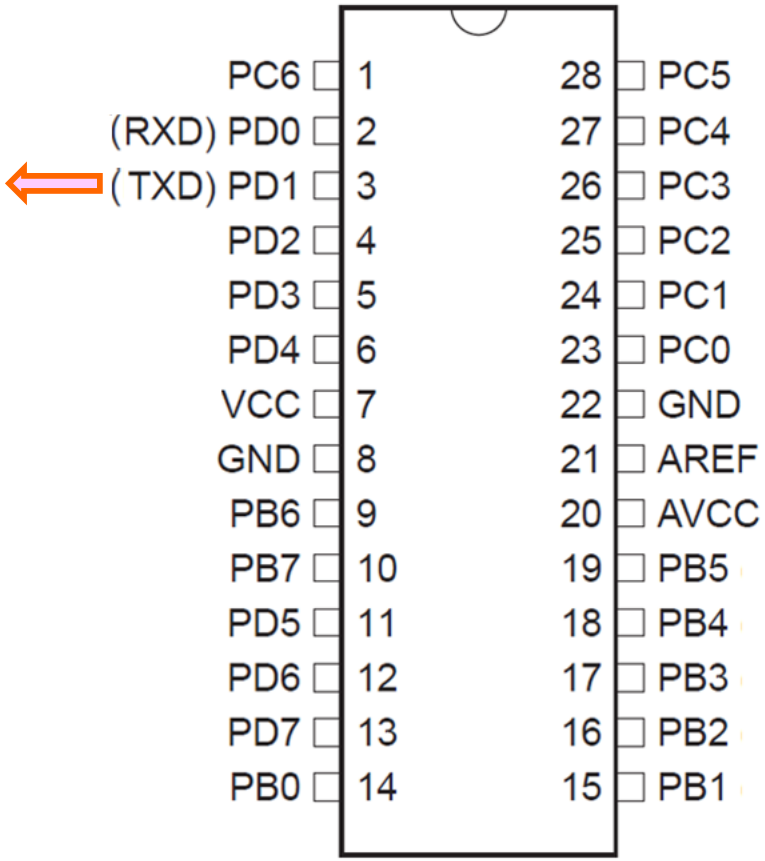
UCSR0A: USART Status

- Status flags



Example: Transmit Letter ‘A’ through ‘Z’

- Continuously transmit letter ‘A’ through ‘Z’ at ½Hz
- Baud rate 9600
- Asynchronous
- No parity
- 8 data bits
- 1 stop bit



Flowchart

- What value do we set the controller registers?

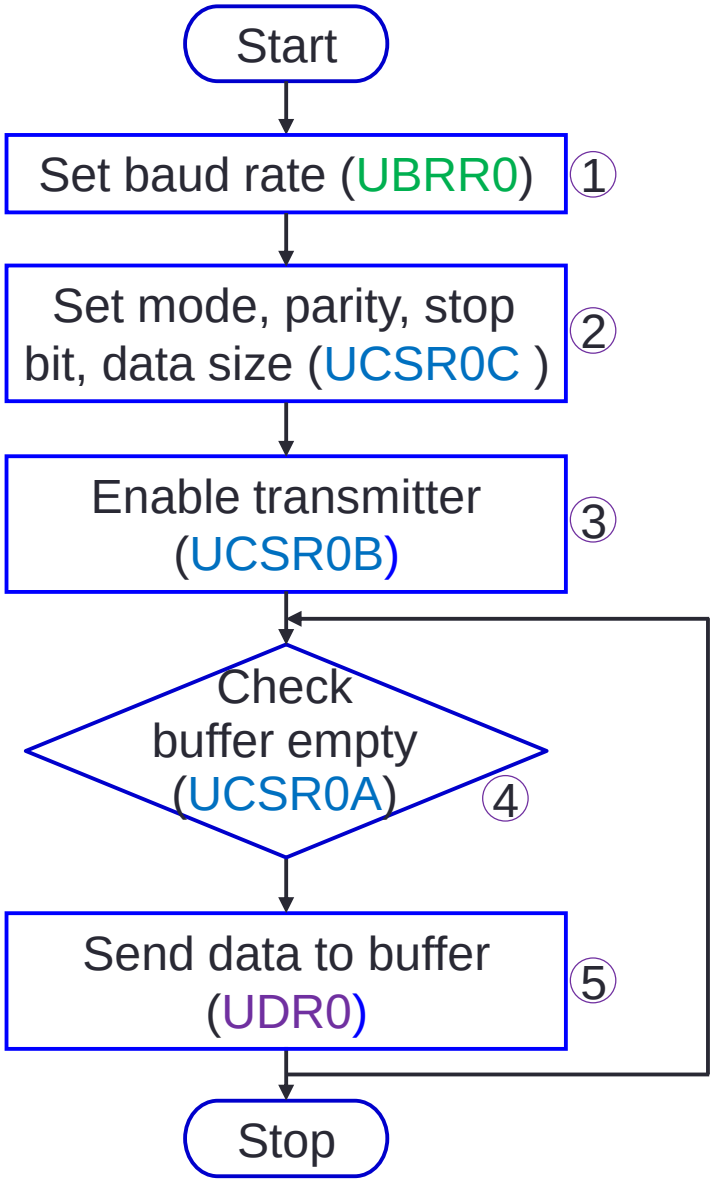
$$\frac{f_{osc}}{16(UBRR+1)} = Baud$$

UCSR0C

0	0	0	0	0	1	1	
UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0

RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
				1			

UCSR0B



Transmit Letter 'A' through 'Z'

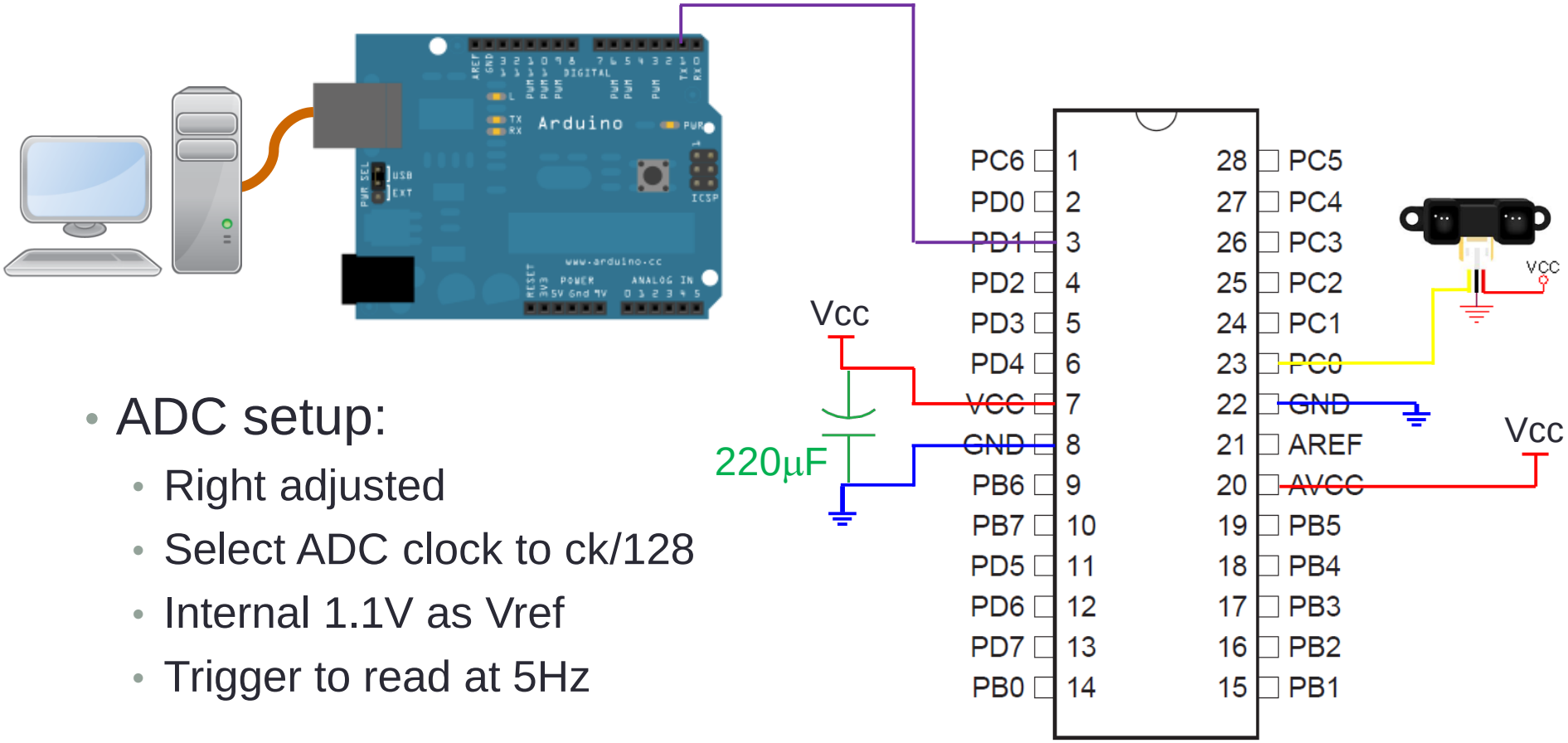
```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    CLKPR=0b10000000;
    CLKPR=0b00000000;
    unsigned int BaudR = 9600;
    unsigned int UbrrV = (F_CPU / (BaudR*16UL))-1;
    UBRR0H=(unsigned char)(UbrrV>>8);    // set Baud rate
    UBRR0L=(unsigned char)UbrrV;
    UCSR0C|=(1<<UCSZ01)|(1<<UCSZ00);     // normal mode, int clk
    UCSR0B|=(1<<TXEN0);                  // enable transmit
    while (1){
        for (char i='A'; i<='Z'; i++){
            while(!(UCSR0A&(1<<UDRE0)));// wait for empty
            UDR0=i;
            while(!(UCSR0A&(1<<UDRE0)));// wait for empty
            UDR0='\n';
            _delay_ms(500);
        }
    }
}
```

Hello World

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
void USART_putstring(char* StringPtr);

int main(void)
{
    unsigned int BaudR=9600;
    unsigned int ubrr=(F_CPU / (BaudR*16UL))-1;
    char String[]="Hello world!!\n";
    CLKPR=0b10000000;
    CLKPR=0b00000000;
    UBRR0H=(unsigned char)(ubrr>>8);
    UBRR0L=(unsigned char)ubrr;
    UCSR0B=(1<<TXEN0);
    UCSR0C=(1<<UCSZ01)|(1<<UCSZ00);

    while (1){
        USART_putstring(String);
        _delay_ms(1000);
    }
}

void USART_putstring(char* StringPtr)
{
    while(*StringPtr != 0x00){
        while(!(UCSR0A & (1<<UDRE0)));
        UDR0 = *StringPtr;
        StringPtr++;
    }
}
```

Example: Read DMS Sensor

- Read from ADC0 (PC0) and display the result on computer screen



- ADC setup:
  - Right adjusted
  - Select ADC clock to ck/128
  - Internal 1.1V as Vref
  - Trigger to read at 5Hz

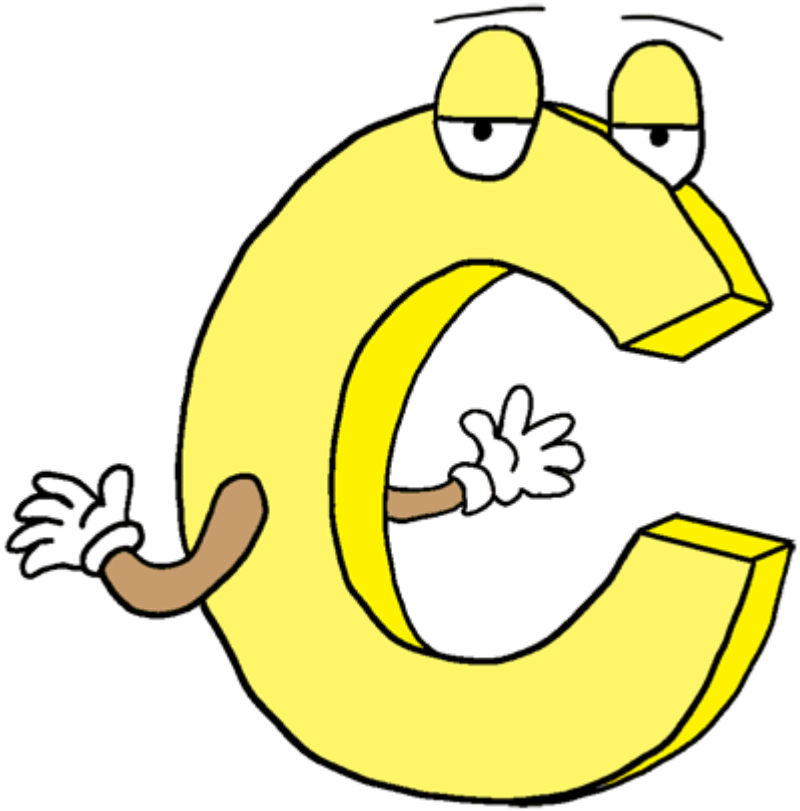
Display DMS Readings on PC Screen

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>
#include <string.h>

uint16_t ADCRead(const int);
void USART_putstring(char* StringPtr);

int main(void){
    CLKPR=0b10000000;
    CLKPR=0b00000000;
    DDRC = 0;
    ADCSRA |= (1<<ADEN);
    unsigned int BaudR = 9600;
    unsigned int ubrr = (F_CPU / (BaudR*16UL))-1;
    UBRR0H = (unsigned char)(ubrr>>8);
    UBRR0L = (unsigned char)ubrr;
    UCSR0C |= (1<<UCSZ01)|(1<<UCSZ00);
    UCSR0B |= (1<<TXEN0);

    while(1){
        float sumVal = 0;
        char Buffer[8];
        for(int i = 0; i < 10; i++)
        sumVal += (float)ADCRead(0);
        sumVal /= 10; //mean of 10 readings
        char *intStr = itoa((int)sumVal, Buffer, 10);
        strcat(intStr, "\n");
        USART_putstring(intStr);
        _delay_ms(500);
    }
}
```

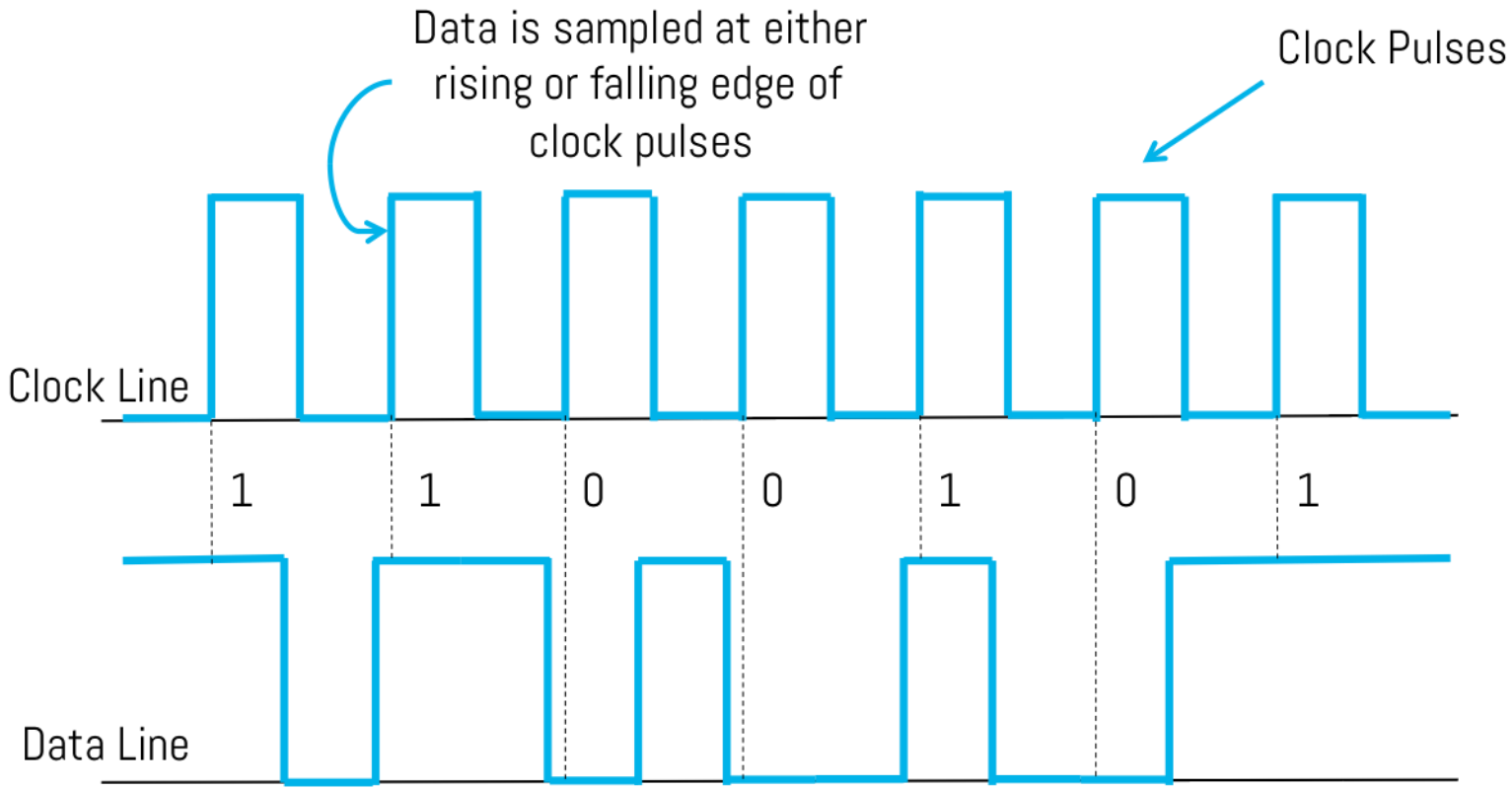
```
uint16_t ADCRead(const int channel) {
    ADMUX = 0b01000000;
    ADMUX |= channel;
    ADCSRA |= (1<<ADSC) | (1<<ADIF);
    while ( (ADCSRA & (1<<ADIF)) == 0);
    ADCSRA &= ~(1<<ADSC);
    return ADC;
}

void USART_putstring(char* StringPtr){
    while(*StringPtr != 0x00){
        while(!(UCSR0A & (1<<UDRE0)));
        UDR0 = *StringPtr;
        StringPtr++;
    }
}
```

Outline (Cont'd)

- Considerations
- Wiring
- Baudrate and data framing
- AVR USART programming
- Synchronous data transmission
  - SPI
  - I^2C
- Getting started

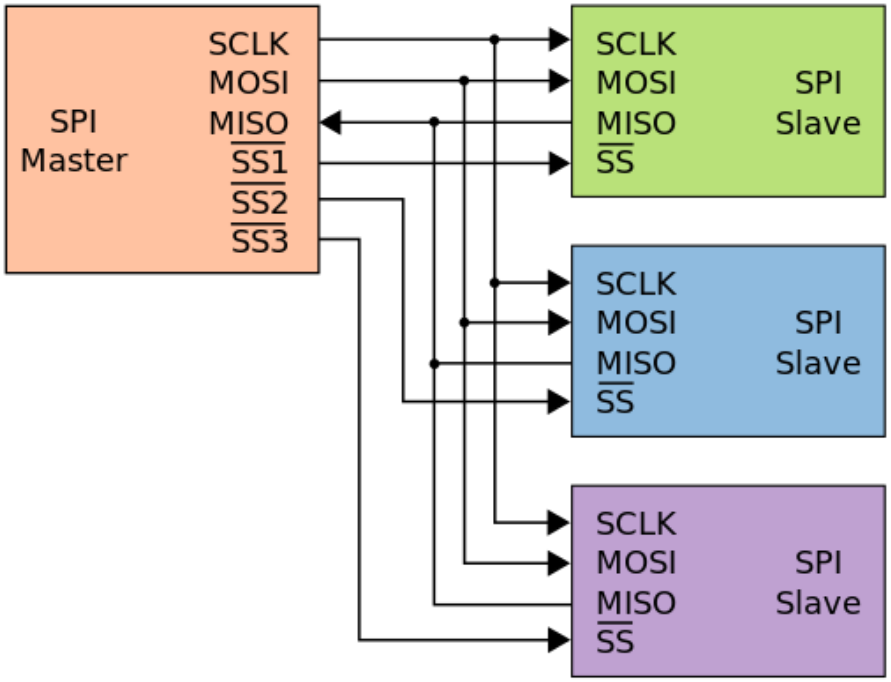


Synchronous Data Transmission



Serial Peripheral Interface (SPI)

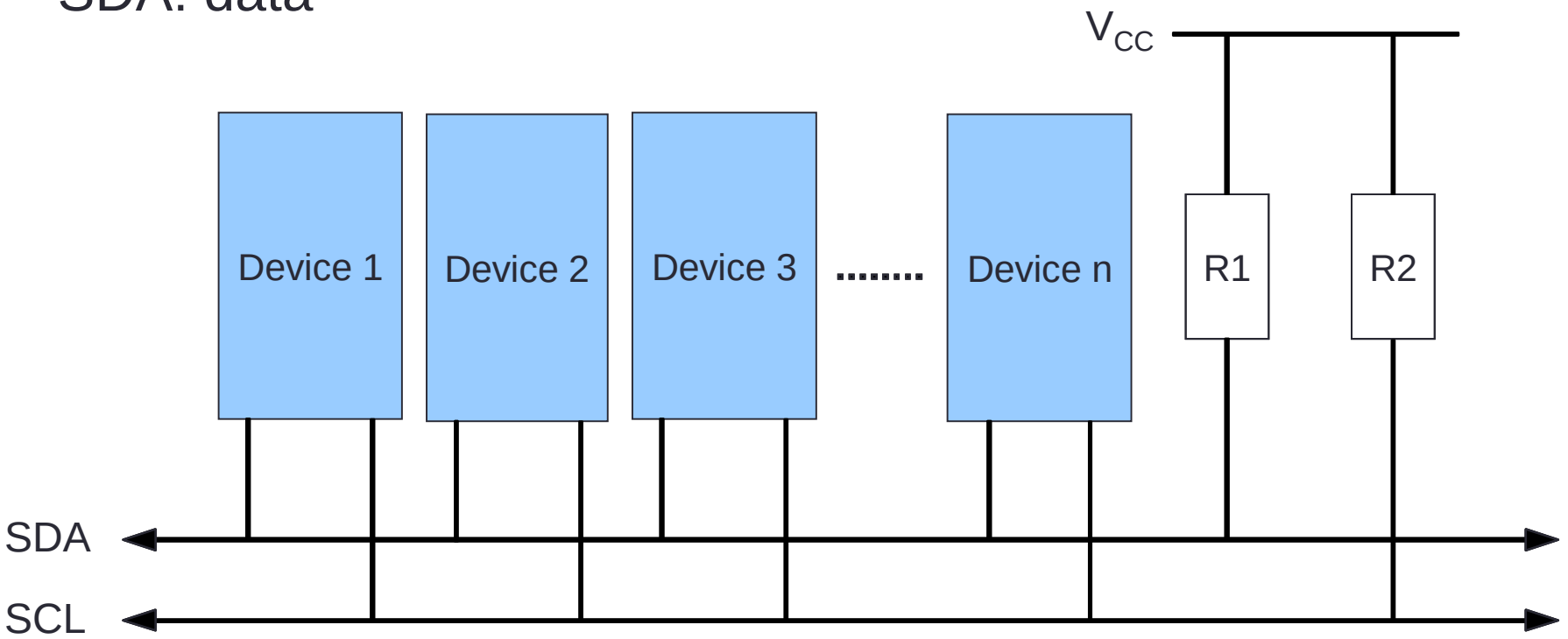
- Synchronous serial data link
- Devices communicante in master/slave mode



Pin	Description
SCLK	Serial Clock
MOSI	Master Output, Slave Input
MISO	Master Input, Slave Output
SS	Slave Select

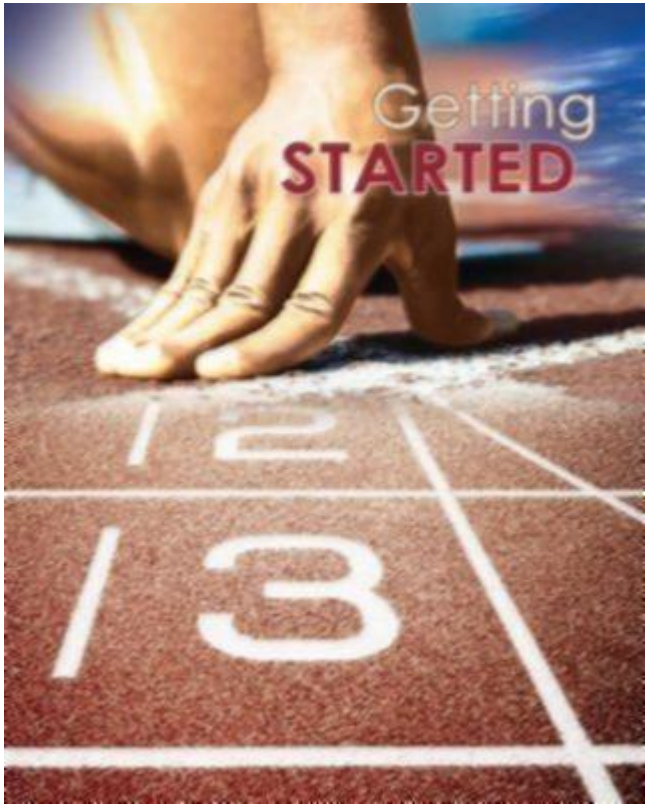
I^2C

- Up to 127 devices – combining address and data packets into a transmission
- SCL: clock
- SDA: data



Outline (Cont'd)

- Considerations
- Wiring
- Baudrate and data framing
- AVR USART programming
- Synchronous data transmission
  - SPI
  - I^2C
- Getting started



Reference

- ATmega328P data sheet
- AVR 8-bit instruction set
- Atmel AVR307: Half Duplex UART Using the USI Module
- M. A. Mazidi, S. Naimi, and S. Naimi, *The AVR Microcontroller and Embedded Systems: Using Assembly and C*, Prentice Hall, 2010