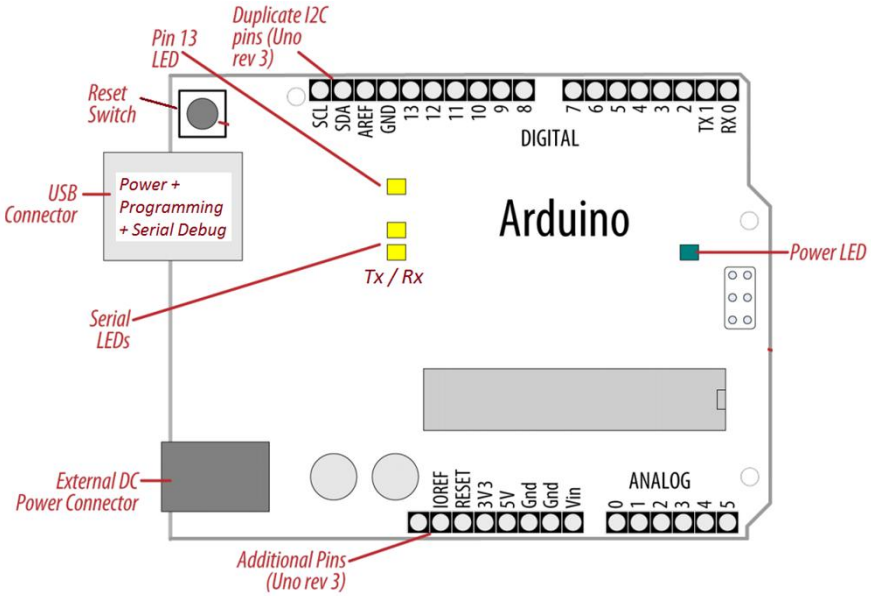
Design with Microprocessors

Year III Computer Science
1-st Semester

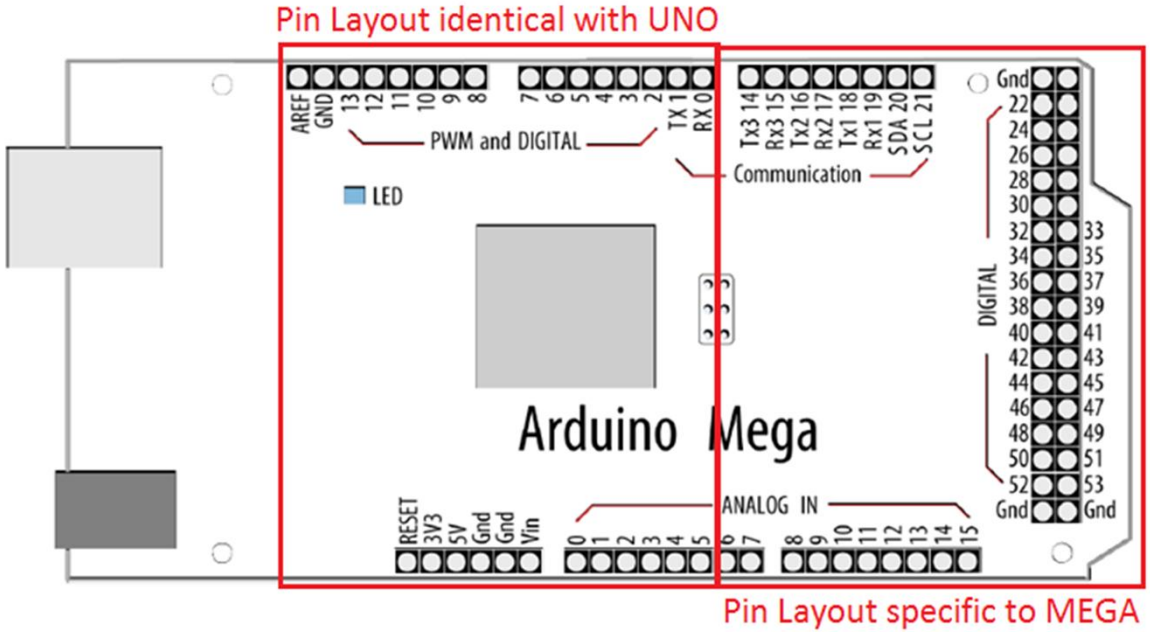
Lecture 7a: Serial data transfer with Arduino

Arduino UNO (rev. 3)



Arduino UNO (rev. 3)

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



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**

- μC ↔ PC communication via the on-board USB port (USB-to-serial adapter) – used also for the board programming !!
- inter-board communication using pins 0 (RX) and 1 (TX) - not recommanded

If you use these functions, you cannot use 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, 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.)

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 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) + costumizable 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 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();
}
```

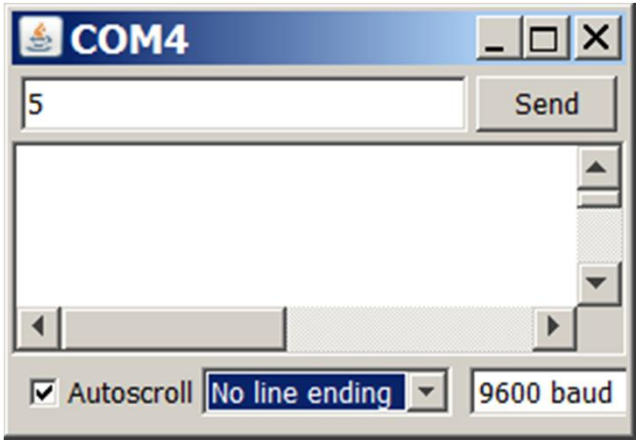
Serial communication with Arduino

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").
- 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 multple 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 multple serial test

```
#include <SoftwareSerial.h>

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

void setup()
{
  // Open serial communications and wait for port to open:
  Serial.begin(57600);   // 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 $\Rightarrow$ 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]

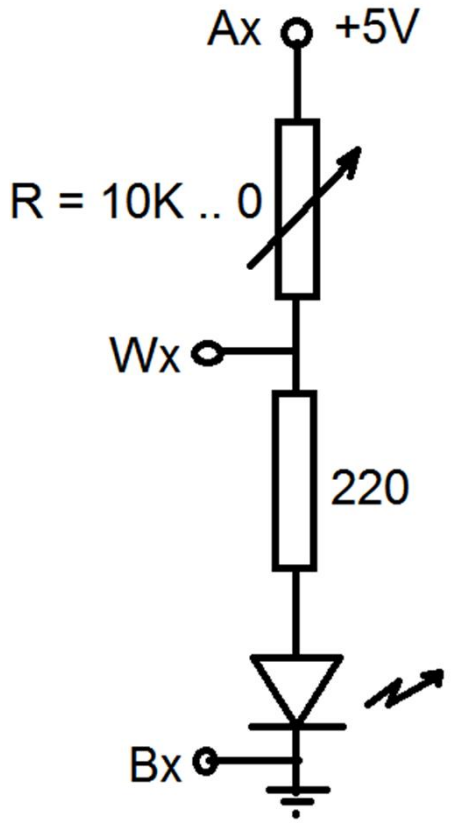
http://arduino.cc/en/Tutorial/SPIDigitalPot

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

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

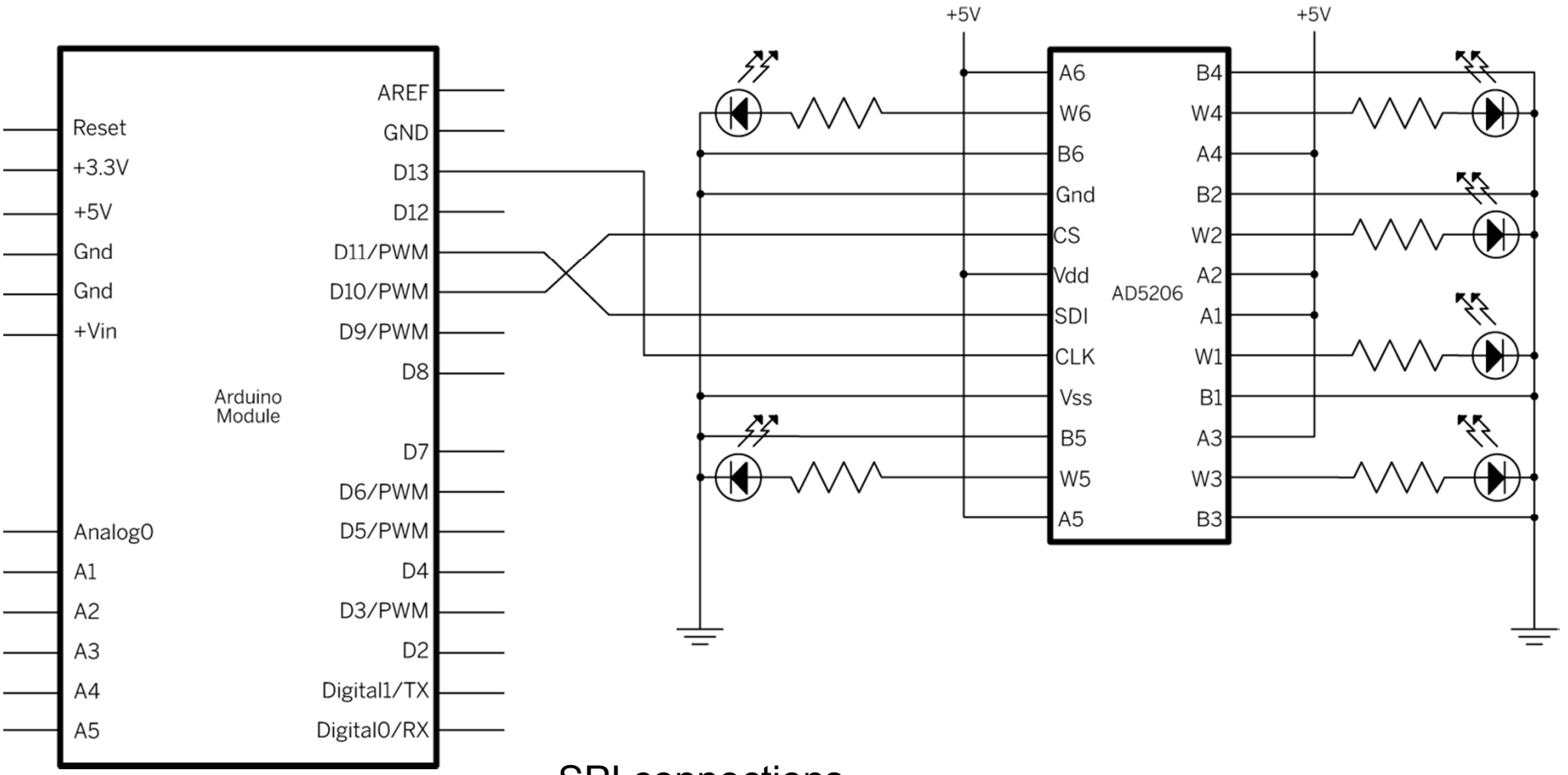
AD5206 is a 6 channel digital potentiometer (six variable resistors (potentiometers) built in for individual electronic control).

- 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 al the 6 LEDs simultaneously / synchronously

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