Warsaw University of Technology
Faculty of Electronics and Information Technology
Institute of Telecommunications

EIOT

Laboratory exercise

Exercise 1
Title: Programming embedded IoT nodes, including their assorted hardware-oriented peripherals.

Prerequisites:
Learn about the Arduino programming model and get familiar with some Arduino code examples::
https://www.arduino.cc/en/Reference/HomePage
https://www.arduino.cc/en/Tutorial/BuiltInExamples

1. Setting up your environment

Laptop computers used in the lab (Lenovo Yoga) run under the Windows 10 operating system. The Arduino IDE (ver. 1.8.1) is installed on them. The IDE is available via a desktop icon or a command line (C:\Programs\Arduino\arduino.exe).

Connect the Arduino UNO board (received from your instructor) to the computer, using the "USB A – USB B" cable. Check that your connection is correct by choosing **`Tools->Port`**. You should see information on a new COM port; it should say **`Arduino Uno`**. Then choose **`Tools->Get Board Info`**. The IDE should read and display the unique serial number of the board. In case the IDE has problems with detecting and identifying the board, disconnect it and then connect it again; wait for a moment, while Windows automatically searches for required drivers. If you still cannot see the Arduino Uno board on the **`Tools->Port`** port list, we might have a hardware problem. Report such a case to your instructor.

2. Compiling and running simple examples

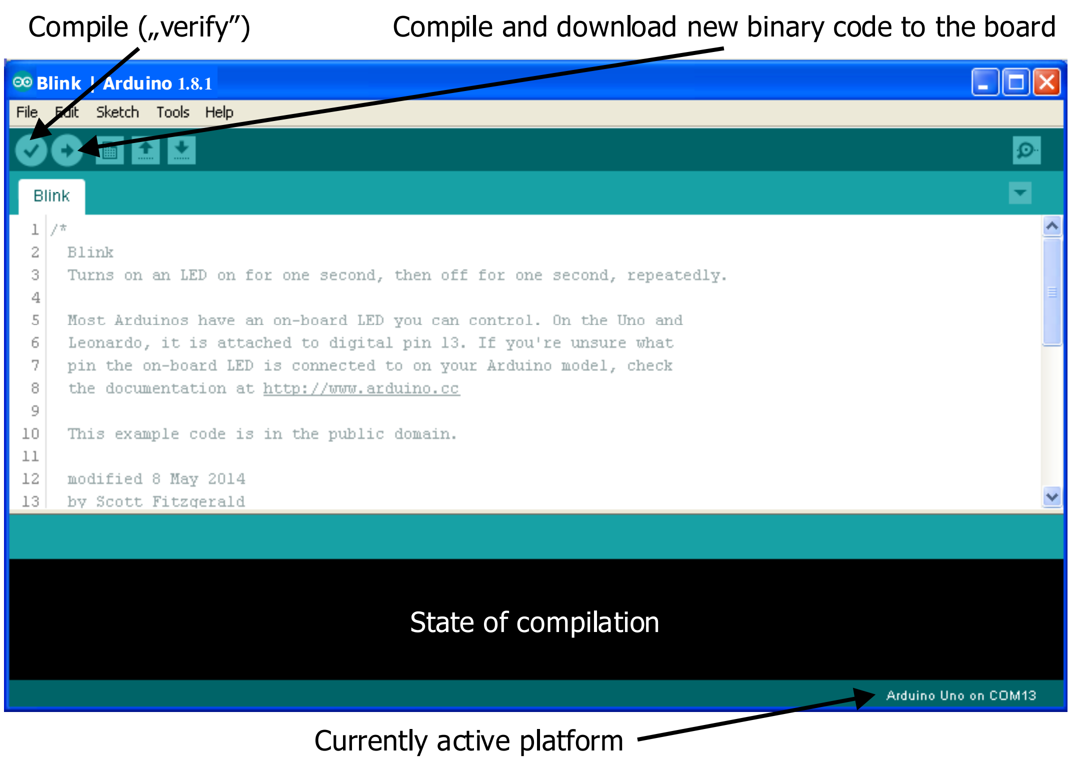
The Arduino IDE comes with many examples, available in **`File->Examples`**. Choose **`01.Basics`** and then pick **`Blink`**. The main screen of the IDE is presented in Fig. 1

If you press the **`Verify`** buttom, the **`Blink`** program will get compiled. Output from the compilation process is presented in the compilation window (see Fig. 1). You can check for any compiler error messages in that window.

The **`Blink`** application consists of two functions:

```
void setup();
void loop();
```

`setup()` is called once (by the Arduino's system software) right after you start the application; its task is to initialize the Arduino board. **`loop()`** is called repetitively (in a loop), for the entire application lifetime, once **`setup()`** returns. The execution time of the **`loop()`** function should be as short as possible, so that there is enough time to correctly perform „system tasks" (say, handling some peripherals).
Once you successfully compile and download your code to the board, the application is started automatically.



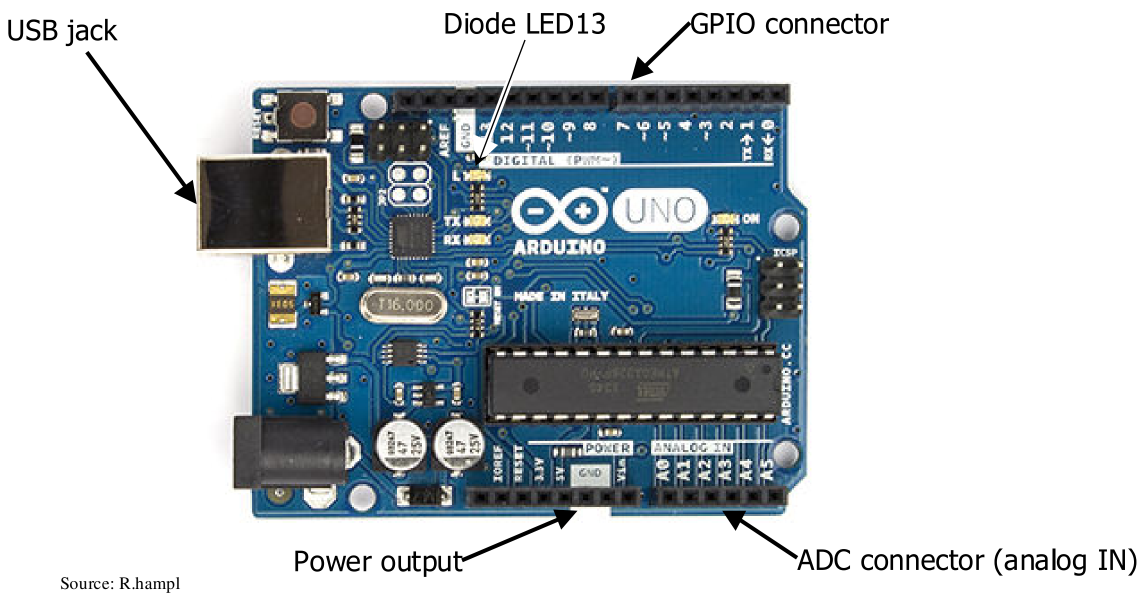
```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the Uno and
  Leonardo, it is attached to digital pin 13. If you're unsure what
  pin the on-board LED is connected to on your Arduino model, check
  the documentation at http://www.arduino.cc

  This example code is in the public domain.

  modified 8 May 2014
  by Scott Fitzgerald
```

Fig. 1. Arduino IDE

The source code of the **Blink** program is presented in Listing 1. In **setup()**, the GPIO pin number 13 is configured as output. The **loop()** changes the state of the GPIO pin; the pace of the changes is determined by the invocations of **delay()**. The position of a LED connected to GPIO pin number 13 is show in Fig. 2.

```
void setup() {
   pinMode(13, OUTPUT);      // initialize digital pin 13 as an output.
}
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

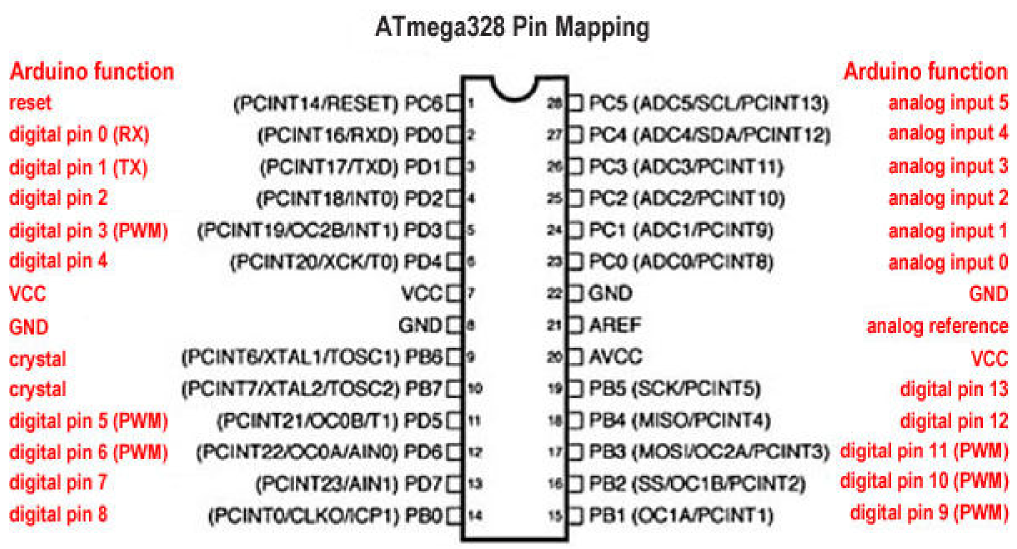
Listing 1. Blink application



Source: R.hampl

Fig. 2. Arduino UNO

In further work with Arduino UNO, one might need the mapping between pin names used in the Atmega 328P (the board's microcontroller) documentation and those used by libraries that come with Arduino IDE. The mapping is shown in Fig. 3.



Źródło: www.jameco.com

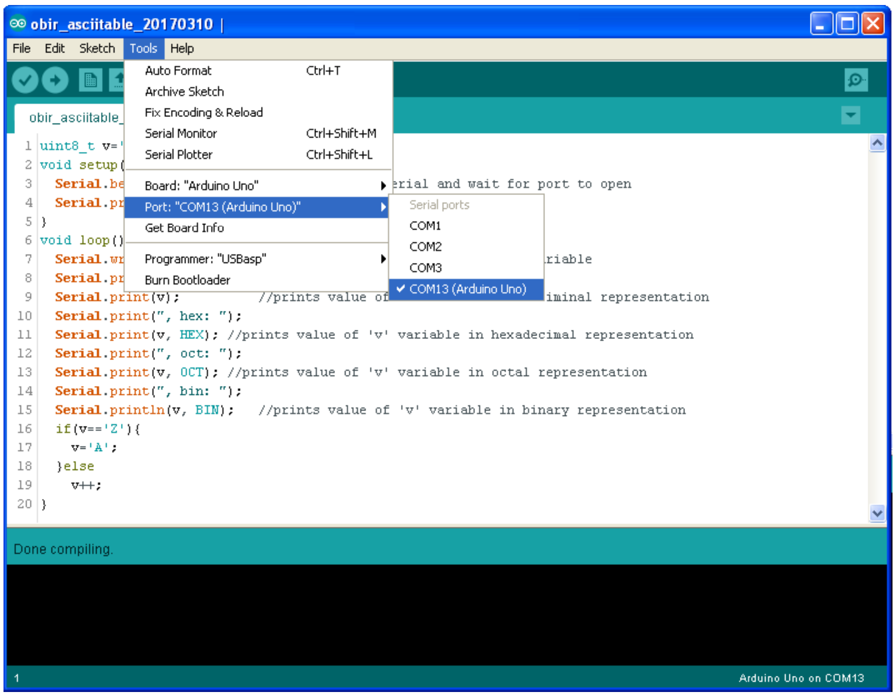
Fig.3. Mapping between Atmega328P and Arduino UNO pin names

Another useful Arduino application example, showing how to handle a serial port, is **ASCIITable**. (Choose **File->Examples** and go to the **04.Communication**.) Listing 2 presents a modified version of the program. You can run it to see if the serial port works correctly. Also, it shows how to produce output on the Serial Monitor.

```
uint8_t v='A';
void setup() {
  Serial.begin(115200);   //Initialize serial and wait for port to open
  Serial.println(F("ASCII Table ~ Character Map"));
}
void loop() {
  Serial.write(v);        //prints raw binary version of 'v' variable
  Serial.print(F(", dec: "));
  Serial.print(v);        //prints value of 'v' variable in decimal representation
  Serial.print(F(", hex: "));
  Serial.print(v, HEX);   //prints value of 'v' variable in hexadecimal representation
  Serial.print(F(", oct: "));
  Serial.print(v, OCT);   //prints value of 'v' variable in octal representation
  Serial.print(F(", bin: "));
  Serial.println(v, BIN); //prints value of 'v' variable in binary representation
  if(v=='Z')
    v='A';
  else
    v++;
}
```

Listing 2. Serial port demonstrating program

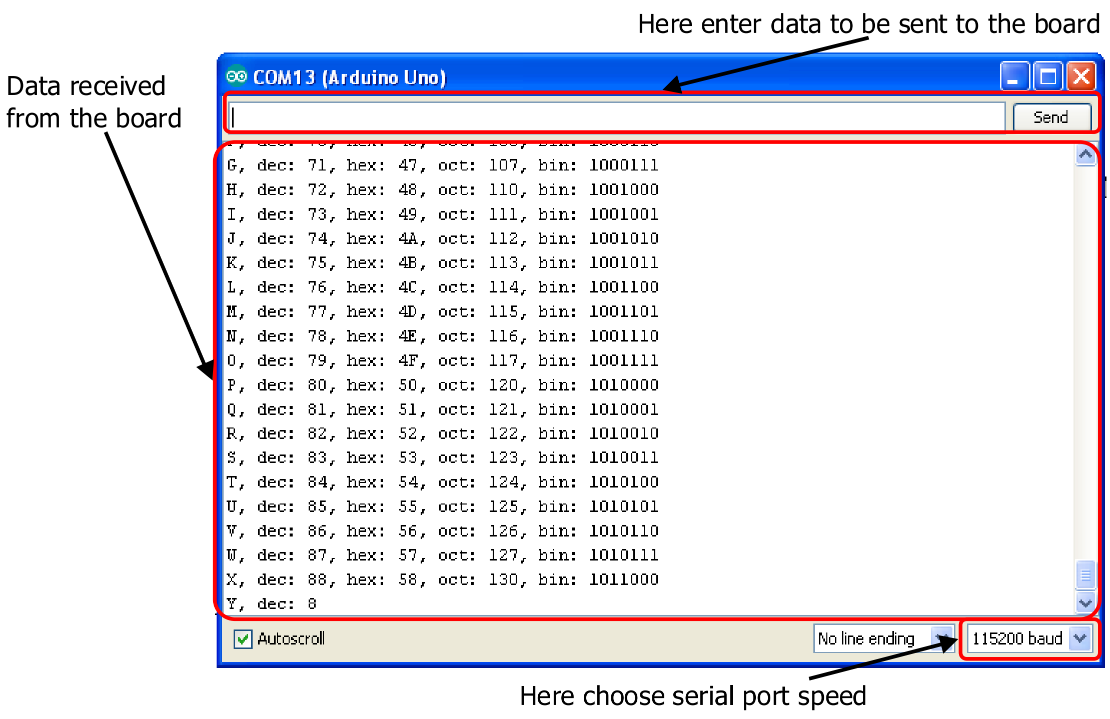
To use the Arduino IDE's Serial Monitor you need to (a) pick the serial port, to which the Arduino Uno is connected (Fig. 4), and (b) to start **Serial Monitor** from the **Tool** menu. The Serial Monitor window, attached to the Arduino UNO, is shown in Fig. 5.
Referring to Listing 2., you will notice invocations to two print functions (methods). **Serial.print()** outputs whatever is provided as the first argument (one can optionally specify a number system), while **Serial.println()** also adds a line break. By using both, you can make your output more readable.



```
uint8_t v='
void setup(
  Serial.be                                                  erial and wait for port to open
  Serial.pr
}
void loop()
  Serial.wr                                                                    riable
  Serial.pr
  Serial.print(v);         //prints value of                                   iminal representation
  Serial.print(", hex: ");
  Serial.print(v, HEX); //prints value of 'v' variable in hexadecimal representation
  Serial.print(", oct: ");
  Serial.print(v, OCT); //prints value of 'v' variable in octal representation
  Serial.print(", bin: ");
  Serial.println(v, BIN);   //prints value of 'v' variable in binary representation
  if(v=='Z'){
    v='A';
  }else
    v++;
}
```

```
Done compiling.
```

Fig.4. Choosing the COM port to communicate with Arduino UNO



```
G, dec: 71, hex: 47, oct: 107, bin: 1000111
H, dec: 72, hex: 48, oct: 110, bin: 1001000
I, dec: 73, hex: 49, oct: 111, bin: 1001001
J, dec: 74, hex: 4A, oct: 112, bin: 1001010
K, dec: 75, hex: 4B, oct: 113, bin: 1001011
L, dec: 76, hex: 4C, oct: 114, bin: 1001100
M, dec: 77, hex: 4D, oct: 115, bin: 1001101
N, dec: 78, hex: 4E, oct: 116, bin: 1001110
O, dec: 79, hex: 4F, oct: 117, bin: 1001111
P, dec: 80, hex: 50, oct: 120, bin: 1010000
Q, dec: 81, hex: 51, oct: 121, bin: 1010001
R, dec: 82, hex: 52, oct: 122, bin: 1010010
S, dec: 83, hex: 53, oct: 123, bin: 1010011
T, dec: 84, hex: 54, oct: 124, bin: 1010100
U, dec: 85, hex: 55, oct: 125, bin: 1010101
V, dec: 86, hex: 56, oct: 126, bin: 1010110
W, dec: 87, hex: 57, oct: 127, bin: 1010111
X, dec: 88, hex: 58, oct: 130, bin: 1011000
Y, dec: 8
```

Fig. 5. Serial monitor

Incidentally, you might have noticed that in some invocations of **`print()`** in Listing 2., the argument string is actually first passed to the auxiliary function **`F()`**. In the listing, the function is used only when strings constants are to be output to the serial port.
In simple applications like the one shown in Listing 2, invocations of **`F()`** could be skipped. However, in more complex applications, invoking the function is a way to deal with the problem of extremely limited amount of RAM memory available in Arduino Uno (just 2 KB). The default compiler behavior is to place all data (including strings) in the precious RAM, even if some of the data items will not change for the entire lifetime of the application. Using **`F()`**, we make the compiler place the respective strings in the code memory. We also make the compiler invoke a

version of **print()** that retrieves a string from the code memory (before sending it to the serial port).

Arduino UNO can measure voltage, which makers it possible to take readings from all kinds of analog sensors. During the lab exercise, we will measure voltage from a rotary potentiometer. A way to connect the potentiometer to the board is presented in Fig. 6.

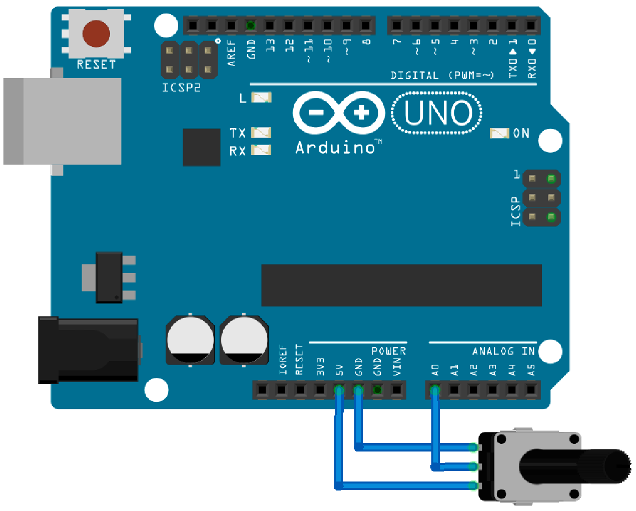


Fig. 6. Connecting a potentiometer to Arduino UNO

To measure the voltage, one should invoke the **analogRead()** function, with the identifier of the respective analog input pin as the argument (see Listing 3.).

```
int sensorValue = 0;
void setup() {
  Serial.begin(115200);
}
void loop() {
  sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(2);
}
```

Listing 3. Measuring input voltage with Arduino UNO

The built-in ADC converter, featured by the Arduino's microcontroller, has 6 inputs (A0, A1, A2, A3, A4, A5). Notably, **analogRead()** is a blocking function (no other code runs when analog-to-digital conversion takes place). While the conversion time is not too long, one should keep the blocking behavior in mind. The converter's resolution is 10 bits, and its reference voltage is 5V. One quantization interval is thus about 4,9mV (5V/1024). The reference voltage could be changed with **analogReference()**.

While working with voltage levels read from sensors, you might need to perform range mapping. Use the **map()** function, as exemplified in Listing 4 (there one converts a 10 bit value read from the converter to an 8 bit value).

```
int v1=analogRead(A0);
uint8_t v2=map(v1, 0, 1023, 0, 255);
```

Listing 4. Using **map()**

Code that you will create during the lab exercise should use only the following functions/classes from Arduino libraries:

- GPIO:

pinMode(), **digitalWrite()**, **digitalRead()**, **analogRead()**

- Serial port communication:

`Serial.begin()`, **`Serial.available()`**, **`Serial.read()`**, **`Serial.print()`**, **`Serial.println()`**, **`Serial.write()`**

- Getting system time and delaying program execution:

`millis()`, **`delay()`**

- General-purpose functions:

`map()`

The above functions are documented in the following file:
C:\Programs\Arduino\reference\www.arduino.cc\en\Reference\HomePage.html