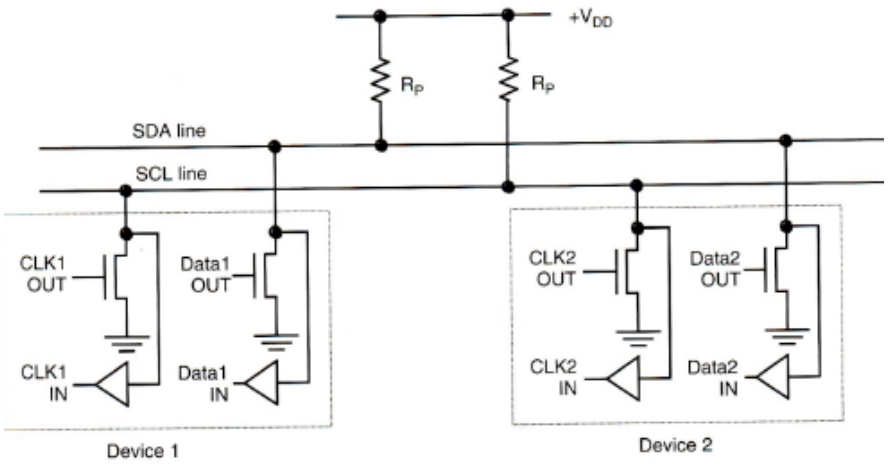
Design with Microprocessors

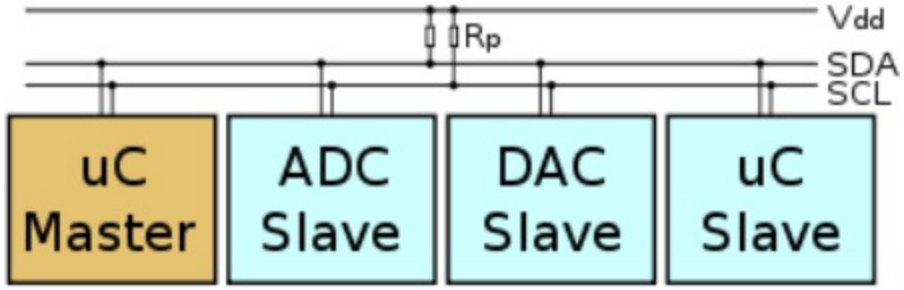
Lecture 7

Year 3 CS
Academic year 2023/2024
1st Semester

Lecturer: Radu Dănescu

I²C (Inter-Integrated Circuit)





Structure

- *SDA, SCL* – data and clock, bi-directional, connected “open collector” or “open drain”
- “Pull up” resistors hold the lines at Vcc
- A line can be pulled down to ‘0’ by any device – it becomes ‘0’ if at least one device writes a ‘0’, otherwise it remains ‘1’
- Each device has an address on 7 bits
- 16 reserved addresses
- 112 available addresses –> 112 devices on a bus

I^2C (Inter-Integrated Circuit)

Types of devices (nodes)

- *Master – generates the clock signal and the addresses*
- *Slave – receives the clock signal and the address*

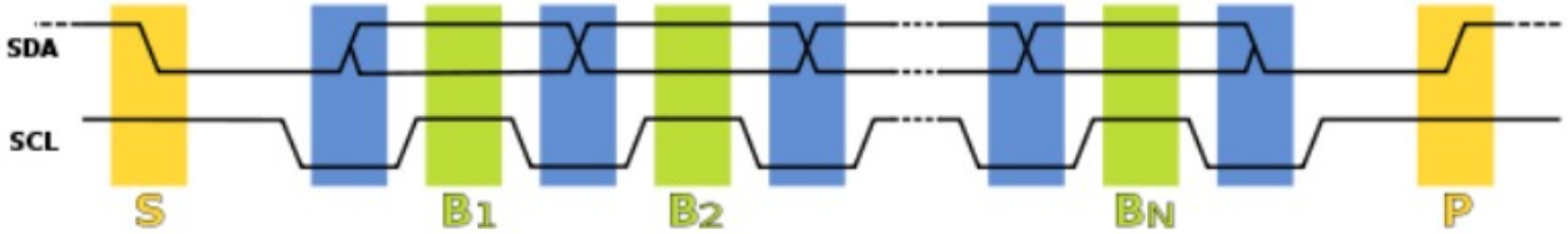
The roles of master and slave can change for the same device

Operating modes

- *master transmit* — the master node sends data to the slave
- *master receive* — the master node receives data from the slave
- *slave transmit* — the slave node sends data to the master
- *slave receive* — the slave node receives data from the master

I²C (Inter-Integrated Circuit)

Timing diagrams

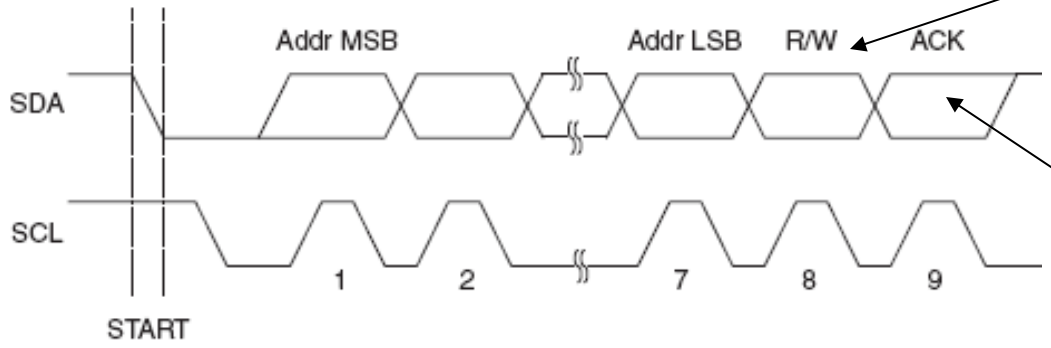


Events:

- *Start* - a transition of SDA from ‘1’ to ‘0’, with SCL remaining ‘1’
- *Bit transfer* – the value of the SDA bit changes when SCL is ‘0’, and is kept stable when SCL is ‘1’
- *Stop* – a transition of SDA from ‘0’ to ‘1’ when SCL is ‘1’

I²C (Inter-Integrated Circuit)

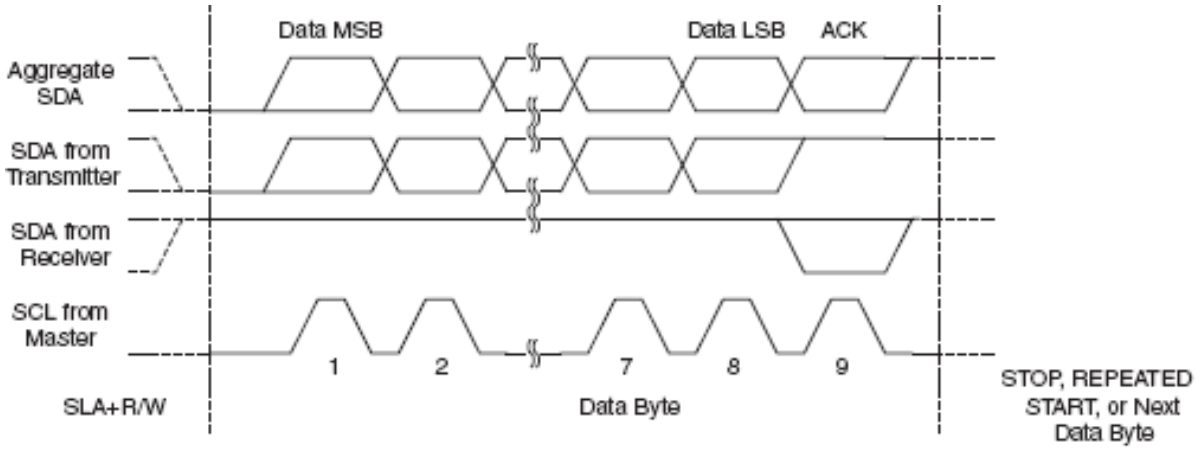
Packet format - address



‘0’ – write
‘1’ – read

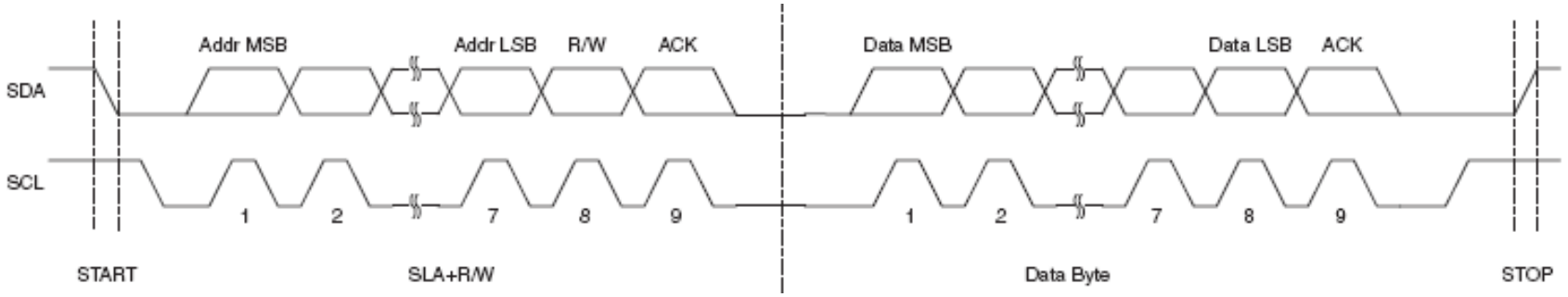
Written by the slave
‘0’ – ACK
‘1’ – NACK

Packet format - data



I²C (Inter-Integrated Circuit)

Typical transmission



Arbitration

- Each master monitors the START and STOP signals, and does not initiate a transmission as long as other master keeps the bus occupied.
- If two masters start the transmission at the same time, arbitration is performed based on the SDA line.
    - Each master checks the SDA line, and if its level is not the expected one (the one written by the master), this master loses arbitration.
    - The first master that writes a ‘1’ when the other master writes a ‘0’ loses the arbitration.
    - The losing master waits for a STOP signal, and then tries again.

I^2C (Inter-Integrated Circuit)

Adjusting the transmission speed

- 'Clock stretching'
- A slave device can hold the clock line (SCL) to '0' more time, indicating the need for a longer time to process the data
- The master device will try to set the SCL line to '1', but it will fail, due to the open collector configuration
- The master device checks whether the SCL line is set to '1', and only then the transmission will resume.

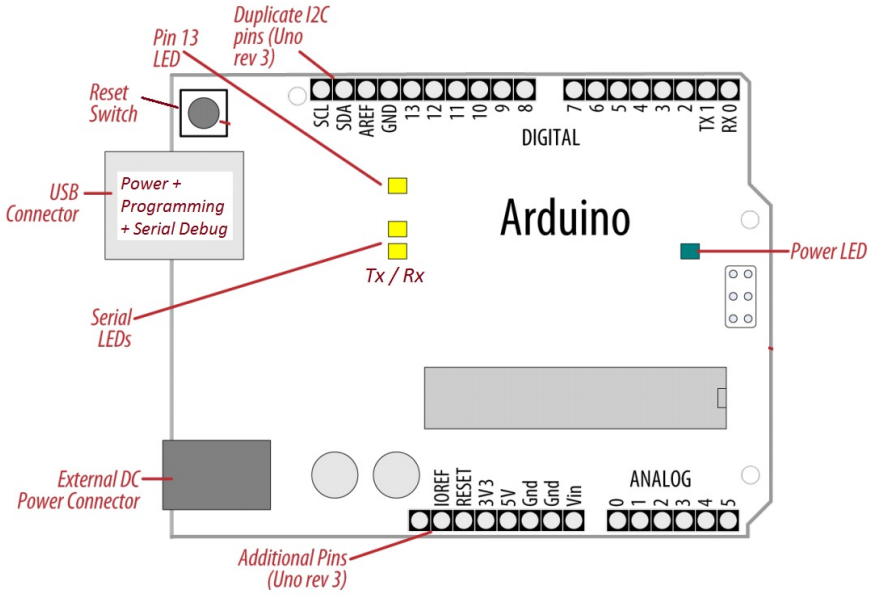
- Not all devices support 'clock stretching'
- The clock stretching interval is limited

I^2C (Inter-Integrated Circuit)

Applications
- Reading configuration data from the EEPROM SPD of SDRAM, DDRAM, DDR2 SDRAM
- Interfacing digital sensors
- Interfacing DAC and ADC converters
- Changing settings of video monitors (Display Data Channel)
- Changing the volume in intelligent speakers
- Reading hardware diagnosis sensors such as CPU temperature or CPU fan speed

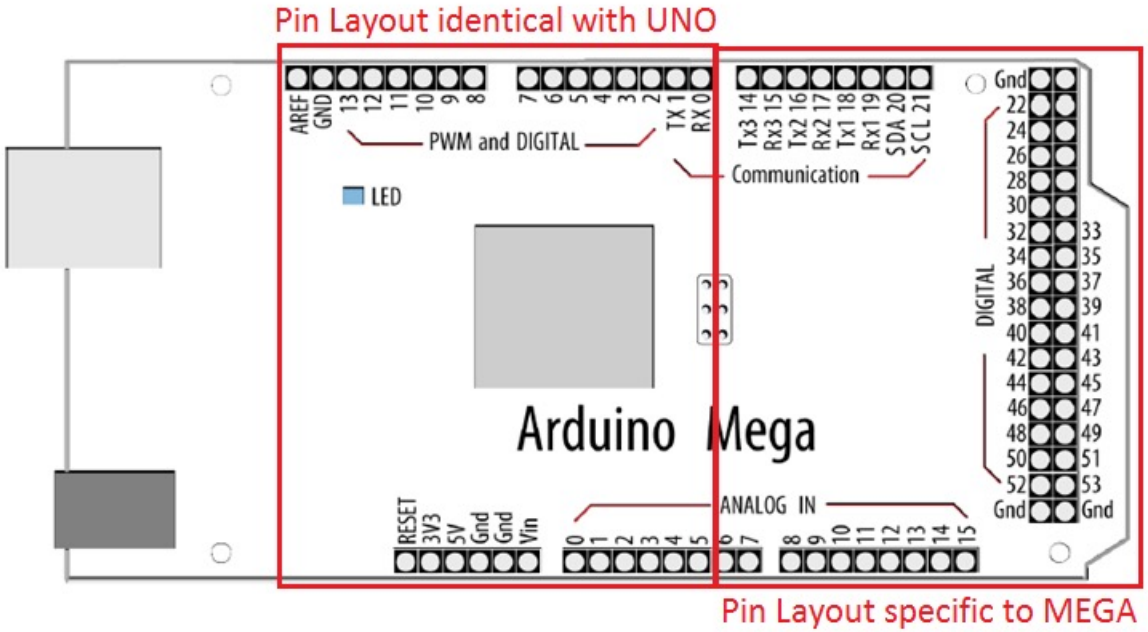
Advantages
- A microcontroller can control multiple devices using only two wires
- Devices can be attached or removed from the bus during runtime

Serial interfaces of Arduino



Arduino UNO (rev. 3)

- **UART**: pin 0 (RX) , pin 1 (TX) – Used for programming!
- **SPI**: pin 10 (SS), pin 11 (MOSI), pin 12 (MISO), pin 13 (SCK).
- **TWI(I2C)**: A4 or SDA, A5 or SCL



Arduino MEGA (rev. 3)

- **UART0**: pin 0 (RX) and pin 1 (TX) – programming; **UART1**: pin 19 (RX) and pin 18 (TX); **UART2**: pin 17 (RX) and 16 (TX); **UART3**: pin 15 (RX) and 14 (TX).
- **SPI:** pin 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS)
- **TWI(I2C)**: pin 20 (SDA) si 21 (SCL)

UART of Arduino

All Arduino boards have at least one serial port (UART / USART), accessible through the C++ object **Serial**

- Communication μC ↔ PC via USB, using the integrated USB-UART adapter – communication used also for programming the board !!
- Communication between boards, using the digital pins 0 (RX) si 1 (TX) - not recommended.

Do not use pins 0 and 1 for general I/O operations !!!

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

- If these interfaces are needed for communicating with the PC, a USB-UART external adapter must be used. Serial 1...3 do not have integrated USB adapters.
- For communicating with a device that uses TTL logic levels, connect the TX pin of the board with the RX pin of the device, the RX pin of the board with the TX pin of the device, and the ground pins (GND) together.

UART of Arduino

The **Serial** library integrated in the Arduino IDE (http://arduino.cc/en/Reference/Serial) – used for handling the serial interface communication.

The methods of the **Serial** class (selection):

- **Serial.begin(speed)** – configures the transmission speed (speed), using the default data format (8 data bits, no parity, one stop bit)
- **Serial.begin(speed, config)** – configures speed (speed) + selects other data formats (config):

  Example for *config*: SERIAL_8N1 (default), SERIAL_7E2, SERIAL_5O1 …
- **Serial.print(val)** – sends as a string of characters in human readable form (ex: Serial.print(20) will send the characters ‘2’ and ‘0’)
- **Serial.print(val, format)** – format specifies the numerical base used (BIN, OCT, DEC, HEX. For floating point, the format specifies the number of decimals used.
- **Serial.println** – Sends the data followed by (ASCII 13, '\r') + (ASCII 10, '\n')

Examples:

Serial.print(78) sends "78“ Serial.print(78, BIN) sends "1001110"
Serial.print(1.23456) sends "1.23“ Serial.println(1.23456, 4) sends "1.2346"
Serial.print("Hello") sends "Hello"

UART on Arduino

The methods of the **Serial** class (selection):

- byte IncomingByte = **Serial.read()** - reads a byte from the serial interface
- int NoOfBytesSent = **Serial.write(data)** – writes a byte, in original binary format, through the serial interface. Data can be given as a byte (val) or as a byte string (str) or as a buffer of a specific length (buf, len)
- **Serial.flush**() – waits until data transmission is complete.
- int NoOfBytes = **Serial.available()** – Returns the number of bytes available for reading. Data is already received and stored in a buffer (maximum capacity 64 bytes)
- **serialEvent()** – a user defined function that is called automatically when data is available in the buffer. Use **Serial.read()** inside this function to read the available data.
- serialEvent1(), serialEvent2(), serialEvent3() – For Arduino Mega, these functions are called automatically for the serial interfaces 1…3.

UART of Arduino

Example program 1:

```
void setup() {
    Serial.begin(9600); // open serial port, sets speed to 9600 bps
    Serial.println(“Hello”);
}

void loop() {}
```

Example program 2 (Arduino Mega only):

```
// Arduino Mega uses four serial ports
// (Serial, Serial1, Serial2, Serial3),
// They can be configured to have different speeds:

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() {}
```

UART of Arduino

Communication μC ↔ PC: receiving data sent via Serial to Arduino – receiving data from the PC or other device, and reacting to it:

Example program 3 – receives a digit (character from '0' to '9') and changes the state of a LED depending on the read digit.

```
const int ledPin = 13;              // pin LED
int blinkRate=0;                    // rate of blink for the LED
void setup()
{
  Serial.begin(9600); // init the serial port
  pinMode(ledPin, OUTPUT); // set the LED pin as output
}
void loop() {
  if ( Serial.available())          // Check if data is available to read
  {
          char ch = Serial.read(); // Read the received character
          If( isDigit(ch) )         // Is it digit ?
          {
              blinkRate = (ch - '0');             // Convert to a number
              blinkRate = blinkRate * 100; // Rate = 100ms * read_digit
          }
  }
  blink();
}
```

UART of Arduino

Example program 3 – cont.

```
// toggle the LED depending on the rate
void blink()
{
  digitalWrite(ledPin,HIGH);
  delay(blinkRate); // computed delay
  digitalWrite(ledPin,LOW);
  delay(blinkRate);
}
```

To use this example:

- Use Serial Monitor, included in the Arduino environment – from the Tools menu or by pressing <CTRL+SHIFT+M>)
- Set the same speed in the serial monitor as the one set using Serial.begin()
- Type digits, and press “Send”.

UART of Arduino

Example program 4 – example 3 changed to use **serialEvent()**

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

void serialEvent() // this function is called automatically when data is available to read
{
  while(Serial.available()) // as long as data is available
  {
     char ch = Serial.read(); // read them
     Serial.write(ch); // echo – send them back for the user to check
     if( isDigit(ch) ) // is it digit ?
     {
        blinkRate = (ch - '0'); // convert to number
        blinkRate = blinkRate * 100; // compute delay time
     }
   }
}
```

Software UART on Arduino

- Besides the hardware interfaces, Arduino allows serial communication on other digital pins, the serialization / de-serialization processes being done in software.

- The SoftwareSerial library is included in the Arduino IDE (requires including the header softwareserial.h)

- The reception pin **(RX) must be connected to a digital pin that can trigger an external interrupt at level change**:

    - In Arduino Mega, these pins are: 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)

- Multiple SoftwareSerial objects can be created, but only one can be active at one time

- Constructing the object – the reception and transmission pins must be specified:

```
SoftwareSerial mySerial =  SoftwareSerial(rxPin, txPin)
```

- The library implements the functions **begin**, **read**, **write**, **print**, **println**, which are used in the same way as in the case of the hardware interface.

Software UART on Arduino

Example: communication using two serial interfaces, a hardware one connected to the PC, and a software one connected to a UART compatible device.

- Arduino plays the role of a communication relay: what is received on one interface is transmitted on another.

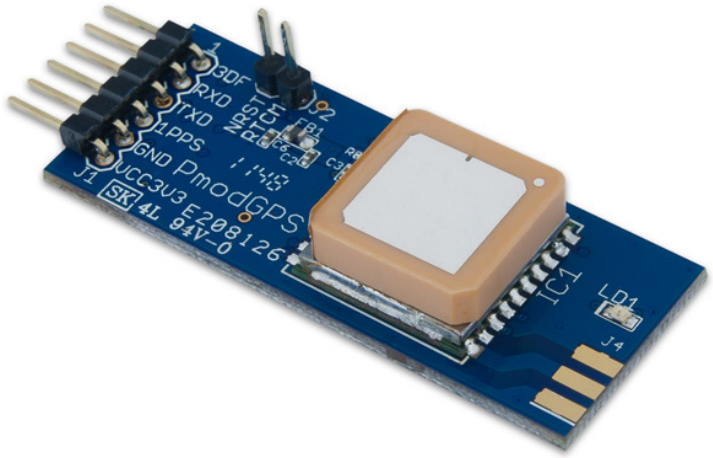
```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(10, 11); // Software serial interface uses pin 10 for RX, and pin 11 for TX

void setup()
{
 Serial.begin(9600); // Hardware interface configuration
 mySerial.begin(9600); // Software interface configuration
}

void loop()
{
  if (mySerial.available())
    Serial.write(mySerial.read());      // read from software, write to hardware
  if (Serial.available())
    mySerial.write(Serial.read());       // read from hardware, write to software
}
```

Software UART on Arduino

Example: communication using two serial interfaces, a hardware one connected to the PC, and a software one connected to a UART compatible device.

- Possible use of the example program: viewing data transferred by a UART device on the Serial Monitor, and sending commands to the device via Serial Monitor

- Example device and its output: Digilent PMod GPS receiver:




```
$GPGGA,064951.000,2307.1256,N,12016.4438,E,1,8,0.
95,39.9,M,17.8,M,,*65
$GPGSA,A,3,29,21,26,15,18,09,06,10,,,,,2.32,0.95,
2.11*00
$GPGSV,3,1,09,29,36,029,42,21,46,314,43,26,44,020
,43,15,21,321,39*7D
$GPRMC,064951.000,A,2307.1256,N,12016.4438,E,0.03
,165.48,260406,3.05,W,A*55
$GPVTG,165.48,T,,M,0.03,N,0.06,K,A*37
```

- Other UART devices that can be verified/debugged this way: Bluetooth adapters, WiFi adapters, etc.

SPI communication on Arduino

Use of the SPI library (http://arduino.cc/en/Reference/SPI) [3]

Available methods:

- **SPI.setBitOrder(order)** – order of the bits: LSBFIRST or MSBFIRST
- **SPI.setDataMode(mode)** – phase and polarity of the transmission: SPI_MODE0, SPI_MODE1, SPI_MODE2 or SPI_MODE3 (the combinations CPHA and CPOL of SPCR)
- **SPI.setClockDivider()** – selects the clock division: SPI_CLOCK_DIV(2 .. 128)
- **SPI.begin()** – activates the SPI interface, configures the pins SCK, MOSI and SS as output, sets SCK and MOSI on LOW, and SS on HIGH.
- **SPI.end()** – de-activates SPI, but leaves the pins in the mode they were set up at initialization
- ReturnByte **SPI.transfer(val)** - transfers a byte (val) on the SPI bus, receiving in the same time ReturnByte.

- The SPI library operates only in master mode
- Any pin can be used as Slave Select (SS).

SPI communication on Arduino

Example (SPI) – Controlling a digital potentiometer using SPI [4]

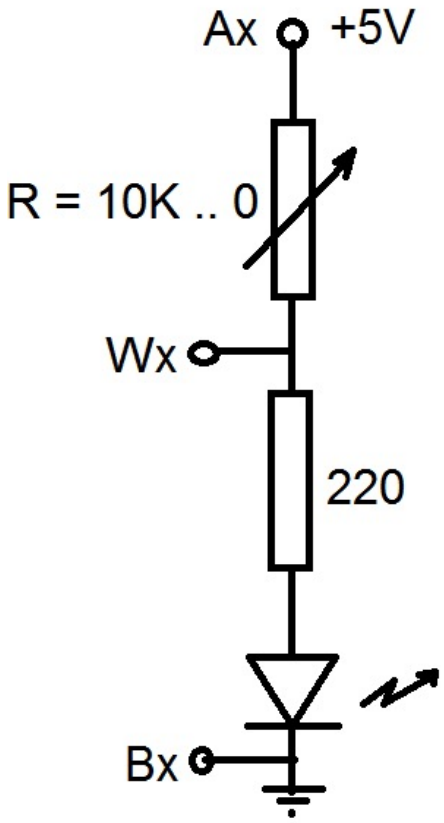
http://arduino.cc/en/Tutorial/SPIDigitalPot

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

AD5206 datasheet: http://datasheet.octopart.com/AD5206BRU10-Analog-Devices-datasheet-8405.pdf

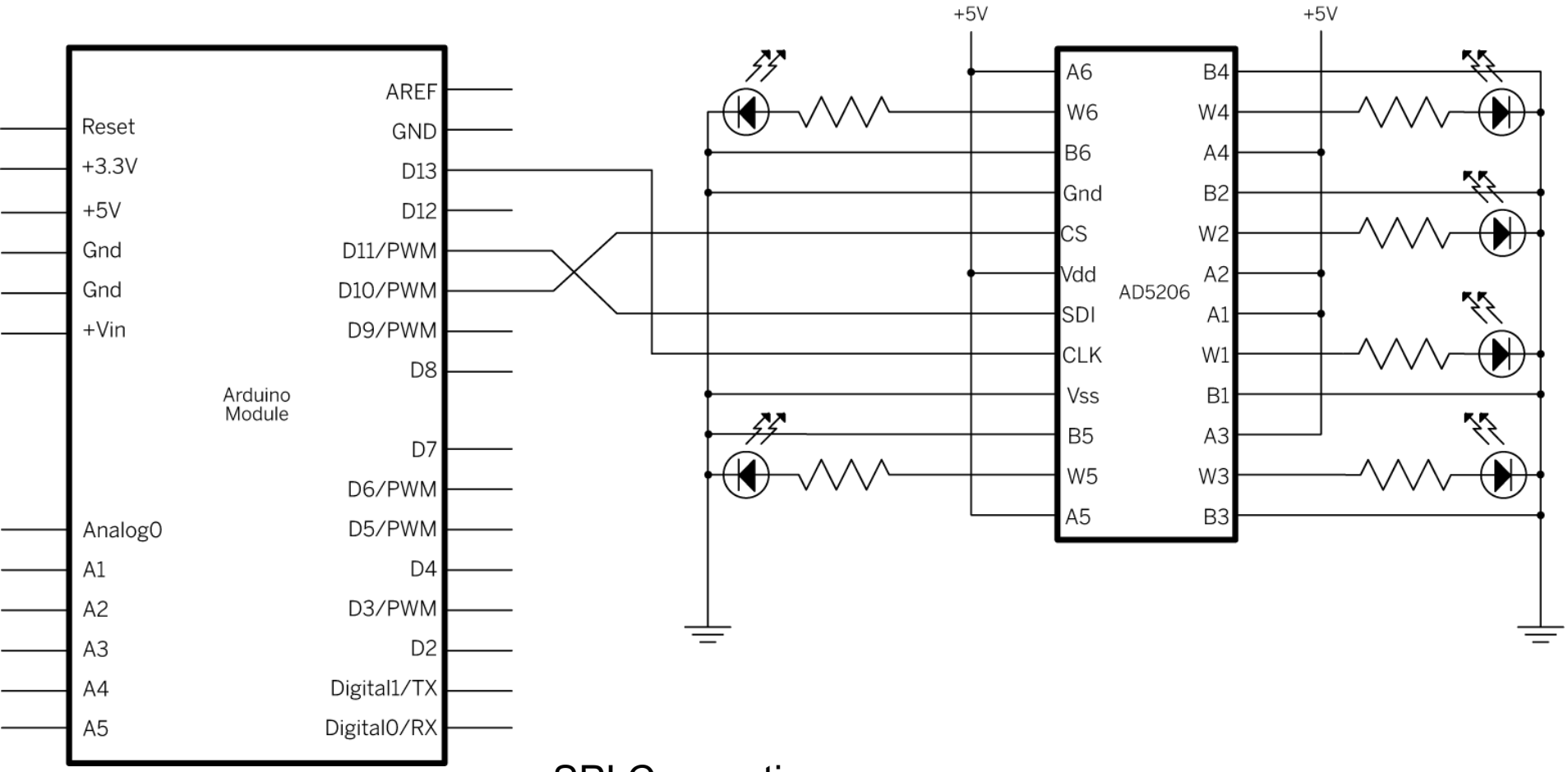
AD5206 is a 6 channel digital potentiometer (equivalent to 6 individual potentiometers).

- 3 pins on the chip for each element: Ax, Bx si Wx (Wiper - cursor).
- pin A = HIGH, pin B = LOW and pin W = variable voltage. R has a maximum resistance of 10 KOhm, divided into 255 steps.
- For controlling the resistance, two bytes must be sent on SPI: the first one selects the channel (0 - 5), and the second one the value for each channel (0 - 255) .



SPI communication on Arduino

Example (SPI) – Controlling a digital potentiometer using SPI [4]



SPI Connections

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

SPI Communication on Arduino

Example (SPI) – Controlling a digital potentiometer using SPI [4]

```
#include <SPI.h>
// pin 10 is SS
const int slaveSelectPin = 10;

void setup() {
  // SS must be configured as output
pinMode (slaveSelectPin, OUTPUT);
SPI.begin();   // activate SPI
}

void loop() {
  // for each of the 6 channels
  for (int channel = 0; channel < 6; channel++) {
    // change the resistance of each channel from min to max
    for (int level = 0; level < 255; level++) {
      digitalPotWrite(channel, level);
      delay(10);
    }
     delay(100); // wait 100 ms at maximum resistance
    // change resistance from max to min
    for (int level = 0; level < 255; level++) {
      digitalPotWrite(channel, 255 - level);
      delay(10);
    }
  }
}
```

```
// SPI based function for changing the
// resistances
void digitalPotWrite(int address, int value) {
  // activates SS by setting it LOW
  digitalWrite(slaveSelectPin, LOW);
  // write the channel number and its value:
  SPI.transfer(address);
  SPI.transfer(value);
  // inactivate SS by setting it to HIGH
  digitalWrite(slaveSelectPin, HIGH);
}
```

I2C Communication on Arduino

Use the **Wire** library of the Arduino IDE

- An Arduino board can be either I2C master or I2C slave

Wire class methods:

- **Wire.begin(address)** – activates the I2C interface. The address parameter indicates the 7-bit address that Arduino will use on the I2C bus as a slave. If no address is given, the device will be set up as master.

- **Wire.requestFrom(address, quantity)** – master requires data of **quantity** bytes from a slave identified by **address**. The function can be called also as **Wire.requestFrom(address, quantity, stop)**, **stop** being a boolean value specifying whether the master will free the bus afterwards or not.

- **Wire.beginTransmission(address)** – begins data transmission from master to a slave specified by **address**. After calling this function, data can be written using write() .

- **Wire.write(value)** – writes a byte. This function can be called by a slave, if requested by the master, or by the master after calling **beginTransmission**. Syntax: **Wire.write(string)**, or **Wire.write(data, length)**.

- **Wire.endTransmission()** – called by master to perform the actual transmission set up by **beginTransmission**, using the data queued by **Wire.write()**.

I2C Communication on Arduino

- **Wire.available()** – returns the number of available bytes to read.

- **Wire.read()** – reads a byte, if available() > 0. Can be called by master or by slave.

- **Wire.onReceive(handler)** – configures a **handler function**, for the slave device, which will be called automatically when data from the master arrives.

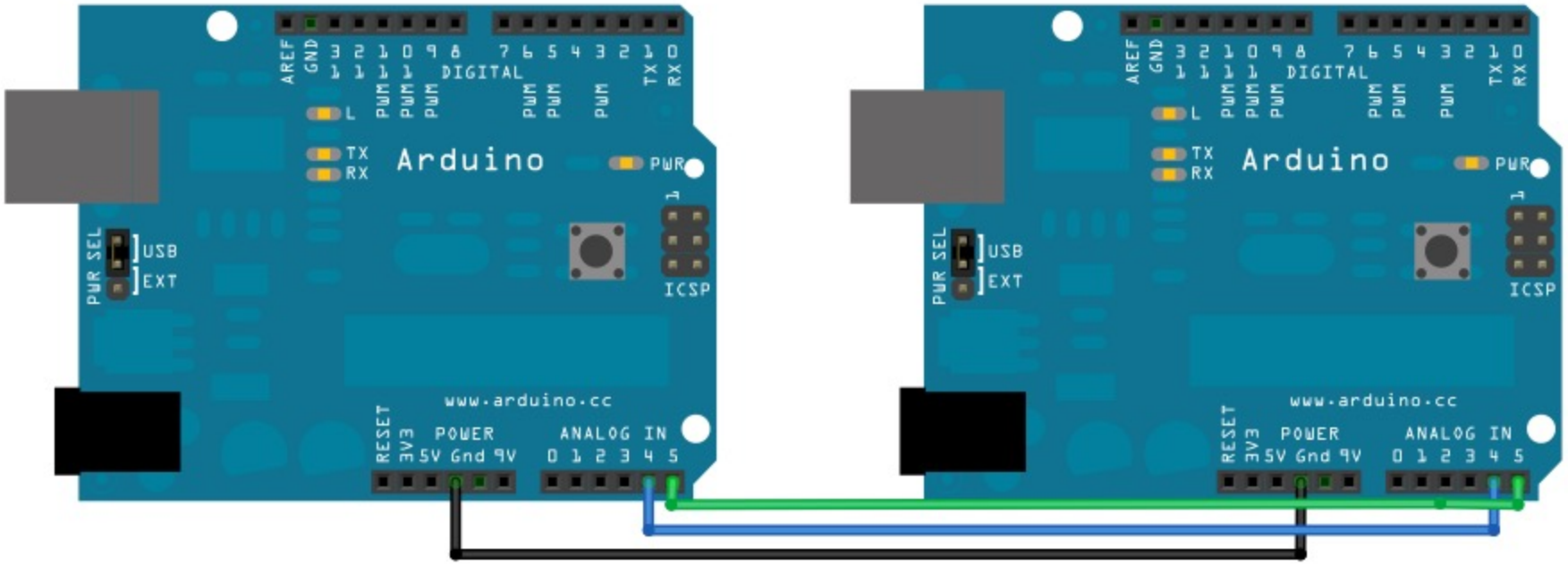
- **Wire.onRequest(handler)** – configures a **handler function**, for the slave device, which will be called automatically when master requests data.

- **Wire.setClock(frequency)** – changes the speed of I2C communication. Default is 100000 Hz, and the maximum allowed is 400000 Hz. Some processors also support 10000 (low speed mode), 1000000 (fast mode plus) and 3400000 (high speed mode).

I2C Communication on Arduino

Example: connecting two Arduino boards by I2C, one being master, the other one slave. Master will send data, and slave will receive.

Source: http://arduino.cc/en/Tutorial/MasterWriter



I2C Communication on Arduino

Example: connecting two Arduino boards by I2C, one being master, the other one slave. Master will send data, and slave will receive.

Code for master:

```
#include <Wire.h>
void setup()
{
  Wire.begin(); // activates the I2C interface as master
}

byte x = 0; // byte to be sent, it will be incremented continuously

void loop()
{
  Wire.beginTransmission(4);  // begins a new transmission process, to slave address 4
  Wire.write("x is ");        // writes a string
  Wire.write(x);              // writes a byte
  Wire.endTransmission();    // finalization of the writing process
  x++; // increment the value to be sent
  delay(500);
}
```

I2C Communication on Arduino

Example: connecting two Arduino boards by I2C, one being master, the other one slave. Master will send data, and slave will receive.

Code for slave:

```
#include <Wire.h>
void setup()
{
  Wire.begin(4);                 // activates the i2c interface as a slave, having the address 4
  Wire.onReceive(receiveEvent);  // registers the function receiveEvent to be called when data arrive
  Serial.begin(9600);            // activates the serial interface, to display the received data on the PC
}

void loop()  // this function does nothing
{
  delay(100);
}

void receiveEvent(int howMany) // this is called automatically when data is available for slave
{
  while(Wire.available() >1 ) // all available bytes but the last one
  {
    char c = Wire.read(); // read byte by byte
    Serial.print(c);         // and send to the PC
  }
  int x = Wire.read();    // last character is interpreted as a number
  Serial.println(x);         // and printed as a number
}
```

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 SPI Tutorials: http://arduino.cc/en/Tutorial/SPIDigitalPot

[5] Arduino I2C Library: http://arduino.cc/en/reference/wire

[6] Arduino I2C tutorial: http://arduino.cc/en/Tutorial/MasterWriter