

Design with Microprocessors

**Year III Computer Science
1-st Semester**

Lecture 6: Serial data transfer

Serial Interfaces on AVR

Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART)

- Synchronous and asynchronous data communication
- Different (programmable) baud-rates
- 5-9 bits data packages (with or without parity bit)
- Errors detection
- Interrupt support for transmission control

Serial Peripheral Interface (SPI)

- Synchronous data communication
- Full duplex
- Master or Slave configuration
- Variable transfer rates

Two Wire Serial Interface (TWI)

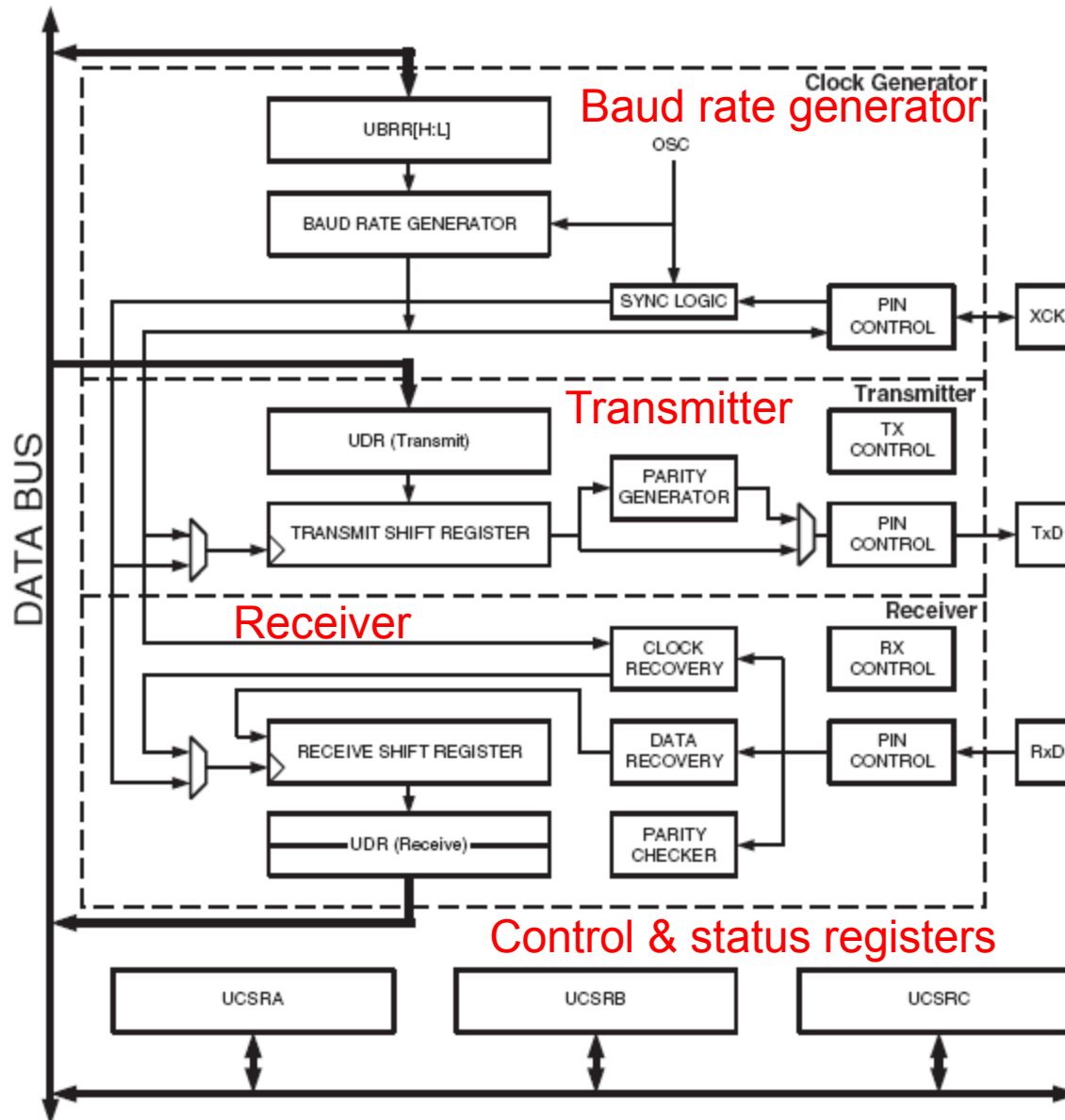
- Clock & data
- I2C protocol (master/slave, 7 bit addresses)
- Multiple masters arbitration
- Programmable slave address

USAGE: $\mu\text{C} \leftrightarrow \text{PC}$ or interboard communication, for interfacing devices & sensors etc.

USART

- **Full Duplex** Operation (Independent Serial Receive and Transmit Registers)
- **Asynchronous** or **Synchronous** Operation
- **Master or Slave** Clocked **Synchronous** Operation
- High Resolution Baud Rate Generator
- Supports Serial Frames with **5, 6, 7, 8, or 9 Data Bits** and **1 or 2 Stop Bits**
- Odd or Even Parity Generation and **Parity Check** Supported by Hardware
- Data OverRun Detection
- Framing Error Detection
- Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
- **Three Separate Interrupts** on TX Complete, TX Data Register Empty and RX Complete
- Multi-processor Communication Mode
- Double Speed Asynchronous Communication Mode

USART (0 & 1) on ATmega



UART

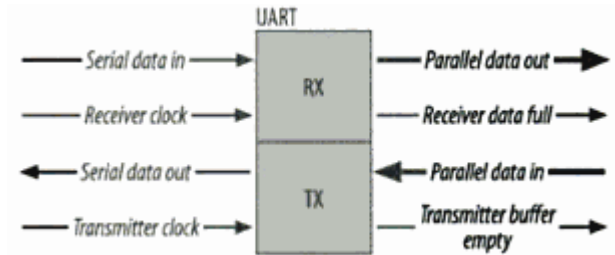
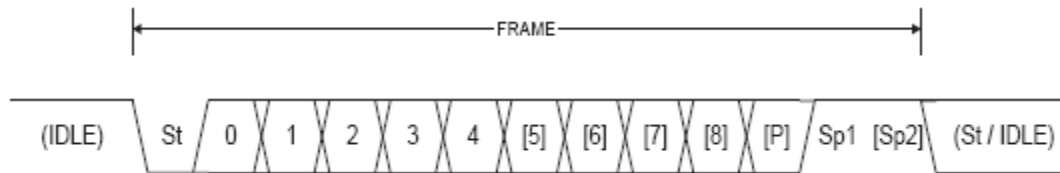
Universal Asynchronous Receiver Transmitter

2 signals:

Rx – serial data reception (input)

Tx – serial data sending (output)

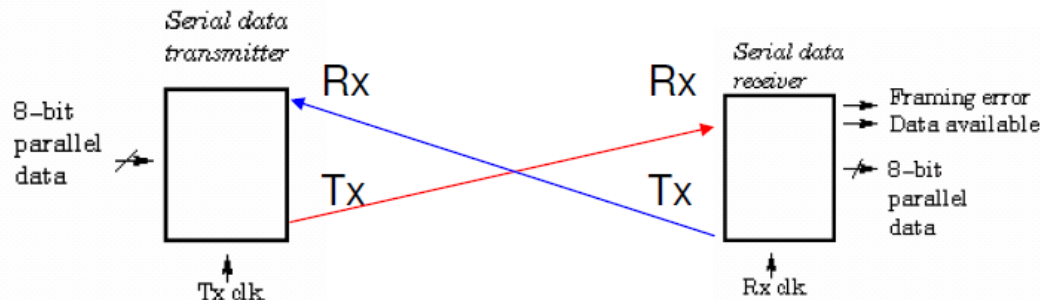
Data format (frames):



Asynchronous – time interval between frames is undefined. The receiver detects when a frame starts (Start bit) and when ends (End bit(s)).

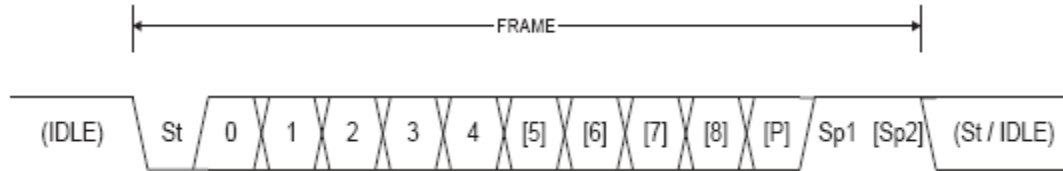
➤ Time interval between consecutive bits (frequency / baud rate) is fixed and programmed in advance ($T_{x_{clk}} \cong R_{x_{clk}}$)

➤ Full duplex communication: each side can initiate data sending



UART

Frame format



- St** - Start bit, low (0)
- (n)** - Data bits (0 to 8)
- P** - Parity (odd / even)
- Sp** - Stop bit, high (1)
- IDLE** - no data, high (1)

UART frame structure - 30 combinations:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no, even or odd parity bit
- 1 or 2 stop bits

Frame format is programmed by bits **UCSZn2:0**, **UPMn1:0** & **USBSn** in control & status registers **UCSRnB** & **UCSRnC**

- **Bit 5:4 – UPMn1:0: Parity Mode**
 - 0 0 Disabled
 - 0 1 Reserved
 - 1 0 Enabled, Even Parity
 - 1 1 Enabled, Odd Parity
- **Bit 3 – USBSn: Stop Bit Select**
 - 0 1-bit
 - 1 2-bit

Bit	7	6	5	4	3	2	1	0	
	RXCIE _n	TXCIE _n	UDRIE _n	RXEN _n	TXEN _n	UCSZn2	RXB8 _n	TXB8 _n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	-	UMSEL _n	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOL _n	UCSRnC
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	1	1	0	

UCSZn2 & UCSZn1:0 :
Character Size

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

UART

Baud rate settings

UBRRnL and UBRRnH – USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	-	-	-	-	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

UCSRnA - Bit 1 (U2Xn): Double the USART Transmission Speed
U2Xn ← 1: Reduces the division factor from 16 to 8

Table 19-12. Examples of UBRRn Settings for $f_{osc} = 16.0000$ MHz

Baud Rate (bps)	$f_{osc} = 16.0000$ MHz			
	U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max. ⁽¹⁾	1 Mbps		2 Mbps	

Table 19-1. Equations for Calculating Baud Rate Register Setting

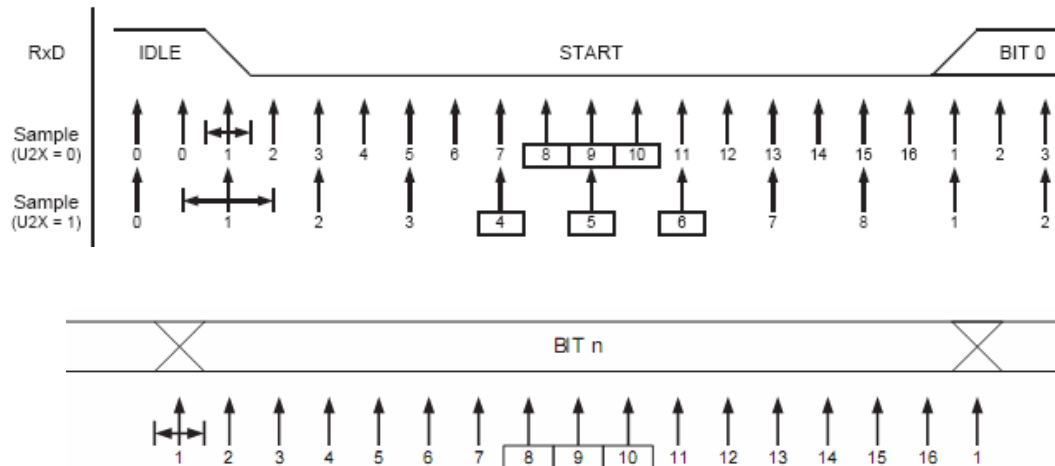
Operating Mode	Equation for Calculating Baud Rate ⁽¹⁾	Equation for Calculating UBRRn Value
Asynchronous Normal mode (U2Xn = 0)	$BAUD = \frac{f_{osc}}{16(UBRRn + 1)}$	$UBRRn = \frac{f_{osc}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$BAUD = \frac{f_{osc}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{osc}}{8BAUD} - 1$

UART

Data reception (for one frame)

1. Data transition from High (idle) '1' to low '0' on Rx is detected
2. Perform a δt delay ($2 * \delta t = 1/BAUD$) \cong middle of the time interval for the start bit. If Rx = '0', initiate frame reception sequence. Otherwise: noise
3. Check the middle of the time interval for the next bits off the frame (data, parity, stop) and re-assemble the frame.
4. If the value detected for the stop bits position is '0' \Rightarrow **framing error** (eroare de impachetare).
5. If the parity computed at destination \neq bit P \Rightarrow **parity error** (eroare de paritate)

Sampling frequency $[f_{OSC}/(UBBR+1)]$ is 8 or 16 times the baud rate for increasing the robustness



UART

UCSRnA – USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0									
	<table border="1"><tr><td>RXCn</td><td>TXCn</td><td>UDREN</td><td>FEn</td><td>DORn</td><td>UPEn</td><td>U2Xn</td><td>MPCMn</td></tr></table>								RXCn	TXCn	UDREN	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
RXCn	TXCn	UDREN	FEn	DORn	UPEn	U2Xn	MPCMn										
Read/Write	R	R/W	R	R	R	R	R/W	R/W									
Initial Value	0	0	1	0	0	0	0	0									

- **Bit 7 – RXCn: USART Receive Complete**

RXCn \leftarrow 1, data in receive buffer

RXCn \leftarrow 0, receive buffer empty

RXCn can trigger Receive Complete interrupt (+RXCIEn bit)

- **Bit 6 – TXCn: USART Transmit Complete**

TXCn \leftarrow 1, if Transmit Shift Register empty & UDR has now new data

TXCn \leftarrow 0, if a „Transmit Complete” interrupt is executed or by writing 1 to UCSRnA(TXCn)

TXCn can trigger a Transmit Complete interrupt (+TXCIEn bit)

- **Bit 5 – UDREN: USART Data Register Empty**

UDREN \leftarrow 1, transmit buffer empty

UDREN can trigger a “Register Empty” interrupt (+UDRIEn bit).

- **Bit 4 – FEn: Frame Error**

FEn \leftarrow 1, framing error

- **Bit 3 – DORn: Data OverRun**

DORn \leftarrow 1, receive buffer full (2 char) and a new Start bit detected

- **Bit 2 – UPEn: USART Parity Error**

UPEn \leftarrow 1, parity error at reception (if parity enabled)

UART

Parity computation

- The parity bit is calculated by doing an **exclusive-or** of all the data bits.
- If odd parity is used, the result of the exclusive or is inverted.
- The relation between the parity bit and data bits is as follows:

$$P_{even} = d_{n-1} \oplus \dots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \dots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

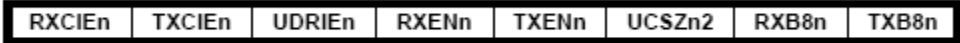
P_{even} Parity bit using even parity

P_{odd} Parity bit using odd parity

d_n Data bit n of the character

UART

UCSRnB – USART Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
									UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – RXCIEn: RX Complete Interrupt Enable**

RXCIEn \leftarrow 1, validate RXCn interrupt request (+ Global Interrupt Flag in SREG)

- **Bit 6 – TXCIEn: TX Complete Interrupt Enable**

TXCIEn \leftarrow 1, validate TXCn interrupt request (+ Global Interrupt Flag in SREG)

- **Bit 5 – UDRIEn: USART Data Register Empty Interrupt Enable**

UDRIEn \leftarrow 1, validate UDREn interrupt request (+ Global Interrupt Flag in SREG)

- **Bit 4 – RXENn: Receiver Enable**

RXENn \leftarrow 1, validate reception

RXENn \leftarrow 0, invalidate reception

- **Bit 3 – TXENn: Transmitter Enable**

TXENn \leftarrow 1, validate transmission

TXENn \leftarrow 0, invalidate transmission (after flushing of Transmit Shift Register & Transmit Buffer Register)

- **Bit 1 – RXB8n: Receive Data Bit 8**

RXB8n 9-th data bit received (frame with 9 data bits)

Read before bits (7:0) from UDRn

- **Bit 0 – TXB8n: Transmit Data Bit 8**

TXB8n - 9-th data bit transmitted (frame cu with 9 data bits)

Written before bits (7:0) from UDRn

UART

UCSRnC – USART Control and Status Register C

Bit	7	6	5	4	3	2	1	0	
	-	UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
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	1	1	0	

- Bit 7 – **Reserved Bit**

UCSRnC \leftarrow 0

- Bit 6 – **UMSELn: USART Mode Select**

UMSELn \leftarrow 0/1 operation mode : asynchronous / synchronous

- Bit 0 – **UCPOLn: Clock Polarity**

UCPOLn \leftarrow 0 for asynchronous mode

Significance only for the synchronous mode

UDRn – USART I/O Data Register (Transmit Data Buffer Register TXB & Receive Data Buffer (RXB))

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDnR (Read)
	TXB[7:0]								UDnR (Write)
Read/Write	R/W	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	0

- **Same I/O address** (UDRn)
- Writing UDRn \rightarrow writing TXBn \Rightarrow triggers data sending if the shift register is free
- Writing is valid only if UDREN=1(empty) in UCSRnA (otherwise ignored)
- Reading UDRn \rightarrow reading RXBn

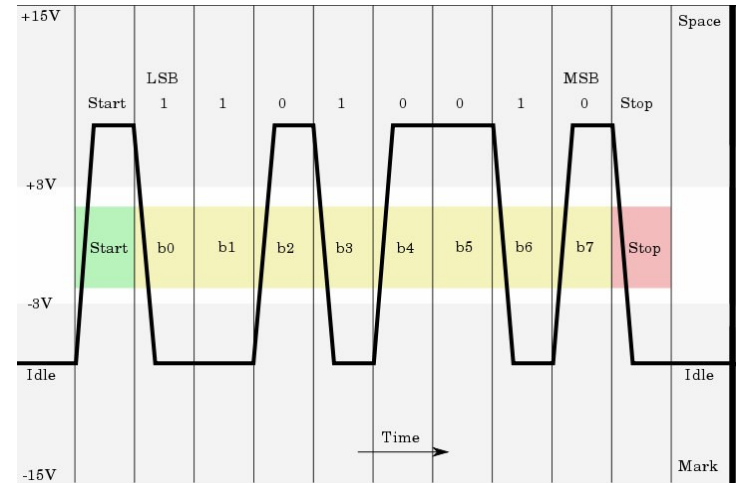
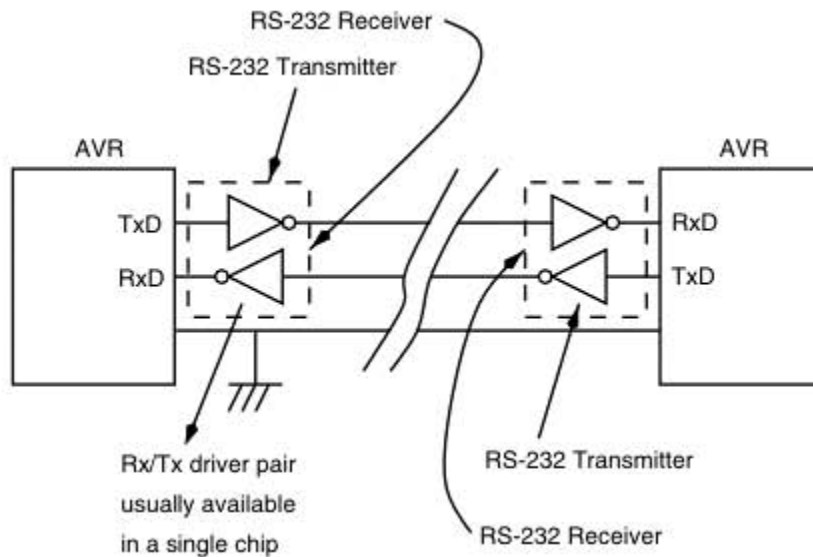
USART & RS232

Signal level adaptation to RS232 standard:

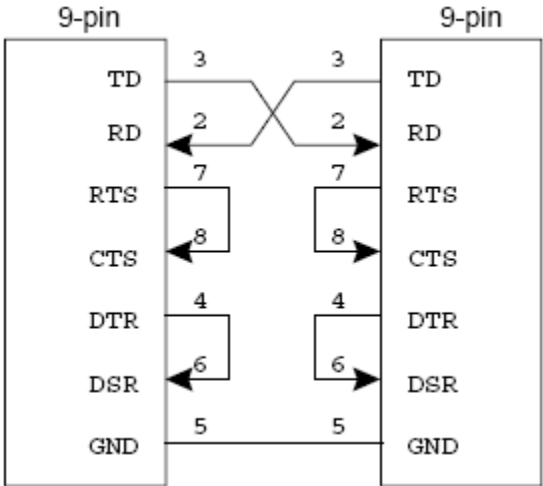
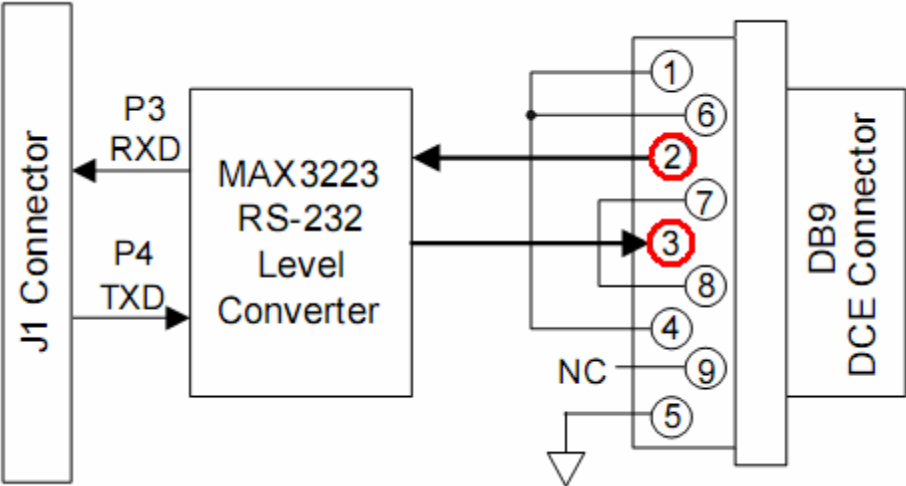
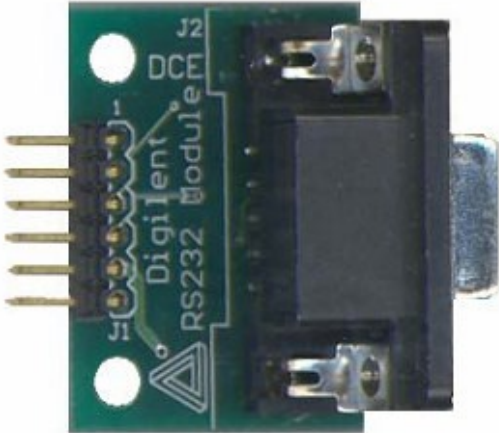
RS232 logic '1' -5... -15 V

RS232 logic '0' +5...+15 V

(For long wire transmission \Rightarrow robustness + optical isolation)



Pmod RS232



USART on ATmega

Example: ATmega + Pmod RS232 ↔ PC communication through a serial cable

1. USART/PC configuration

Baud: 9600

Frame size: 8 data bits

Stop bits: 2

Parity: none

$$UBRR_n = \frac{f_{osc}}{16BAUD} - 1$$

$$f_{osc} = 16.000.000$$

$$UBRR_n = 103$$

2. Wait for character receiving

- check RXCn (UCSRnA bit 7): wait until becomes 1

3. Read received character from UDRn

4. Write character to be sent in UDRn

5. Wait character transmission

- check TXCn (UCSRnA bit 6), wait until becomes 1

6. Jump to step 2

USART on ATmega

Example: ATmega (USART1) + Pmod RS232 ↔ PC communication through a serial cable

```
ldi r16, 0b00011000 ; activates Rx && Tx
```

```
sts UCSR1B,r16
```

```
ldi r16, 0b00001110 ; frame size 8 bits, no parity, 2 stop bits
```

```
sts UCSR1C,r16
```

```
ldi r16, 103 ; lower 8 bits of BAUD
```

```
ldi r17, 0 ; upper bits of BAUD are 0
```

```
sts UBRR1H, r17
```

```
sts UBRR1L, r16
```

```
mainloop:
```

```
  rxloop:
```

```
    lds r20, UCSR1A
```

```
    sbrs r20, 7 ; bit 7 (RXCn) in UCSR1A = 1 ⇒ reception complete
```

```
  rjmp rxloop
```

```
lds r16, UDR1 ; read received data
```

```
sts UDR1,r16 ; write data for transmission
```

```
  txloop:
```

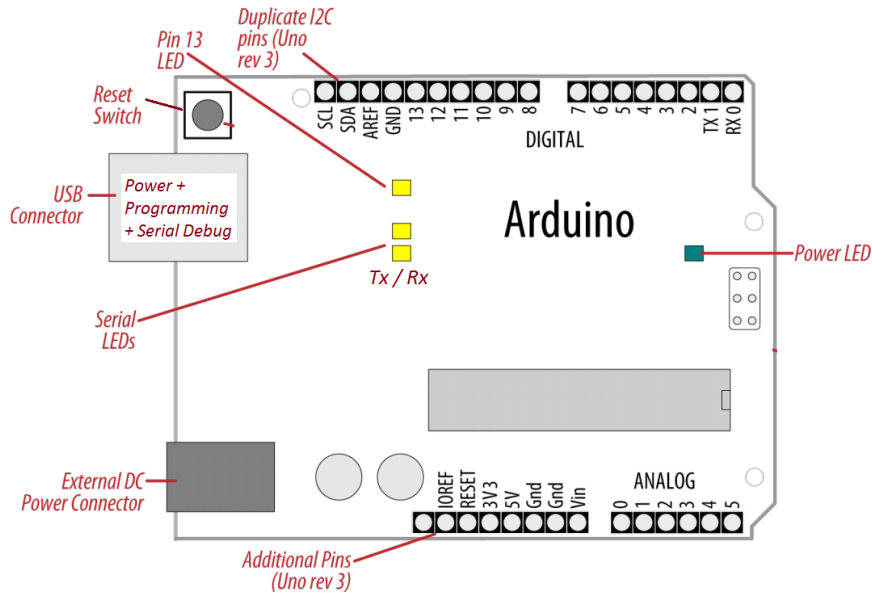
```
    lds r20, UCSR1A ; wait for transmission finish
```

```
    sbrs r20,UDRE1 ; bit 5 in UCSR1A = 1 ⇒ transmission buffer is empty  
    ; or sbrs r20, 6 ; bit 6 (TXCn) in UCSR1A = 1 ⇒ transmission complete
```

```
  rjmp txloop
```

```
rjmp mainloop
```

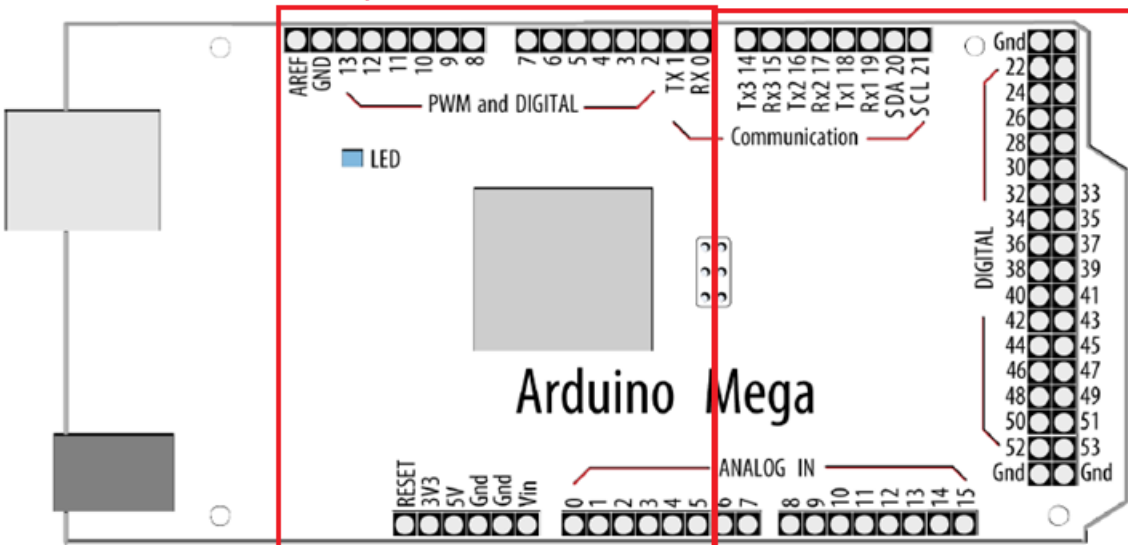

Serial Communication with Arduino



Arduino UNO (rev. 3)

- Serial: 0 (RX) and 1 (TX);
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).
- TWI (I2C): A4 or SDA pin and A5 or SCL pin

Pin Layout identical with UNO



Pin Layout specific to MEGA

Arduino MEGA (rev. 3)

- Serial : 0 (RX) and 1 (TX);
Serial 1: 19 (RX) and 18 (TX);
Serial 2: 17 (RX) and 16 (TX);
Serial 3: 15 (RX) and 14 (TX)
- SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS)
- TWI: 20 (SDA) and 21 (SCL)

Serial communication with Arduino

All Arduino boards have at least one **native** serial port (also known as a UART or USART): **Serial**

- μ C ↔ PC communication via the on-board USB port (USB-to-serial adapter – ATmega 16U2) – used **also for the board programming !!**
- inter-board communication using pins 0 (RX) and 1 (TX) - **not recommended**
Also is not recommended to use these pins (0 and 1) for digital I/O !!!

The Arduino MEGA has three additional serial ports: **Serial1** on pins 19 (RX) and 18 (TX), **Serial2** on pins 17 (RX) and 16 (TX), **Serial3** on pins 15 (RX) and 14 (TX).

- to **communicate with your personal** computer through these ports, you will need an **additional USB-to-serial adaptor** (they are **not connected to the Mega's USB-to-serial adaptor**).
- to **communicate with an external TTL serial device**, connect the TX pin to your device's RX pin, the RX to your device's TX pin, and the ground of your Mega to your device's ground. (**Don't connect these pins directly to an RS232 serial port which operate at +/- 12V ⇒ damage your Arduino board; unless you use a RS232 adapter as the Pmod RS232**)

Serial communication with Arduino

The built-in Arduino **Serial** library (<http://arduino.cc/en/Reference/Serial>) [1] - used for communication between the Arduino board and a computer or other TTL serial devices

Serial library methods (selection):

- **Serial.begin(speed)** – sets the baud rate (speed) and the default serial frame format (*8 data bits, no parity, one stop bit*)
- **Serial.begin(speed, config)** - sets the baud rate (speed) + customizable frame format (config)

config – ex: SERIAL_8N1 (the default), SERIAL_7E2, SERIAL_5O1 ...

- **Serial.print(val)** - prints data to the serial port as human-readable ASCII text
- **Serial.print(val, format)** – format specifies the base to use (BIN, OCT, DEC, HEX. For floating point numbers - the number of decimal places to use.
- **Serial.println** - Prints data followed (ASCII 13, or '\r') + (ASCII 10, or '\n')

Examples:

Serial.print(78) gives "78"

Serial.print(1.23456) gives "1.23"

Serial.print("Hello.") gives "Hello"

Serial.print(78, BIN) gives "1001110"

Serial.println(1.23456, 4) gives "1.2346"

Serial communication with Arduino

Arduino example:

```
void setup() {  
    Serial.begin(9600); // opens serial port, sets data rate to 9600 bps  
}
```

```
void loop() {}
```

Arduino Mega example:

```
// Arduino Mega using all four of its Serial ports  
// (Serial, Serial1, Serial2, Serial3),  
// with different baud rates:
```

```
void setup(){  
    Serial.begin(9600);  
    Serial1.begin(38400);  
    Serial2.begin(19200);  
    Serial3.begin(4800);  
  
    Serial.println("Hello Computer");  
    Serial1.println("Hello Serial 1");  
    Serial2.println("Hello Serial 2");  
    Serial3.println("Hello Serial 3");  
}
```

```
void loop() {}
```

Serial communication with Arduino

Serial library methods (selection):

- int IncomingByte **Serial.read()** - reads incoming serial data
- Int NoOfBytesSent **Serial.write(data)** – writes binary data to the serial port. Data is sent as a byte (val) or a series of bytes specified as a string (str) or as an array (buf, len)

To send the characters representing the digits of a number use the [print\(\)](#) function instead write().

- **Serial.flush()** - waits for the transmission of outgoing serial data to complete
- Int NoOfBytes **Serial.available()** - Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds up to 64 bytes)
- **serialEvent()** – user defined function called when data is available. Use Serial.read() to capture this data.
- serialEvent1(), serialEvent2(), serialEvent3() - Arduino Mega only

Serial communication with Arduino

μC ↔ PC communication : receiving Serial Data in Arduino - receive data on Arduino from a computer or another serial device and react to commands or data sent from your computer [2]

Example 1 - receives a digit (single characters 0 ... 9) and blinks the LED on pin 13 at a rate proportional to the received digit value

```
const int ledPin = 13;          // LED pin
int blinkRate=0;              // blink rate
void setup()
{
  Serial.begin(9600); // Initialize serial port to send and receive at 9600 baud
  pinMode(ledPin, OUTPUT); // set this pin as output
}
void loop() {
  if ( Serial.available())      // Check to see if at least one character is available
  {
    char ch = Serial.read();
    If( isDigit(ch) )           // is this an ascii digit between 0 and 9?
    {
      blinkRate = (ch - '0');    // ASCII value converted to numeric value
      blinkRate = blinkRate * 100; // actual rate is 100ms times received digit
    }
  }
  blink();
}
```

Serial communication with Arduino

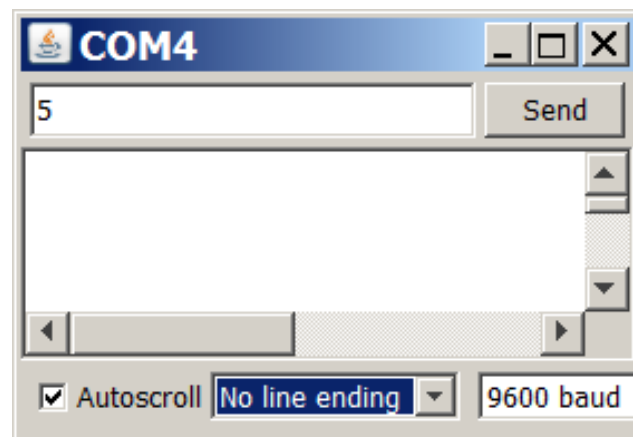
Example 1 – cont.

// blink the LED with the on and off times determined by blinkRate

```
void blink()
{
  digitalWrite(ledPin,HIGH);
  delay(blinkRate); // delay depends on blinkrate value
  digitalWrite(ledPin,LOW);
  delay(blinkRate);
}
```

To use the example (PC side):

- use the Arduino environment's [built-in serial monitor](#) (click the serial monitor button in the toolbar or <CTRL+SHIFT+M>)
- select the same baud rate used in the call to `serial.begin()`
- type a digit in the text box at the top of the Serial Monitor window
- clicking the Send button will send the character typed into the text box; if you type a digit, you should see the blink rate change



Serial communication with Arduino

Example 1 – modified using serialEvent()

```
void loop()
{
  blink();
}

void serialEvent()
{
  while(Serial.available())
  {
    char ch = Serial.read();
    // Serial.write(ch);
    if( isDigit(ch) ) // is this an ascii digit between 0 and 9?
    {
      blinkRate = (ch - '0'); // ASCII value converted to numeric value
      blinkRate = blinkRate * 100; // actual rate is 100mS times received digit
    }
  }
}
```

Homework: optimize the blink function code (without using delay).

Serial communication with Arduino

SoftwareSerial library (<http://arduino.cc/en/Reference/softwareSerial>)

- Developed to allow serial communication on other digital pins of the Arduino, using software to replicate the functionality (hence the name "SoftwareSerial") – useful especially on UNO boards
- It is possible to have multiple software serial ports with speeds up to 115200 bps. A parameter enables inverted signaling for devices which require that protocol.

Limitations

- If using multiple software serial ports, **only one can receive data at a time**.
- Not all pins on the Mega and Mega 2560 support change interrupts, so only the following can be used for **RX**: 10, 11, 12, 13, 14, 15, 50, 51, 52, 53, A8 (62), A9 (63), A10 (64), A11 (65), A12 (66), A13 (67), A14 (68), A15 (69).
- If your project requires simultaneous data flows

Example 2: Software serial multiple serial test

Receives from the hardware serial, sends to software serial.

Receives from software serial, sends to hardware serial.

The circuit:

- * RX is digital pin 10 (connect to TX of other device)
- * TX is digital pin 11 (connect to RX of other device)

Serial communication with Arduino

Example 2 – Software serial multiple serial test

```
#include <SoftwareSerial.h>
```

```
SoftwareSerial mySerial(10, 11); // RX, TX
```

```
void setup()
```

```
{  
  // Open serial communications and wait for port to open:  
  Serial.begin(4800); // uses the native / built-in serial (USART hardware device)  
  while (!Serial) {  
    ; // wait for serial port to connect. Needed for Leonardo only  
  }  
}
```

```
  Serial.println("Goodnight moon!"); // send data through the built-in serial (USART hardware device)
```

```
  // set the data rate for the SoftwareSerial port
```

```
  mySerial.begin(4800);  
  mySerial.println("Hello, world?");  
}
```

```
void loop() // run over and over
```

```
{ // exchanges data between the hardware and the software serials  
  if (mySerial.available())  
    Serial.write(mySerial.read());  
  if (Serial.available())  
    mySerial.write(Serial.read());  
}
```

SPI with Arduino

SPI library (<http://arduino.cc/en/Reference/SPI>) [3]

- **SPI.setBitOrder(order)** – bit order = LSBFIRST or MSBFIRST
- **SPI.setDataMode(mode)** – mode = SPI_MODE0 or SPI_MODE1 or SPI_MODE2 or SPI_MODE3 (set clock phase and polarity)
- **SPI.setClockDivider()** – SPI clock divider = SPI_CLOCK_DIV(2 .. 128)
- **SPI.begin()** - initialize the SPI bus by setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low, and SS high.
- **SPI.end()** - disables the SPI bus (leaving pin modes unchanged)
- ReturnByte **SPI.transfer(val)** - transfer one byte over the SPI bus, both sending and receiving.

Note about Slave Select (SS) pin on AVR based boards

- **SPI library supports only master mode** ⇒ SS pin should be set always as OUTPUT (otherwise the SPI interface could be put automatically into slave mode by hardware, rendering the library inoperative).
- **It possible to use any pin as the Slave Select (SS) for the devices.** For example, the Arduino Ethernet shield uses pin 4 to control the SPI connection to the on-board SD card, and pin 10 to control the connection to the Ethernet controller.

SPI with Arduino

Example 3 (SPI) - Controlling a Digital Potentiometer Using SPI [4]

<http://arduino.cc/en/Tutorial/SPIDigitalPot>

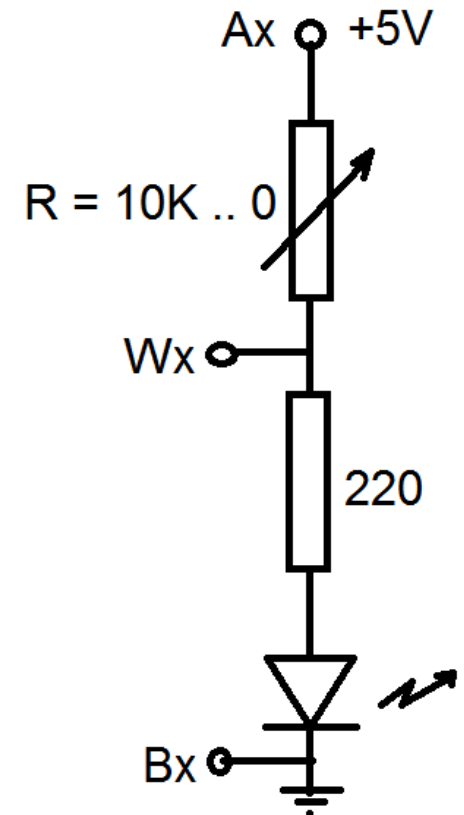
<http://www.youtube.com/watch?v=1nO2SSExEnQ>

https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

AD5206's datasheet: <http://datasheet.octopart.com/AD5206BRU10-Analog-Devices-datasheet-8405.pdf>

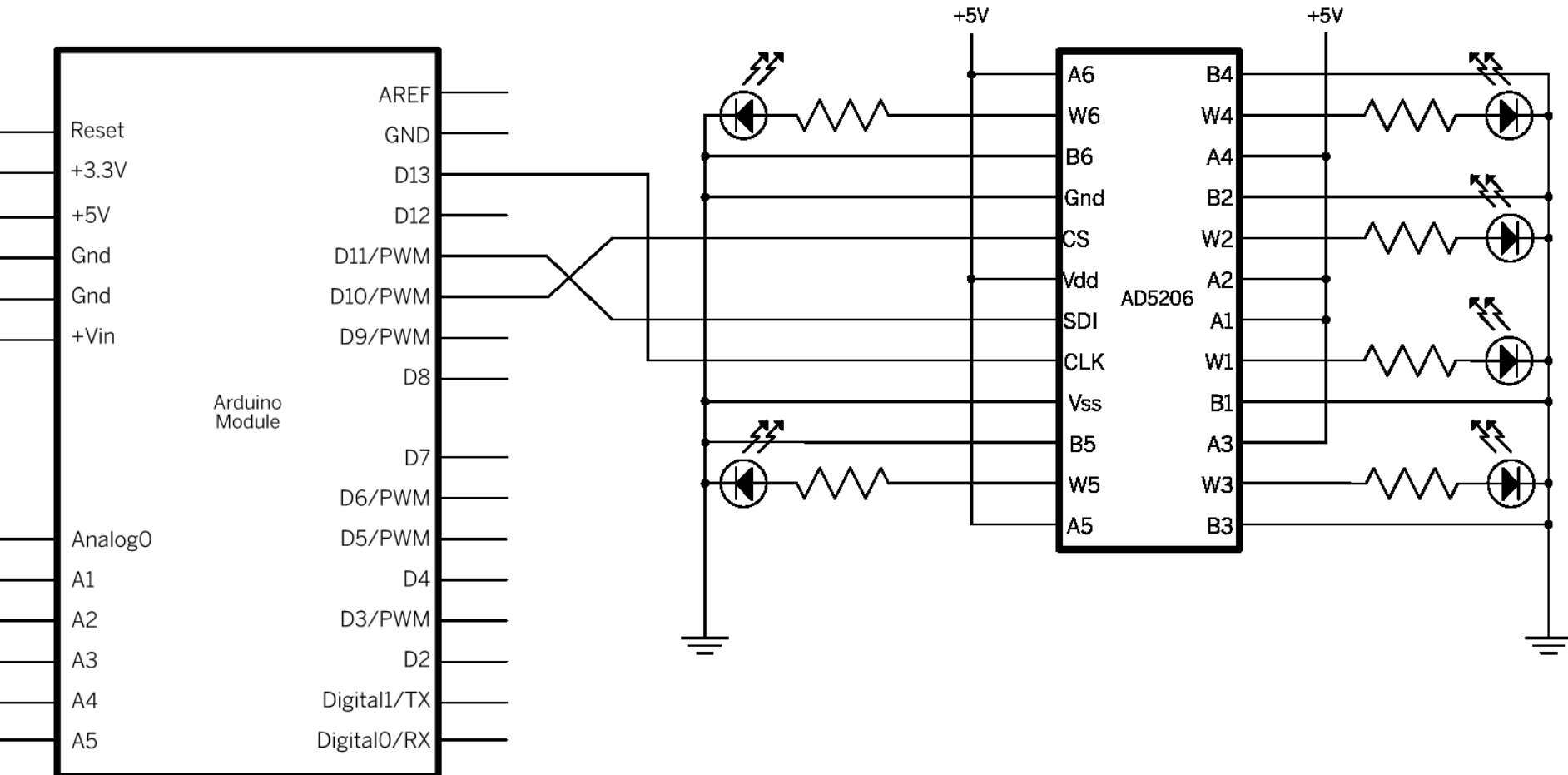
AD5206 is a 6 channel digital potentiometer (six variable resistors (potentiometers) built in for individual electronic control).

- 3 pins on the chip for each variable resistors (they can be interfaced as for a mechanical potentiometer): Ax, Bx and Wx (Wiper).
- pin A = high, pin B = low and pin W = variable voltage output connected to an LED (AD5206 provides a maximum resistance of 10K ohms, delivered in 255 steps (255 being the max, 0 being the least)).
- To control R you send on the SPI 2 bytes: one with the channel number (0 - 5) and one with the resistance value for the channel (0 - 255)



SPI with Arduino

Example 3 (SPI) - Controlling a Digital Potentiometer Using SPI [4]



SPI connections

- * CS - to digital pin 10 (SS pin)
- * SDI - to digital pin 11 (MOSI pin)
- * CLK - to digital pin 13 (SCK pin)

SPI with Arduino

Example 3 (SPI) - Controlling a Digital Potentiometer Using SPI

```
#include <SPI.h>
// set pin 10 as the slave select for the digital pot:
const int slaveSelectPin = 10;

void setup() {
  // set the slaveSelectPin as an output:
  pinMode (slaveSelectPin, OUTPUT);
  SPI.begin(); // initialize SPI:
}

void loop() {
  // go through the six channels of the digital pot:
  for (int channel = 0; channel < 6; channel++) {
    // change the resistance on this channel from min to max:
    for (int level = 0; level < 255; level++) {
      digitalPotWrite(channel, level);
      delay(10);
    }
    delay(100); // wait a second at the top:
    // change the resistance on this channel from max to min:
    for (int level = 0; level < 255; level++) {
      digitalPotWrite(channel, 255 - level);
      delay(10);
    }
  }
}
```

```
void digitalPotWrite(int address, int value) {
  // take the SS pin low to select the chip:
  digitalWrite(slaveSelectPin,LOW);
  // send in the address and value via SPI:
  SPI.transfer(address);
  SPI.transfer(value);
  // take the SS pin high to de-select the chip:
  digitalWrite(slaveSelectPin,HIGH);
}
```

Homework: modify the above example in order to dim-in and dim-out all the 6 LEDs simultaneously / synchronously

ASCII codes

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Extended ASCII codes

128	Ç	144	É	161	í	177	⌘	193	⊥	209	⌘	225	β	241	±
129	ü	145	æ	162	ó	178	⌘	194	⌘	210	⌘	226	Γ	242	≥
130	é	146	Æ	163	ú	179		195	⌘	211	⌘	227	π	243	≤
131	â	147	ô	164	ñ	180	⌘	196	—	212	⌘	228	Σ	244	∫
132	ä	148	ö	165	Ñ	181	⌘	197	+	213	⌘	229	σ	245	∫
133	à	149	ò	166	ª	182	⌘	198	⌘	214	⌘	230	μ	246	+
134	â	150	û	167	º	183	⌘	199	⌘	215	⌘	231	τ	247	≈
135	ç	151	ù	168	¸	184	⌘	200	⌘	216	⌘	232	Φ	248	°
136	ê	152	—	169	—	185	⌘	201	⌘	217	⌘	233	⊙	249	.
137	ë	153	Ö	170	¬	186	⌘	202	⌘	218	⌘	234	Ω	250	.
138	è	154	Û	171	½	187	⌘	203	⌘	219	■	235	δ	251	√
139	ï	156	£	172	¼	188	⌘	204	⌘	220	■	236	∞	252	_
140	î	157	¥	173	¡	189	⌘	205	=	221	■	237	φ	253	²
141	ì	158	—	174	«	190	⌘	206	⌘	222	■	238	ε	254	■
142	Ä	159	f	175	»	191	⌘	207	⌘	223	■	239	∩	255	
143	Å	160	á	176	⌘	192	⌘	208	⌘	224	α	240	≡		

Source: www.LookupTables.com

References

- [1] Arduino Serial reference guide: <http://arduino.cc/en/Reference/Serial>
- [2] Michael Margolis, Arduino Cookbook, 2-nd Edition, O'Reilly, 2012.
- [3] Arduino SPI reference guide: <http://arduino.cc/en/Reference/SPI>
- [4] Arduino Tutorials: <http://arduino.cc/en/Tutorial/SPIDigitalPot>