Microprocessors and Embedded Controllers

EE5563

Laboratory Series 1

Introduction to microcontroller programming with Arduino

Alireza Mousavi, Alexandre Spieser, Morad Danishvar

Material for Laboratory Series 1 by Alexandre Spieser[1]

Summary

---

[1] – alexandre.spieser@brunel.ac.uk or a.spieser@gmail.com

INTRODUCTION

1.1   Overview

The Arduino microcontroller is an easy to use yet powerful single board computer that has gained considerable traction in the hobby and professional market.

The Arduino is open-source, which means hardware is reasonably priced and development software is free. This document covers the material in the course related to the Arduino Development Toolkit.
For further information you are advised to conducted self-research and reading with regard to Arduino development.

The Arduino project was started in Italy to develop low cost hardware for interaction design. An overview is on the Wikipedia entry for Arduino. The Arduino home page is http://www.arduino.cc/.

This guide covers the Arduino Uno board, a good choice for students and educators. With the Arduino board, you can write programs and create interface circuits to read switches and other sensors, and to control motors and lights with little effort.

Many of the pictures and drawings in this guide were taken from the documentation on the Arduino site, the place to turn if you need more information.

This is what the Arduino Uno board looks like.

The Arduino programming language is a simplified version of C/C++. If you know C, programming the Arduino will be familiar. If you do not know C, no need to worry as only a few commands are needed to perform useful functions.

An important feature of the Arduino is that you can create a control program on the host PC, download it onto the Arduino device and it will run automatically. Remove the USB cable connection to the PC, and the program will still run from the top each time you push the reset button. Remove the battery and put the Arduino board in a closet for six months. When you reconnect the battery, the last program you stored

will run. This means that you connect the board to the host PC to develop and debug your program, but once that is done, you no longer need the PC to run the program.

1.2   *What You Need for a Working System*

1. Arduino Duemilanove board
2. USB programming cable (A to B)
3. 9V battery or external power supply(for stand-alone operation)
4. Solderless breadboard for external circuits, and 22 solid wire for connections
5. Host PC running the Arduino development environment. Versions exist for Windows, Mac and Linux

1.3   *Installing the Software*

Follow the instructions on the Getting Started section of the Arduino website, http://arduino.cc/en/Guide/HomePage . Go all the way through the steps to where you see the pin 13 LED blinking. This is the indication that you have all software and drivers successfully installed and can start exploring with your own programs.

1.4   *Connecting a Battery*

For stand-alone operations the board is powered by a battery rather than through the USB connection to the computer. The external power can be anywhere in the range of 6 to 24 V (for example, you could use a car battery) or a standard 9V battery. While you could jam the leads of a battery snap into the Vin and Gnd connections on the board, it is better to solder the battery snap leads to a DC power plug and connect to the power jack on the board.

A possible suitable plug is part number 28760 can be found at www.jameco.com.

Warning:
Watch the polarity as you connect your battery to the snap as reverse orientation could blow out your board.

Disconnect your Arduino from the computer. Connect a 9 V battery to the Arduino power jack using the battery snap adapter. Confirm that the blinking program runs. This shows that you can power the Arduino from a battery and that the program you download runs without needing a connection to the host PC

1.5   *Moving On*

Connect your Arduino to the computer with the USB cable (*You do not need the battery for now. The green PWR LED will light).*
If there was already a program uploaded into the Arduino, it will run.
Warning:
Do not put your board down on a conductive surface; you will short out the pins on the back!

Start the Arduino development environment. In Arduino-speak, programs are called *"sketches"*, but here we will just call them programs.

In the editing window that comes up, enter the following program, paying attention to where semi-colons appear at the end of command lines.

```
void setup()
{
Serial.begin(9600);
Serial.println("Hello World");
}

void loop()
{}
```

Your window will look something like this:

Click the Upload button ⬛ or Ctrl-U to compile the program and load on the Arduino board.

Click the Serial Monitor button ⬛. If all has gone well, the monitor window will show your message and look something like this:

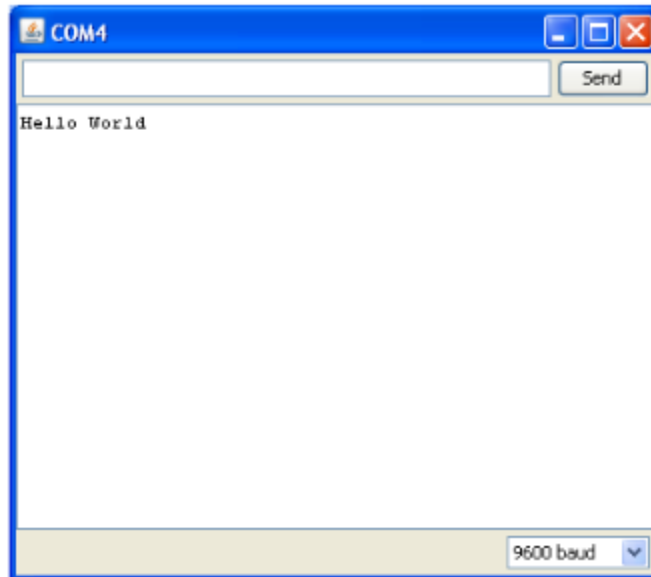Congratulations, you have created and run your first Arduino program!

Push the Arduino reset button a few times and see what happens.

Hint: If you want to check code syntax without an Arduino board connected, click the Verify button or Ctrl-R.

Hint: If you want to see how much memory your program takes up, Verify then look at the message at the bottom of the programming window.

2. *Writing Functions in Arduino*

```
void setup()
{
Serial.begin(9600);
Serial.println("Hello World");
}

void loop()
{}
```

The function `void setup(){}` contains the instructions that will be executed when the Arduino will be powered on.

The function `void loop(){}` contains the instruction to be executed in a loop when the Arduino has finished the setup instructions.

By creating a `setup()` function one initialises the program and sets the initial values to the ones specified by the programmer. The `loop()` function is used for repetitive operations through looping a set of instructions consecutively, allowing your program to change and respond. Use the `loop()` function to actively control the Arduino board.

Note the sample code below:

Anatomy of a C function

Datatype of data returned,
any C datatype.

"void" if nothing is returned.

Parameters passed to
function, any C datatype.

Function name

```
int myMultiplyFunction(int x, int y){

    int result;

    result = x * y;
    return result;
}
```

Return statement,
datatype matches
declaration.

Curly braces required.
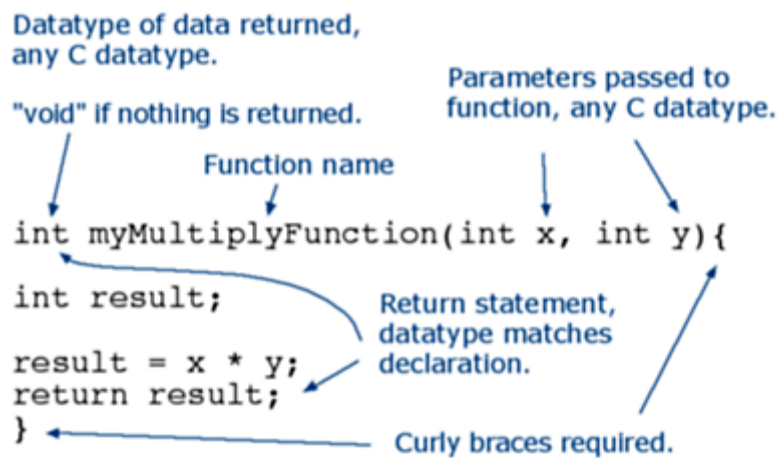
One can observe that the `loop()` and `setup()` are ***void*** which means that they do not return any values; and since there is nothing between the brackets they are not given any parameter.

The two required functions in an Arduino sketch are `setup()` and `loop()`. Other functions must be created outside the brackets of those two functions.

3. *Basics in C programming*

3.1 *if (conditional) and ==, !=, <, > (comparison operators)*

- **IF conditional statement**

The conditional statement "if" is used in conjunction with a comparison operator. The purpose of a conditional statement is to examine whether a certain condition has been met (e.g. an input being above a certain number) or otherwise. If the condition has been met, the statement returns Boolean value *"TRUE"*, and if the condition is not met (i.e. otherwise), the statement returns the Boolean value of *"FALSE".* The syntax of an "if" statement is:

```
if (someVariable > 50)
{
  // do something here
}
```

In the above example the program tests whether some variable is greater than 50 or not. If it is, the program takes a particular action. Put another way, if the statement in parentheses is true, the statements inside the brackets will run. If *FALSE*, the program skips over the code within the "if" statement curly brackets {*do something* }.

The curly brackets may be omitted after an "*if*" statement. If this is done, the next line (defined by the semicolon) becomes the only conditional statement. The conditional statements below are all correct and do the same thing.

```
if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)
digitalWrite(LEDpin, HIGH);

if (x > 120){ digitalWrite(LEDpin, HIGH); }

if (x > 120){
  digitalWrite(LEDpin1, HIGH);
  digitalWrite(LEDpin2, HIGH);
}                              // all are correct
```

The statements being evaluated inside the parentheses require the use of one or more operators:

- **Comparison Operators:**

```
 x == y (x is equal to y)
 x != y (x is not equal to y)
 x <  y (x is less than y)
 x >  y (x is greater than y)
 x <= y (x is less than or equal to y)
```

x >= y (x is greater than or equal to y)

Warning:

Beware of accidentally using the single equal sign (e.g. `if (x = 10)`). The single equal sign is the assignment operator, and sets x to 10 (puts the value 10 into the variable x). Instead use the double equal sign (e.g. `if (x == 10)`), which is the comparison operator, and tests *whether* x is equal to 10 or not. The latter statement is only true if x equals 10, but the former statement will always be true.

This is because C evaluates the statement `if (x=10)` as follows: 10 is assigned to x (the single equal sign is the assignment operator), so x now contains 10. Then the 'if' conditional evaluates 10, which always evaluates to TRUE, since any non-zero number evaluates to TRUE. Consequently, `if (x = 10)` will always evaluate to TRUE, which is not the desired result when using an 'if' statement. Additionally, the variable x will be set to 10, which is also not a desired action.

3.2 *Variables and Data types*

3.2.1 *Variables*

A variable is a way of naming and storing a value for later use by the program, such as data from a sensor or an intermediate value used in a calculation.

- **Declaring Variables**

Before they are used, all variables have to be declared. Declaring a variable means defining its type, and optionally, setting an initial value (initializing the variable).

Variables do not necessarily need to be initialized (assigned a value) when they are declared, but it is often useful. The snippets of code below are both correct.

```
int inputVariable1;
int inputVariable2 = 0;
```

Programmers should consider the size of the numbers they wish to store in choosing variable types. Variables will roll over when the value stored exceeds the space assigned to store it. See below for an example.

- **Variable Scope**

Another important choice that programmers face is where to declare variables. The specific place that variables are declared influences how various functions in a program will *see* the variable. This is called variable scope.

- **Initializing Variables**

Variables may be *initialized* (assigned a starting value) when they are declared or not. It is always good programming practice however to double check that a variable has valid data associated with it; before it is accessed for some other purpose.

Example:

```
 int calibrationVal = 17;  // declare calibrationVal and set initial value
```

- **Variable Rollover**

When variables are made to exceed their maximum capacity they "roll over" back to their minimum capacity, note that this happens in both directions.

```
   int x
   x = -32,768;
   x = x - 1;        // x now contains 32,767 - rolls over in neg. direction
   x = 32,767;
   x = x + 1;        // x now contains -32,768 - rolls over
```

- **Using Variables**

Once variables have been declared, they are used by setting the variable equal to the value one wishes to store with the assignment operator (single equal (=) sign). The assignment operator tells the program to put whatever is on the right side of the equal sign into the variable on the left side.

```
inputVariable1 = 7; // sets the variable named inputVariable1 to 7
inputVariable2 = analogRead(2); // sets the variable named inputVariable2
                                // to the (digitized) input voltage read
                                // from analog pin #2
```

Examples

```
int lightSensVal;
char currentLetter;
unsigned long speedOfLight = 186000UL;
char errorMessage = {"choose another option"}; // see string
```

Once a variable has been set (assigned a value), you can examine if its value meets a certain condition, or you can use its value directly. For instance, the following code tests whether the inputVariable2 is less than 100, then sets a delay based on inputVariable2 which is a minimum of 100:

```
if (inputVariable2 < 100)
{
  inputVariable2 = 100;
}

delay(inputVariable2);
```

This example shows all three useful operations with variables. It tests the variable (`if (inputVariable2 < 100)`), it sets the variable if it passes the test (`inputVariable2 = 100`), and it uses the value of the variable as an input parameter to the delay() function (`delay(inputVariable2)`)

Style Note: You should give your variables descriptive names, so as to make your code more readable. Variable names like **tiltSensor** or **pushButton** help you (and anyone else reading your code) to understand what the variable represents. Variable names like **var** or **value**, on the other hand, do little to make your code readable.

You can name a variable any word that is not already one of the keywords (Reserved Words) in Arduino. Avoid beginning variable names with numeral characters.

3.2.2 *Data types*

For our application, the use of standard C functions is required, but one shall be aware of the different types of data prior to any coding:

In the C programming language, data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines

how much space it occupies in the storage area (memory) of the device and how the bit pattern stored is interpreted.

The types in C can be classified as follows:

Types and Description
Basic Types: They are arithmetic types and consists of the two types: (a) integer types and (b) floating-point types.
Enumerated types: They are again arithmetic types and they are used to define variables that can only be assigned certain discrete integer values throughout the program.
The type void: The type specifier *void* indicates that no value is available.
Derived types: They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

The array types and structure types are referred to collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see basic types in the following section, whereas, other types will be covered in the later chapters.

3.2.2.1 *Integer Types*

Following table gives you details about standard integer types with its storage sizes and value ranges:

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647 (long int)
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767

| unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** operator. The expressions *sizeof(type)* yields the storage size of the object or type in bytes. Following is an example to get the size of int type on any machine:

On the Arduino platform, if you type the following sketch…

```
void setup() {
   // start serial port at 9600 bps:
  Serial.begin(9600); // start serial communication
   if(DEBUG){
   Serial.println("Serial.begin ok \n"); // says serial communication is ok
   Serial.println("Size of an int: ");
   Serial.println(sizeof(int));
   Serial.println("\n");
   }
}

void loop(){
   // nothing to do here
}
```

…and open the serial communication terminal, the following message will appear:
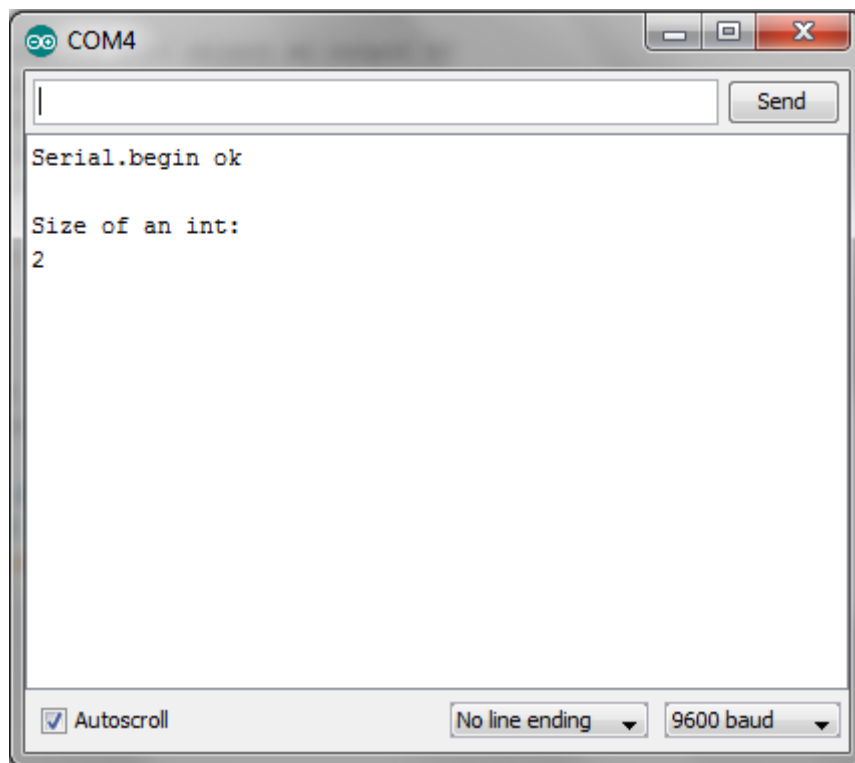
As you can see the size of an int (integer) is 2, which means its take 2 bytes (16 bits) in the memory of the Arduino.

3.2.2.2 *Floating-Point Types*

Following table gives you details about standard floating-point types with storage sizes and value ranges and their precision:

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

3.2.2.3 *The void Type*

The void type specifies that no value is available. It can be used for three different situations:

Types and Description
Function returns as void There are various functions in C which do not return value or you can say they return void. A function with no return value has the return type as void. For example **void exit (int status);**
Function arguments as void There are various functions in C which do not accept any parameter. A function with no parameter can accept as a void. For example, **int rand(void);**
Pointers to void A pointer of type void * represents the address of an object, but not its type. For example a memory allocation function **void *malloc(size_t size);** returns a pointer to void which can be casted to any data type.

3.2.2.4 *Derived type (arrays)*

An array is a collection of variables that are accessed with an index number. Arrays in the C programming language, on which Arduino is based, can be complicated, but using simple arrays is relatively straightforward.

An array of 6 integers can be visually represented as follows:

Array Values	2	4	8	3	6	-5
Array Index Value	0	1	2	3	4	5

Notice that the index of the first array cell is 0, not 1. The array index is used to access each of the cells (element) of the array.

An array is identified by its name and can only contain one type of data (e.g. int, byte, char, etc…).

- **Creating (Declaring) an Array**

All of the methods below are valid ways to create (declare) an array.

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hello";//creates an array of chars, also called string
```

Note: One can declare an array without initializing it as in myInts.

In myPins we declare an array without explicitly choosing a size. The compiler counts the elements and creates an array of the appropriate size.

Finally, you can both initialise and size your array, as in mySensVals. Note that when declaring an array of type char, one more element than your initialisation is required, to hold the required null character.

- **Accessing an Array**

Arrays are **zero indexed**, that is, referring to the array initialisation above, the first element of the array is at index 0, hence

`mySensVals[0] == 2, mySensVals[1] == 4`, and so forth.

It also means that in an array with ten elements, index nine is the last element. Hence:

```
int myArray[10]={9,3,2,4,3,2,7,8,9,11};
    // myArray[9]    contains 11
    // myArray[10]   is invalid and contains random information (other
memory address)
```

For this reason you should be careful in accessing arrays.

Accessing past the end of an array (using an index number greater than your declared array size - 1) is reading from memory that is in use for other purposes. Reading from these locations is probably not going to do much except yield invalid data.

Writing to random memory locations is definitely a bad idea and can often lead to unexpected results such as crashes or program malfunction. This can also be a difficult bug to track down.

Unlike BASIC or JAVA, the C compiler does no check whether the array access statement is within the legal bounds of the array size that has been declared.

**To assign a value to an array:** `mySensVals[0] = 10;`

To retrieve a value from an array: `x = mySensVals[4];`

- **Array manipulation: Arrays and FOR Loops**

The "`for`" statement is used to repeat a block of statements enclosed in curly brackets. An increment counter is usually used to increment and terminate the loop. The "`for`" statement is useful for any repetitive operation and is often used in combination with the arrays to collect the information from the data sources (e.g. pins).

There are three parts to the "`for`" loop header:

```
for (initialization; condition; increment) {
//statement(s);
}
```
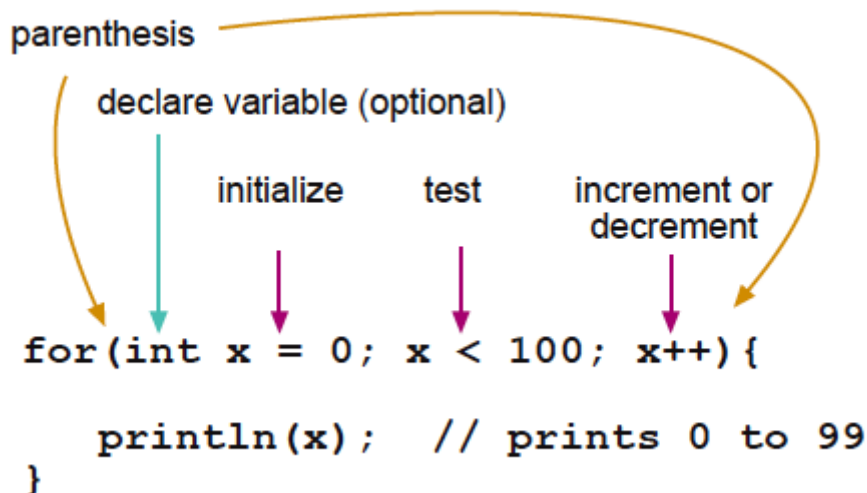

The initialisation happens first and exactly once. Each time through the loop, the condition is tested; if it's true, the statement block, and the increment is executed, then the condition is tested again. When the condition becomes false, the loop ends.

Arrays are often manipulated inside the "`for`" loops, where the loop counter is used as the index for each array element. For example, to print the elements of an array over the serial port, you could do something like this:

```
int i; // creates (declares) our array index
for (i = 0; i < 5; i++) {
  Serial.println(myPins[i]);
```

```
}
```

`i++` is an expression stating that the integer `i` – used as the array index – is incremented by 1 each time the statement in the for loop as been executed. It is equivalent to writing the statement `i = i + 1`.

When `i` reaches the value 5, the condition for the "`for`" loop to execute is not valid anymore so the loop is terminated.

4. <u>*Troubleshooting*</u>

If there is a syntax error in the program caused by a mistake in typing, an error message will appear in the bottom of the program window. Generally, staring at the error will reveal the problem. If you continue to have problems, try these ideas:

- Run the Arduino program again
- Comment your code to see what you are doing more clearly and help others understand your programme (formatting: instruction; //comment)
- Check that the USB cable is secure at both ends.
- Reboot your PC because sometimes the serial port can lock up
- If a "Serial port...already in use" error appears when uploading close other applications that might be talking to Arduino (i.e. LabVIEW programme).

5. *The setup*

5.1 *TinkerKit sensor and I/O shield*
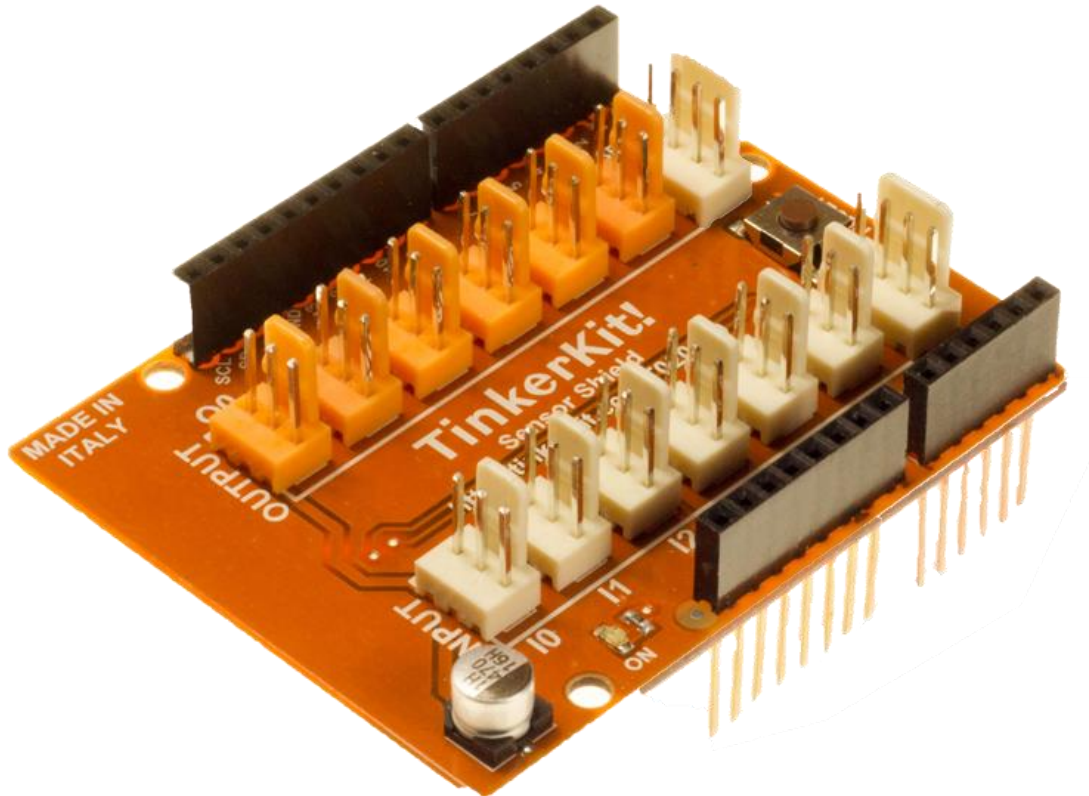
The Sensor Shield allows you to hook up the TinkerKit sensors and actuators directly to the Arduino, without the use of the breadboard.

It has 12 standard TinkerKit 3pin connectors. The 6 labelled **I0** through **I5** are **Analog Inputs**. The ones labelled **O0** through **05** are **Analog Outputs** connected to the PWM capable outputs of the Arduino Board (it is possible to change these to Digital Inputs, in which case they will report either HIGH or LOW, but nothing in between).

- Pin **11** on the Arduino is **O0** on the shield.
- Pin **10** on the Arduino is **O1** on the shield.
- Pin **9** on the Arduino is **O2** on the shield.
- Pin **6** on the Arduino is **O3** on the shield.
- Pin **5** on the Arduino is **O4** on the shield.
- Pin **3** on the Arduino is **O5** on the shield.

Module description: A green LED signals that the shield is correctly powered and a standard 6mm pushbutton allows you to RESET the board.

The **4pin TWI socket** allows communication to any device supporting the I2C protocol through the Wire library on Arduino. 5V and Ground are provided on the socket. Note that on the Arduino I2C bus uses Analog Input 4 and 5, using the TWI connection precludes the use of those analog inputs.

The **4pin SERIAL socket** allows the board to communicate with other devices that support serial communication. 5V and Ground are provided on the socket for your convenience.
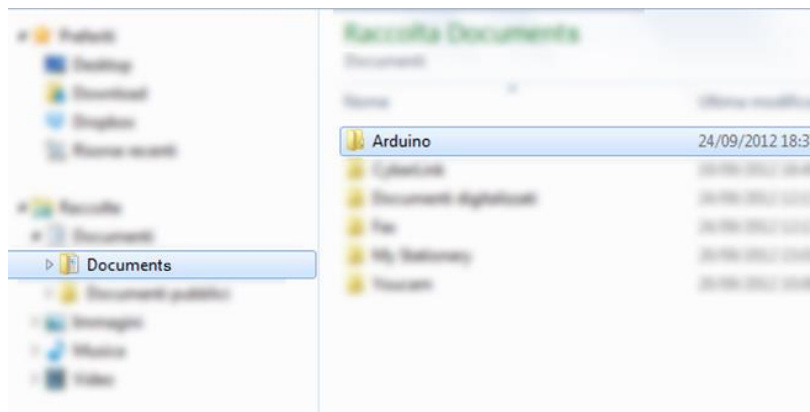
Note: If you are sending or receiving data to and from the computer this serial connector is not available.

## 5.2   *Importing the TinkerKit Library (if necessary)*

Unzip the downloaded TinkerKit.zip
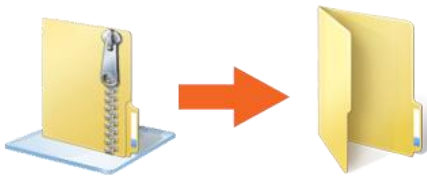
Find the Arduino folder that is located inside the Documents folder. This folder is generated automatically when you open the Arduino software for the first time


If you already installed other libraries, there should be a "libraries" folder inside the Arduino folder. If this is not the case, just create a new folder and name it "libraries":
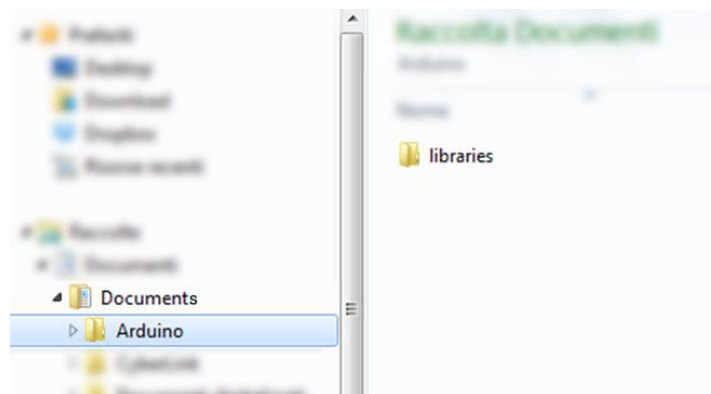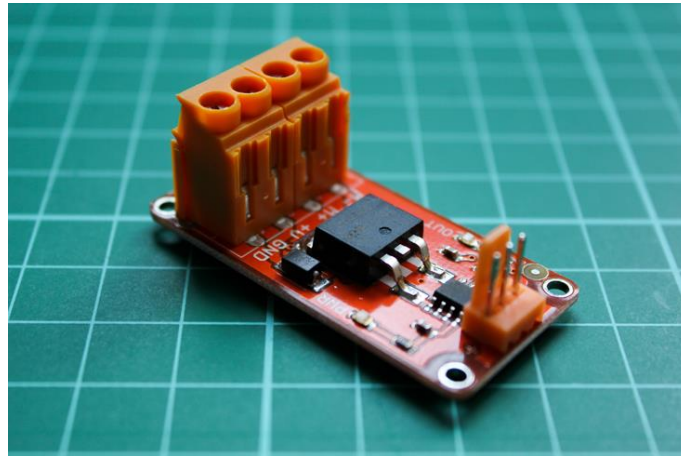
Now copy the unzipped library that you just downloaded inside the "libraries" folder
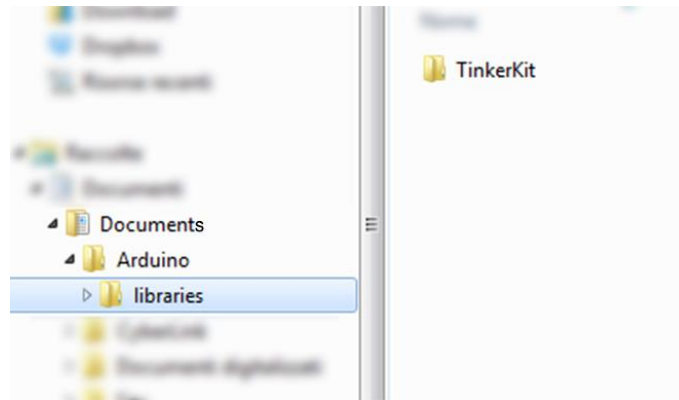
Almost done! We just have to make sure that the library is working. Open (or close and relaunch) the Arduino Software and in the top menu under Sketch–>Import Library.. you should see TinkerKit like in the picture below
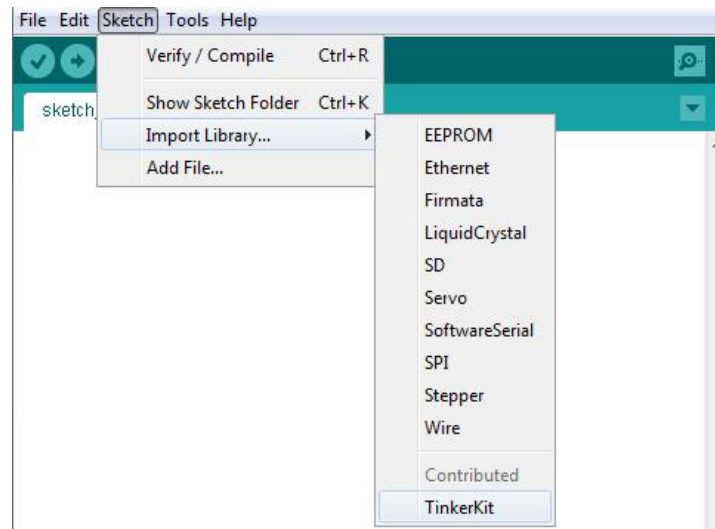
5.3   *TinkerKit MOSFET module*

This module switches a high current load using a high power transistor. Unlike a mechanical relay, this is capable of high speed switching for use with PWM.

Output: This module lets you control devices operating at a maximum of 24VDC with an Arduino pin. To wire the module, connect the power supply for your device (max 24 V) to the V+ and GND terminals. Connect the device to M+ and M-.

Be aware of your circuit's polarity, you could damage your components if it is not wired correctly.

Module Description: This module features a power MOSFET transistor (IRFS3806 from International Rectifier), a kick-back (free wheeling) diode, a standard TinkerKit 3pin connector, a signal amplifier (LMV358 from ST), a green LED that signals that the module is correctly powered and one yellow LED whose brightness depends on the input signal received by the module.

MOSFET Specifications (IRFS3806)	
Parameter	Value
Package	D2-Pak
Circuit	Discrete
VBRDSS (V)	60
VGs Max (V)	20
RDS(on) Max 10V (mOhms)	15.8
ID (drain current)@ TC = 25C (A)	43
ID (drain current) @ TC = 100C (A)	31
Tj Max	175
Power Dissipation @ TC = 25C (W)	71
Part Status	Active

Available methods

```
TKMosFet mos(O0);
```

on( )   switch the MosFet on

off( )   switch the MosFet off

state( )   return the value of the current MosFet state

write(value)
- passing HIGH or LOW value switch the MosFet on and off

- passing an analog value from 0 to 1023 you can vary the PWM duty cycle

Copy the following code into the Arduino IDE

```
#include <TinkerKit.h>    // include the TinkerKit library

TKMosFet mos(O0);          //create the mos object at output O0

void setup() {
  //nothing here
}

void loop()
{
  mos.on();        //assign the values to the mosfet
  delay(10);       //rest for 10 milliseconds.
  mos.off();       // switch the mosfet off
  delay(1000);     //rest for 1000 milliseconds
}
```
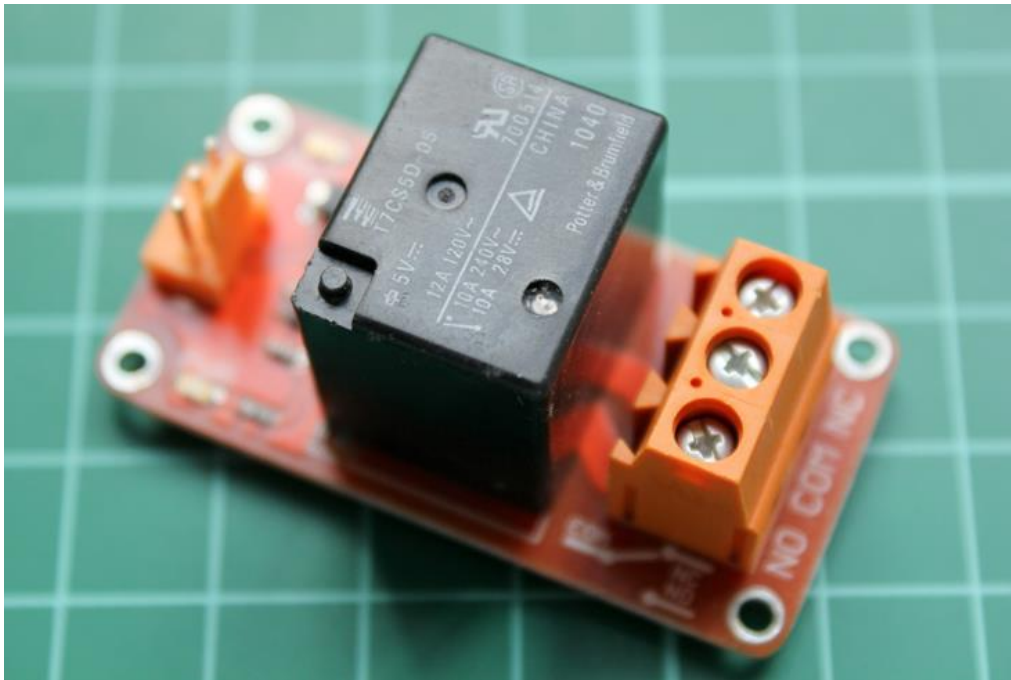
You should see the "OUT" LED of the MOSFET module blink every 1 second. This means that the MOSFET is switched ON for 10ms and then switched off for 1000.
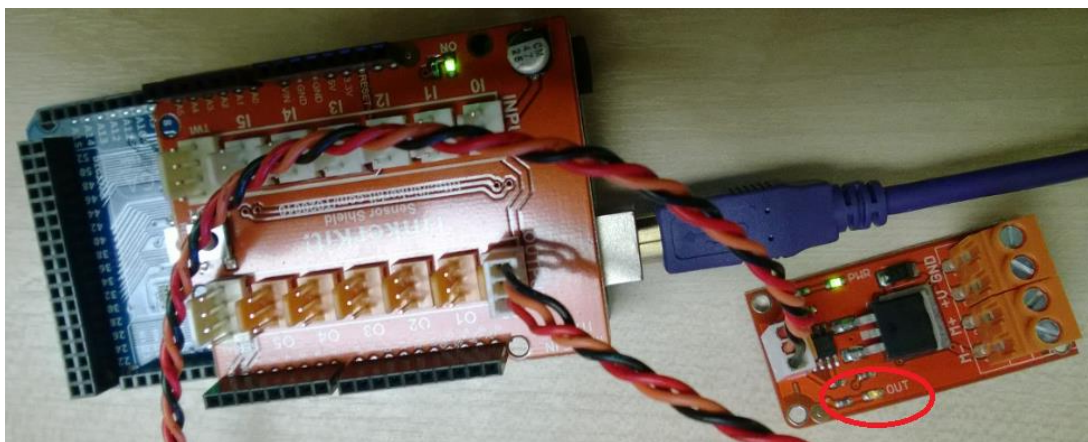
5.4   *TinkerKit relay module*

A **relay** is an **electrically operated switch** that allows you to turn on or off a circuit using voltage and/or current much higher than the Arduino could handle. There is no connection between the low voltage circuit operated by Arduino and the high power circuit. The relay protects each circuit from each other.

Warning: We don't recommend you operate circuits powered at more than 24V without the supervision of an expert.

Input: The relay is a simple mechanical on/off switch. It activates when the input reaches 5V and turns off when the input is 0v. You can control it through the digitalWrite( ) function on Arduino.

The module provides three connections labeled **COM**, **NC** and **NO**. **NC** stands for "**NORMALLY CLOSED**". This means that when the relay has no signal (LOW or 0V from an Arduino), the connected circuit will be active; conversely, if you apply 5V or pull the pin HIGH, it will turn the connected circuit off. **NO** stands for "**NORMALLY OPEN**", and functions in the opposite way; when you apply 5V the circuit turns on, and at 0V the circuit turns off. Relays can replace a manual switch. Remove the switch and connect its wires to **COM** and **NO**. When the relay is activated the circuit is closed and current can flow to the device you are controlling.

Module Description: this module features an 250v 10A mounted on a 2 module TinkerKit board, one standard TinkerKit 3pin connector, one transistor, a green LED that signals that the module is correctly powered and an yellow LED that indicates when the relay is active.

Available methods

```
TKRelay relay(O0);
```

on()   switch the Relay on

off()   switch the Relay off

state()   return the boolean value of the current Relay state

Code example:

```
#include <TinkerKit.h>    // include the TinkerKit library

TKMosFet mos1(O0);  //create the mos1 object at output O0
TKRelay relay1(O1);  //create the relay1 object at output O1


void setup() {
  //nothing here
}

void loop()
{
  mos1.on();        //assign the values to the mosfet
  relay1.off();    // switch the relay off
  delay(1000);      //rest for 1000 milliseconds.
  mos1.off();      // switch the mosfet off
  relay1.on();     // switch the relay on
  delay(1000);    //rest for 1000 milliseconds
}
```

If the MOSFET module is connected to O0 and the relay module is connected to O1, you can see - by checking the "OUT" LED - (and hear) the relay being switched on when the MOSFET is being turned off and vice versa.

5.5 *TinkerKit LED module*

This module is made of a big RED LED which can be used to signal a failure or that a programme is running on the controller.

We will write a programme that will make the LED glow by using the PWM capabilities of the Arduino microcontroller.

The PWM duty cycle can be set by using the write ( ) function of the TinkerKit library. If you don't want to use this library, you can also use the Arduino function analogWrite ( ).

```
#include <TinkerKit.h>    // include the TinkerKit library

TKLed LED(O2);              //create the LED object at output O2
int i;
void setup() {
  //nothing here
}
```

```
void loop()
{
  for(i=0; i<500;i++) {
    LED.write(i); // apply PWM of duty cycle i
    delay(5);
  }
  for(i=500; i>=0; i--) {
    LED.write(i); // apply PWM of duty cycle i
    delay(5);
  }
}
```

Using the standard Arduino `analogWrite()` function, the code will look like that:

```
#include <TinkerKit.h>    // include the TinkerKit library

int i;
void setup() {
  //nothing here
}

void loop()
{
// increase duty cycle progressively in this for loop
  for(i=0; i<124;i++) {
    analogWrite(O2,i); // apply PWM of duty cycle i at O2
    delay(10); // 10 ms delay
  }
// decrease duty cycle progressively in this for loop
  for(i=125; i>=0; i--) {
    analogWrite(O2,i); // apply PWM of duty cycle i at O2
    delay(10); // 10 ms delay
  }
}
```

In both cases, you can notice the brightness of the LED progressively increasing and decreasing over time. The duty cycle in increased until it approaches 50% and is then decreased. The delay function set the speed at which we went the task to be performed.

6. <u>Actuating elements using the Serial Terminal.</u>

In this section, we are going to do something relatively more complex. In many application (industrial or not) the user wants to control or communicate with a device using a human machine interface (HMI). In this exercise we will switch on/off the relay and the MOSFET by sending commands to the microcontroller, which will achieve the desired outcome. For this purpose, we will use the serial communication terminal of the Arduino IDE.

6.1 <u>Parameters and command formatting</u>

The first step is to think about how we can tell the controller what we want it to do? A simple way to achieve this is to send the desired parameters to the controller. The values of these parameters will affect the instructions that the microcontroller will perform. Usually, these parameters are given in a specific order so that the controller can map them to their respective entity. Also, the parameters are usually separated by a character, which will help the controller identify the values of these parameters.

In our example it has been decided that the parameters would be separated by a comma ',' and that the end of the instruction would be marked by a full stop '.'. Furthermore, the first parameter will be assigned to the MOSFET and the second one to the relay. Therefore, 4 different commands can be sent:

| Command syntax	Meaning
0,0.	MOSFET OFF, RELAY OFF
0,1.	MOSFET OFF, RELAY ON
1,0.	MOSFET ON, RELAY OFF
1,1.	MOSFET ON, RELAY ON.

6.2   *Gathering the data*

In this section, we are going to make a program that gathers the data sent from a computer via serial.

We will send our commands and ask for the microcontroller to write them back to us, to make sure that the message has been received properly.

```
void setup() {
   // start serial port at 9600 bps:
  Serial.begin(9600); // start serial communication
   if(DEBUG){
    Serial.println("Serial.begin ok \n");
    }
}

void loop()
{
   //wait for parameters
  if (Serial.available() <= 0) {
      delay(100); // just wait
  } else if (Serial.available() > 0){ // if parameters have been sent
 char aChar = Serial.read(); // set aChar to the value of the read
                            // character
 Serial.println(aChar);
 }

}
```

Now let us send the word "hello" and see what the reply of the microcontroller is…
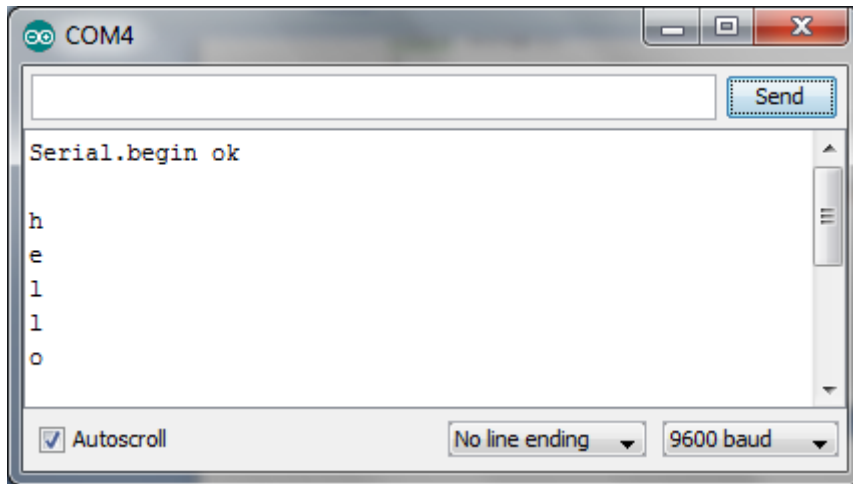
We can see that the controller behaves accordingly, but the way we are handling the input data is not very nice for our application because the controller treats the message character by character and not as a whole message.

Therefore in this next example we will make the controller store the input characters in an array to create a string that can then be printed out and processed. We also need a character to define when to end our message, so we will use the full stop '.' character for this purpose.

```
int DEBUG = 1;
char inData[200]; // declares an array of 200 chars
int indexChar = 0; // declares the index of the array inData as an integer
char aChar;
int i;
void setup() {
  Serial.begin(9600); // start serial port at 9600 bps
  Serial.println("Serial communication started");
}

void loop()
{
   //wait for parameters
  if (Serial.available() <= 0) { // if no data
     delay(100); // just wait
  } else if (Serial.available() > 0){ // if parameters have been sent
    aChar = Serial.read(); // read incoming char value and set aChar to it
    if(aChar != '.'){ // if the read char is different from a full stop
    inData[indexChar] = aChar; // gives the array cell the value of aChar
    indexChar++; // increment the array index for the next char to come
    inData[indexChar] = '\0'; // add "end of string" null character
                              // each time
    }else{ // if the read char is a full stop
    inData[indexChar] = aChar; // gives the array cell the value of aChar
    indexChar++; // increment the array index for the next char to come
    inData[indexChar] = '\0'; // add "end of string" null character
    Serial.println(inData); // print the string out
      for(i=0; i<indexChar;i++){
      inData[i] = '\0'; // empties char buffer
      }
    indexChar = 0; // reset char buffer index for next parameters

    } // end of of if(aChar == '.')
  }// end of else if (Serial.available() > 0)
```

```
}// end of void loop
```

Let us check what the controller is responding if we send "hello."…
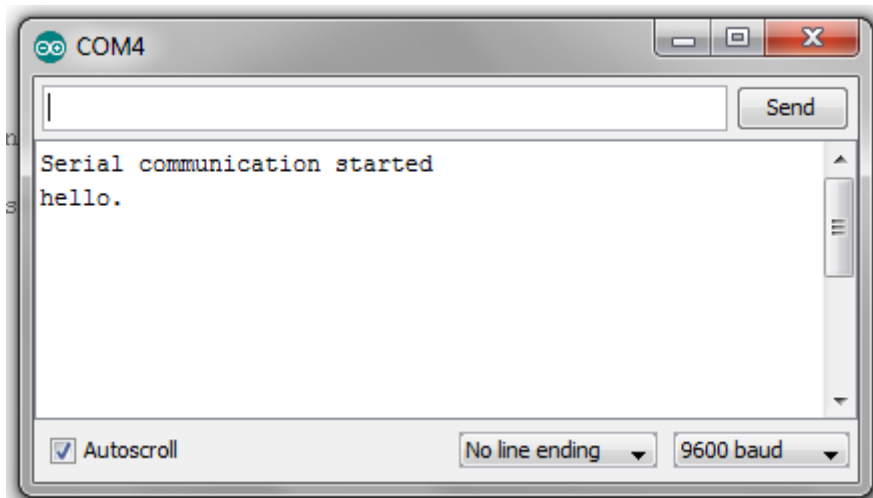
Mission accomplished!

You can also write sentences like "This is my sentence." and as long as you do not forget the '.' or do not go over the 200 character limit of the `inData` array the controller will do its work properly… This means that the controller is ready to receive our instructions like "**0,0.**" and so on.
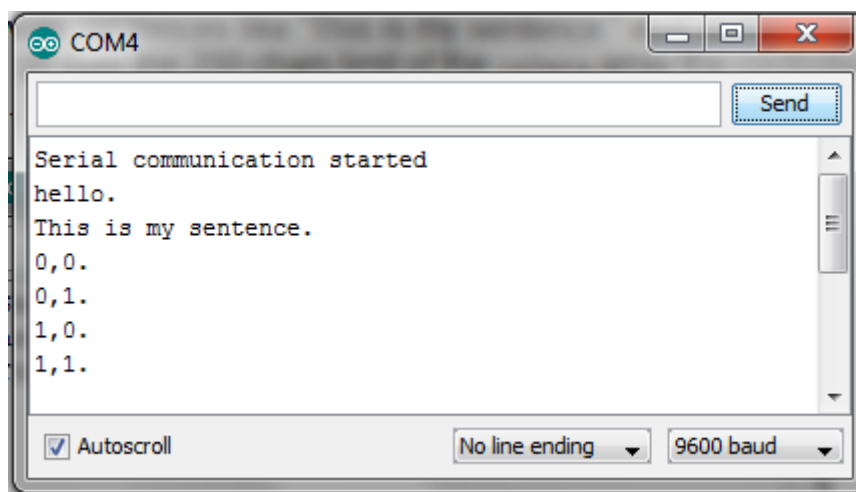
Also, it is worth knowing that an "end of message" character can be automatically added by the serial communication terminal running on the computer. If you click on the drop down menu "No line ending" you will see different options appearing:
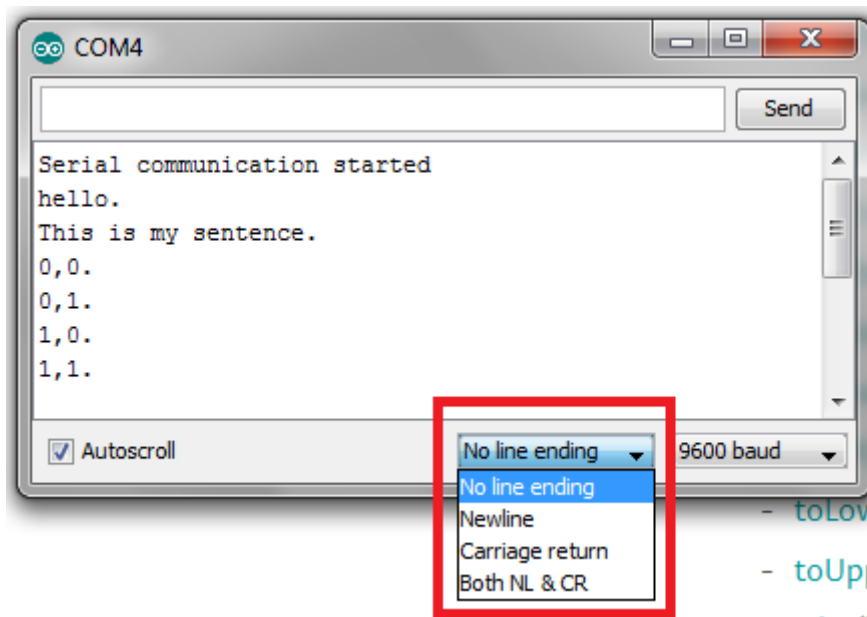
These options allow you to setup your serial communication terminal to add a specific character at the end of the message.

- Newline (NL) adds the character '\n'
- Carriage return (CR) adds '\r'
- And Boths NL & CR adds '\r\n'

You could therefore set your conditional statements with a CR setup like so:

```
int DEBUG = 1;
char inData[200]; // declares an array of 200 chars
int indexChar = 0; // declares the index of the array inData as an integer
char aChar;
int i;
void setup() {
  Serial.begin(9600); // start serial port at 9600 bps
  Serial.println("Serial communication started");
}

void loop()
{
  //wait for parameters
  if (Serial.available() <= 0) { // if no data
     delay(100); // just wait
  } else if (Serial.available() > 0){ // if parameters have been sent
    aChar = Serial.read(); // read incoming char value and set aChar to it
    if(aChar != '\r'){ // if the read char is different from a CR
    inData[indexChar] = aChar; // gives the array cell the value of aChar
    indexChar++; // increment the array index for the next char to come
    inData[indexChar] = '\0'; // add "end of string" null character
                             // each time
    }else if(aChar == '\r'){ // if the read char is a CR
    Serial.println(inData); // print the string out
      for(i=0; i<indexChar;i++){
      inData[i] = '\0'; // empties char buffer
      }
    indexChar = 0; // reset char buffer index for next parameters

    } // end of of if(aChar == '.')
```

```
  }// end of else if (Serial.available() > 0)
}// end of void loop
```

6.3   *Converting characters to integers*

In our previous example note that the data sent to the controller has a char type, which is the default data type for serial communication in most of the serial communication terminals. To be able to conduct mathematical operations and compare numbers it is therefore necessary to convert the char (or string) to an integer or a float data type.

For example, we want to convert the string "1000" into an integer whose value is 1000.

We can use the following C functions for this purpose:

- **atoi** : `int atoi (const char * str);`

The `*` of `char * str` means that the `atoi` function expects a pointer towards characters. Arrays of characters (strings) are also pointers so they can be processed by this function.

This function converts a string to an integer: it parses the C-string *str* interpreting its content as an integral number, which is returned as a value of type int.

Example:

```
int myInt = 0; // create an int
char string[5] = "1000"; // create a string with value "1000"
myInt = atoi (string); // convert the string into an int and set myInt to
                       //the converted value
Serial.println(myInt); // send the new value of myInt via serial com.
```

- **atof**: `double atof (const char* str);`

Convert string to double: Parses the C string *str*, interpreting its content as a floating point number and returns its value as a double.

```
int myInt = 0;
char string[5] = "1000";
myInt = atoi (string);
Serial.println(myInt);
```

Now we are going to convert the received characters into integers:

```
#include <TinkerKit.h>    // include the TinkerKit library

byte DEBUG = 1;  // used for setting the debugging messages
int indexNumber = 0; // create an integer and initialize it to 0
int indexChar = 0; // create an integer
int inParams[2]; // creates an array of 2 ints
char inData[6]; // creates an array of 6 characters
int i;

void setup() {
// start serial port at 9600 bps:
Serial.begin(9600); // start serial communication
   if(DEBUG){
   Serial.println("Serial communication started ");
   }
}

void loop()
{
   //wait for parameters
  if (Serial.available() <= 0) {
     delay(100); // just wait
  } else if (Serial.available() > 0){ // if parameters have been sent
  char aChar = Serial.read(); // reads string, char by char
    if (aChar == ','){ //fill in next parameter if you meet a ',' or a'.'
    inParams[indexNumber] = atol(inData); // Convert char array to long
                                          //integer
      for(i=0; i<=indexChar;i++){
      inData[i] = '\0'; // empties char buffer
      }
    indexChar = 0; // reset buffer index for next parameter
    indexNumber++; // increment parameter index
    }
    if (aChar != ',' && aChar != '.'){ // if a char is different from ','
                                       //put it in inData
    inData[indexChar] = aChar;
    indexChar++;
    inData[indexChar] = '\0'; // add "end of string" character at each
iteration
    }
    if (aChar == '.'){ //fill in last parameter if you meet a'.'
    inParams[indexNumber] = atoi(inData); // Convert char array to
                                          // integer and store it inside
                                         // the integer array "inParams"
      for(i=0; i<=indexChar;i++){
      inData[i] = '\0'; // empties char buffer
      }
```

```
    indexChar = 0; // reset char buffer index for next parameters
Serial.println("Value of the first parameter: %i", inParams[0]);
Serial.println("Value of the second parameter: %i", inParams[1]);

    // get ready for next time, so clean everything
      for(i=0; i<=indexNumber; i++){
      inParams[i] = 0;
      } // end of for(i=0; i<=indexNumber; i++)
    indexNumber = 0;
    inData[indexChar] = '\0';
    }// end of if (aChar == '.')
  }// end of if(Serial.available()>0
}// end of loop()
```

Let us see the result:
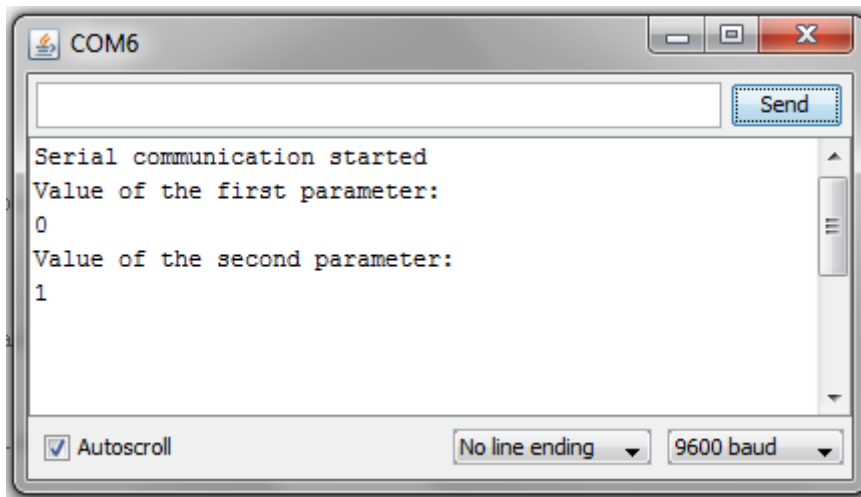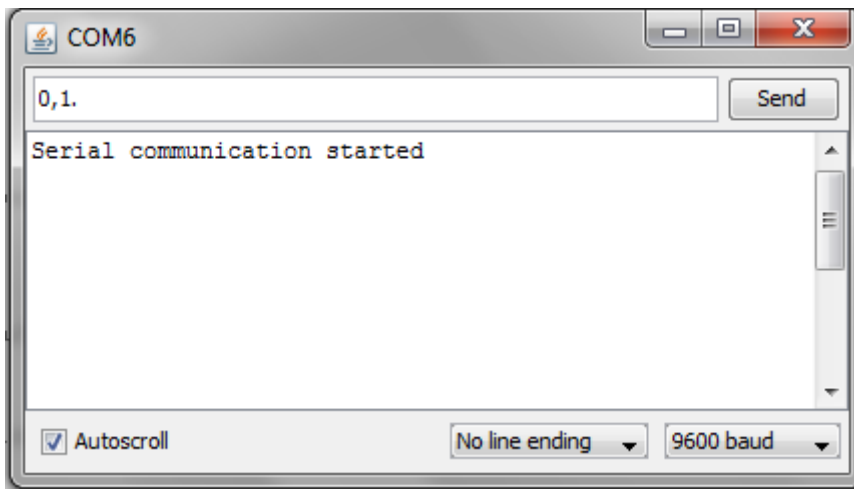
6.4   *Final programme to recap what we have learnt*

We should now have all the elements necessary to achieve our main goal. We know how to gather the required parameters and how to convert them into integers. The only thing left to do is to take appropriate actions according to the entered values.

Command syntax	Meaning
0,0.	MOSFET OFF, RELAY OFF
0,1.	MOSFET OFF, RELAY ON
1,0.	MOSFET ON, RELAY OFF
1,1.	MOSFET ON, RELAY ON.

For this, we will create a function that will take an array of integers as parameter and will execute the desired instructions. We will name this function "`ActuateComponents`".

Since we need to give an array of parameters to this function, we must declare it as follows:

```
void ActuateComponents(int *Parameters){ //declaration of the function

     //code of the function

}// end of the function
```

You can observe that this function takes a pointer towards integers as a parameter. This pointer is the array containing the integers needed to decide what needs to be actuated.

The integer contained in the first cell of the array will represent the desired status of the MOSFET. So if `Parameters[0]` is equal to 0, the MOSFET must be switched OFF and if `Parameters[0]` is equal to 1, the MOSFET must be switched ON.

The same logic goes for the relay, whose status will be decided according to the second cell `Parameters[1]` of the array.

The MOSFET is switched ON by executing the command : `mos1.on ();`

…and switched OFF with the command: `mos1.off();`

Where `mos1` corresponds to the name of the MOSFET that we have declared in our programme.

We can use conditional statements to check the values of the given parameters:

```
void ActuateComponents(int *Parameters){ //declaration of the function

  if(DEBUG){Serial.println("Function ActuateComponents called \n");}

      if(Parameters[0] == 0 && Parameters[1] == 0){
      mos1.off();        // switch mosfet off
      relay1.off();      // switch the relay off
       if(DEBUG){Serial.println("MOSFET OFF, RELAY OFF \n");}
      }else if(Parameters[0] == 1 && Parameters[1] == 0){
      mos1.on();         // switch mosfet on
      relay1.off();      // switch the relay off
       if(DEBUG){Serial.println("MOSFET ON, RELAY OFF \n");}
      }else if(Parameters[0] == 0 && Parameters[1] == 1){
      mos1.off();        // switch mosfet off
      relay1.on();       // switch the relay on
       if(DEBUG){Serial.println("MOSFET OFF, RELAY ON \n");}
      }else if(Parameters[0] == 1 && Parameters[1] == 1){
      mos1.on();         // switch mosfet on
      relay1.on();       // switch the relay on
       if(DEBUG){Serial.println("MOSFET ON, RELAY ON \n");}
      }

}// end of the function
```

Our function is ready. Now we need to insert it into our code and decide when and how we want to call it. We also need to allocate the array of parameters for this function. Previously, we stored the parameters inside the integer array `inParams`. We will therefore give this array to the function `ActuateComponents`.

The function…

```
#include <TinkerKit.h>    // include the TinkerKit library

TKMosFet mos1(O0);  //create the mos1 object at output O0
TKRelay relay1(O1);  //create the relay1 object at output O1
byte DEBUG = 1;  // used for setting the debugging messages
int indexNumber = 0; // create an integer and initialize it to 0
int indexChar = 0; // create an integer
int inParams[2]; // creates an array of 2 ints
char inData[6]; // creates an array of 6 characters
int i;

void setup() {
  Serial.begin(9600); // start serial communication at 9600 bps:
   if(DEBUG){
   Serial.println("Serial communication started");
   }
}

void loop()
{
   //wait for parameters
  if (Serial.available() <= 0) {
     delay(100); // just wait
  } else if (Serial.available() > 0){ // if parameters have been sent
  char aChar = Serial.read(); // reads string, char by char
    if (aChar == ','){ //fill in next parameter if you meet a ',' or a'.'
```

```
      inParams[indexNumber] = atoi(inData); // Convert char array to integer
        for(i=0; i<=indexChar;i++){
        inData[i] = '\0'; // empties char buffer
        }
      indexChar = 0; // reset buffer index for next parameter
      indexNumber++; // increment parameter index
      }
      if (aChar != ',' && aChar != '.'){ // if a char is different from ','
and '.' put it in inData
      inData[indexChar] = aChar;
      indexChar++;
      inData[indexChar] = '\0'; // add "end of string" character
                                // at each iteration
      }
      if (aChar == '.'){ //fill in last parameter if you meet a'.'
      inParams[indexNumber] = atoi(inData); // Convert char array to integer
        ActuateComponents(inParams);
      // get ready for next time, so clean everything
        Reset();
      }// end of if (aChar == '.')
    }// end of if(Serial.available()>0
}// end of loop()

void ActuateComponents(int *Parameters){ //declaration of the function
  if(DEBUG){Serial.println("Function ActuateComponents called");}
      if(Parameters[0] == 0 && Parameters[1] == 0){
      mos1.off();        // switch mosfet off
      relay1.off();      // switch the relay off
       if(DEBUG){Serial.println("MOSFET OFF, RELAY OFF");}
      }else if(Parameters[0] == 1 && Parameters[1] == 0){
      mos1.on();         // switch mosfet on
      relay1.off();      // switch the relay off
       if(DEBUG){Serial.println("MOSFET ON, RELAY OFF");}
      }else if(Parameters[0] == 0 && Parameters[1] == 1){
      mos1.off();        // switch mosfet off
      relay1.on();       // switch the relay on
       if(DEBUG){Serial.println("MOSFET OFF, RELAY ON");}
      }else if(Parameters[0] == 1 && Parameters[1] == 1){
      mos1.on();         // switch mosfet on
      relay1.on();       // switch the relay on
       if(DEBUG){Serial.println("MOSFET ON, RELAY ON");}
      }
}// end of the function ActuateComponents

void Reset(){
     // empties char buffer
       for(i=0; i<=indexChar;i++){
       inData[i] = '\0'; // sets all the cells to NULL character
       }
     indexChar = 0; // reset char buffer index for next parameters
      for(i=0; i<=indexNumber; i++){
       inParams[i] = 0; // sets al the cells to zero
       } // end of for(i=0; i<=indexNumber; i++)
     indexNumber = 0;
     inData[indexChar] = '\0';
}// end of the function Reset
```

## 7. *Interfacing Arduino with LabVIEW*

In this part, we are going to create an interface with the Arduino Uno thanks to LabVIEW.
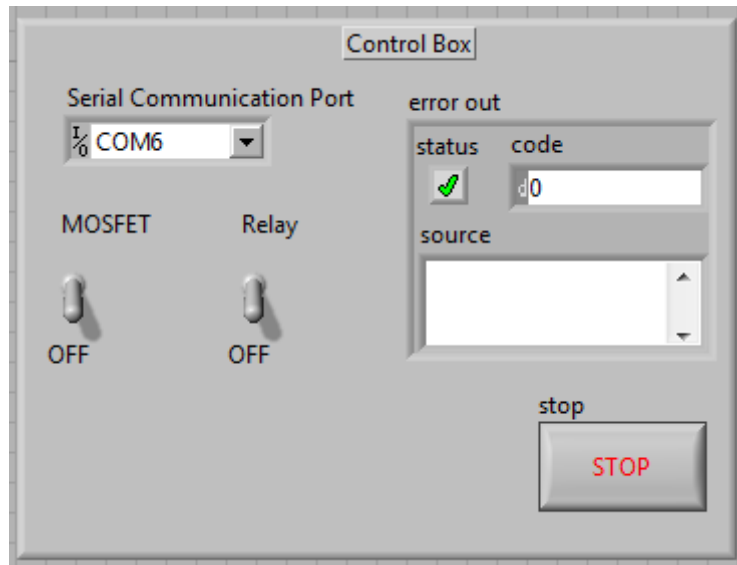
The purpose of this interface is to replace the Serial Communication Terminal, to allow a user who is not familiar with the Arduino command syntax, to send command messages to the controller for the purpose of actuating the devices. A situation like this one is often met when a customer needs to be able to operate a machine without resorting to coding in Arduino. Moreover, creating a user interface will prevent the user from entering wrong data, which could lead to dangerous conditions in real life.

We will therefore automate the writing of the commands necessary for the proper operation of our programme.

We will first start our programme by creating a VI which opens a serial communication to our controller. We will use the VI "VISA Configure Serial Port" for this.
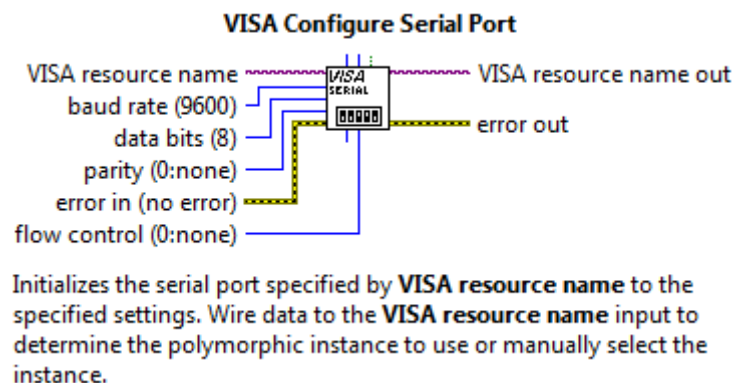
This VI needs to be given a VISA resource name, which will be a COM port (e.g. COM6) which is located in the Data Communication>Protocols>Serial Palette. So we

Right click on the resource name node and go to Create>Control. If you click on Control, a control block should appear like so:
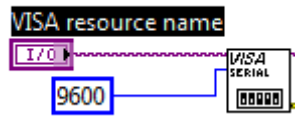

Rename this block to "Serial Communication Port". You can also set the baud rate to 9600 by right-clicking on the input "baud rate", and then Create>Constant and setting it to 9600.

Now we must open the communication that we have configured, which is the role of the VI "VISA Open":
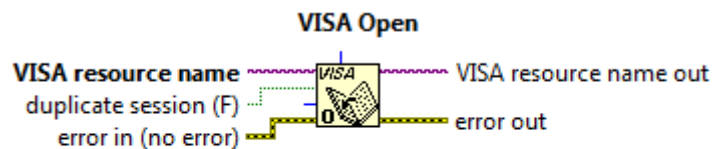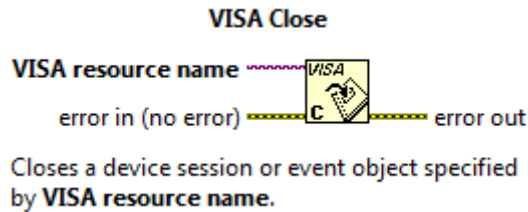

This VI can be found by pressing Ctrl + Space and typing "VISA Open", or by looking for the Instrument I/O>VISA> VISA Advanced palette as shown below:
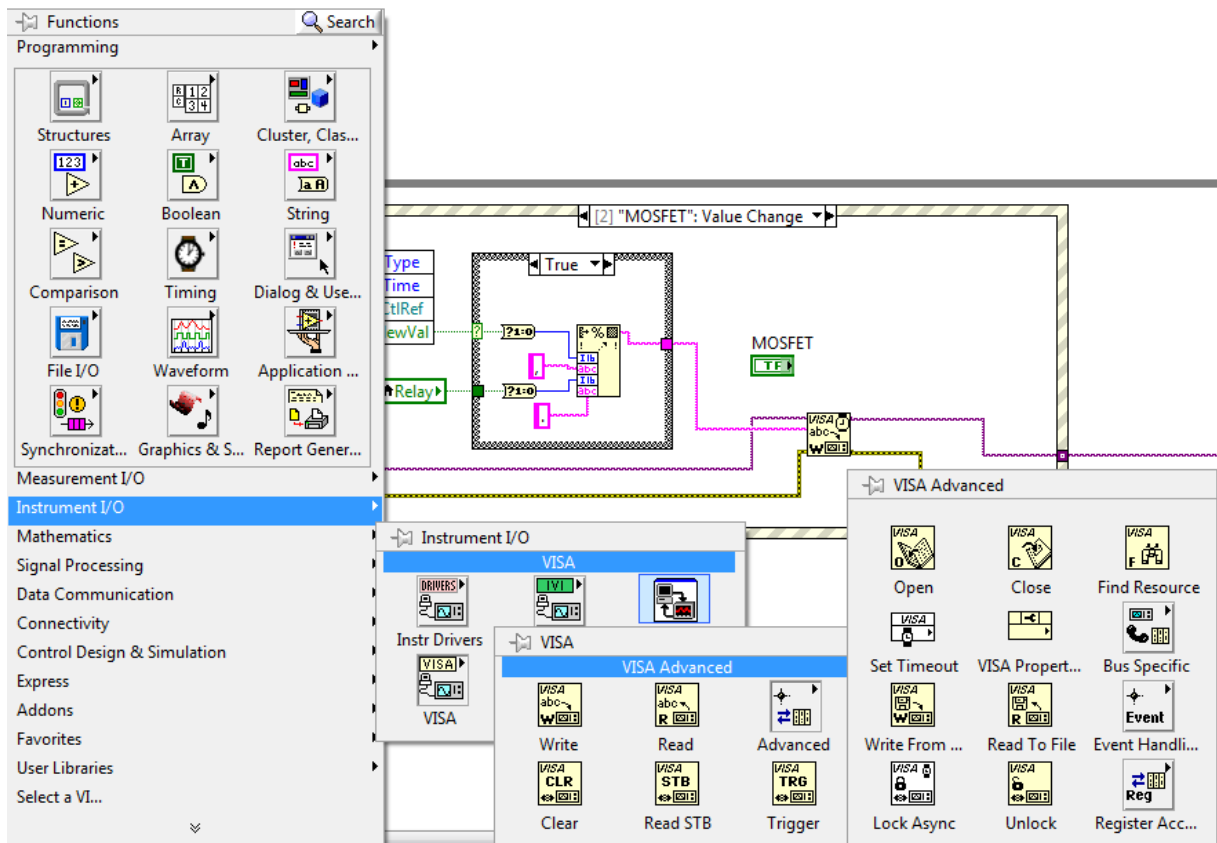
We will then create an Event Structure inside a while loop and add a "VISA close" VI which will close the serial communication port when we will press the stop button.


On the front panel, two toggle switches will be added, one of them will be named "MOSFET" and the other one "Relay", the toggle switches can be found in the Buttons palette.

We will then Add 3 Event cases to the Event structure:

1. Add one event case for the toggle switch MOSFET: Value change
2. Add one event case for the toggle switch Relay: Value change
3. Add one event case for the button STOP: Value change

The code inside the event case will be executed every time its respective toggle switch or button is actuated.

- Event case "MOSFET: Value Change"
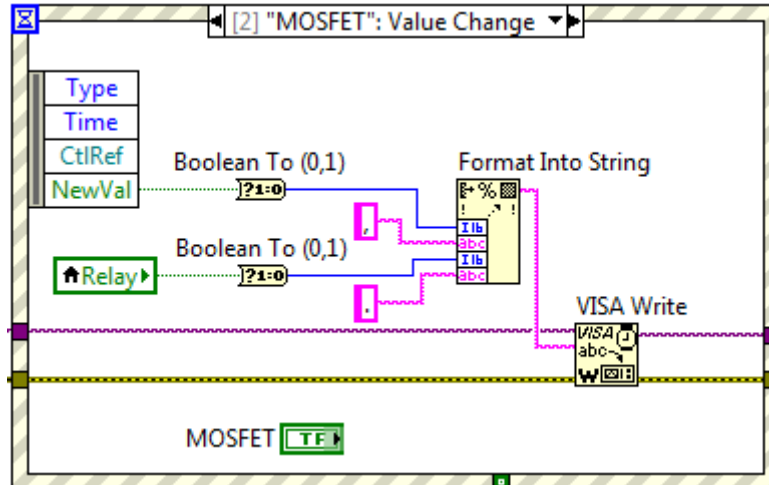
In this event case, we want to send a message to the Arduino to switch on or off the MOSFET module.

The value of the MOSFET toggle switch is accessed by the "*NewVal*" terminal of the event structure. On the other hand, the value of the toggle switch Relay is accessed using a Local Variable.

Local Variable

▸Local

Use local variables to read or
write to one of the controls or
indicators on the front panel of a
VI.

To create a Local Variable, hit Ctrl+Space and search for Local Variable…
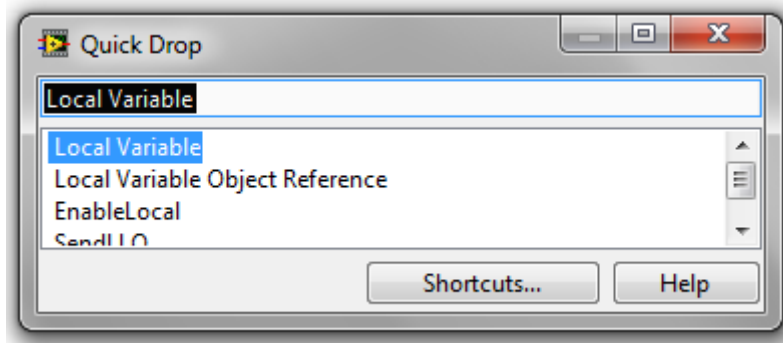
A block like this ▸⌂? should appear. Right click on it and select "Change to read" (1). You will see the arrow moving to the right in the clock label (2). Then left click on the amended block and select Relay (3). The local variable will be set to the value of the toggle switch Relay.
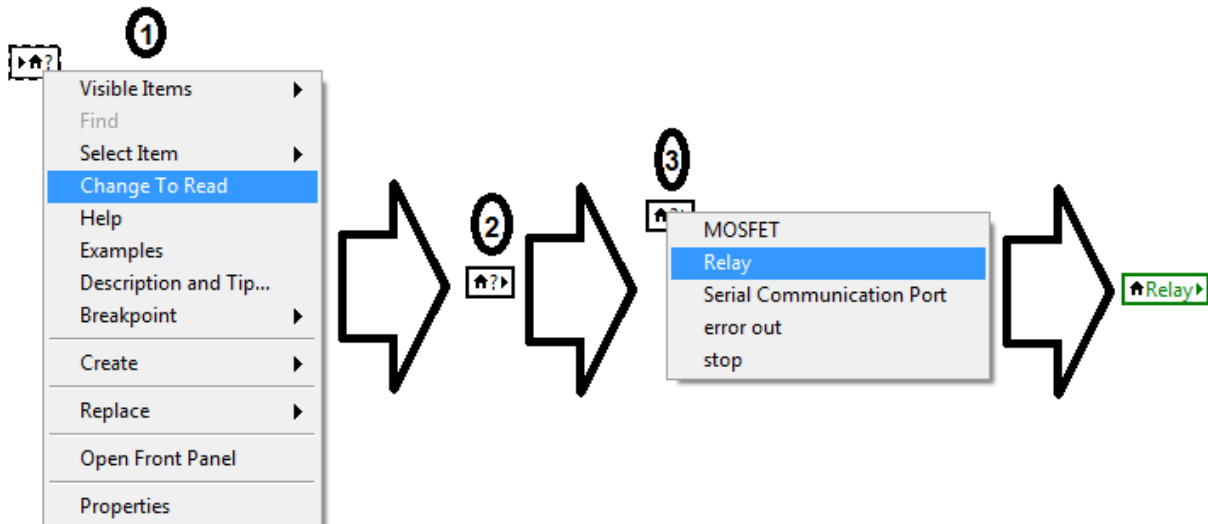
Since the buttons and switches have a Boolean data type (eg TRUE/FALSE) in LabVIEW, we use a "Boolean To (0,1)" VI to convert the Boolean values into integer values.

Note: The color of the wires change according to the type of data they contain, wire wired to a Boolean input/output will always be green.
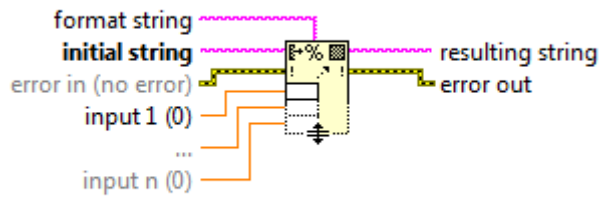
Boolean To (0,1)

Boolean ·········· |?1:0| ———— 0, 1

Converts a Boolean FALSE or
TRUE value to a 16-bit integer
with a value of 0 or 1,
respectively.

These values are then formatted into a string which will be sent to the microcontroller via the "Format Into String" and "VISA Write" VIs.

Format Into String

format string ~~~~~~~~~~~~~~~~~~~~
initial string ~~~~~~~~~~~~~~~~~~~ |⊦%▦| ~~~~~~~~~~ resulting string
error in (no error) ⊸⌐⌐⌐⌐⌐⌐⌐⌐⌐⌐ error out
input 1 (0) ⊸⌐
...
input n (0) ⊸

Formats string path, enumerated type, time stamp,
Boolean, or numeric data as text.

VISA Write

VISA resource name ~~~~~~~~~~ |VISA| ~~~~~~~~ VISA resource name out
write buffer ⊸⌐ |abc~| ⌐ return count
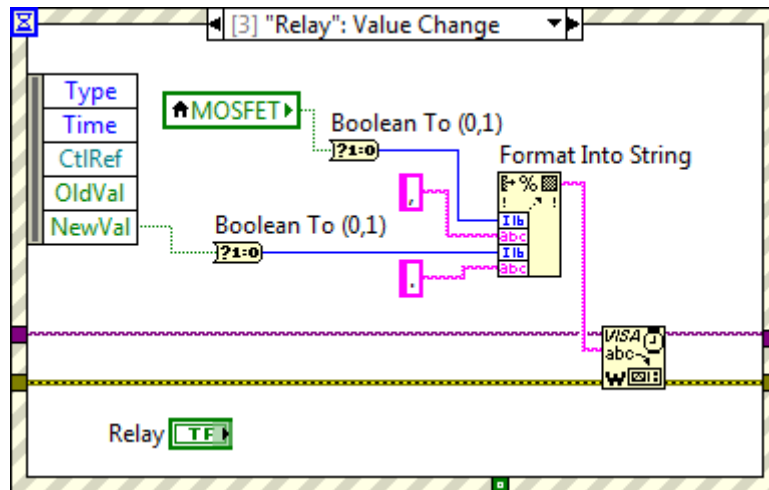error in (no error) ⊸⌐ |W▣▤| ⌐⊸ error out

Writes the data from **write buffer** to the device or interface
specified by **VISA resource name**.

The code in the MOSFET: Value Change event case should look like this:
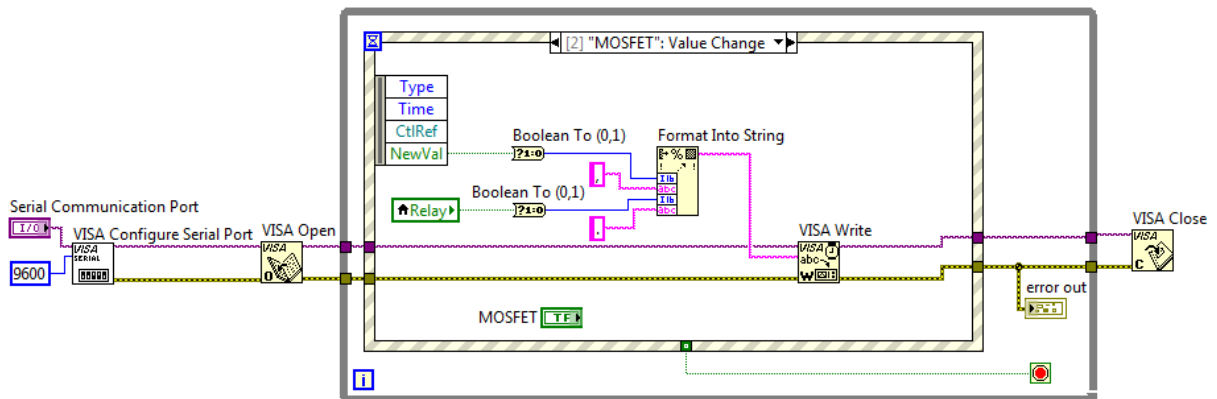
- Event case: "Relay: Value change"

In this event case, we want to send a message to the Arduino to switch on or off the Relay module. Since the buttons and switches have a Boolean data type (e.g. TRUE/FALSE) in LabVIEW, we use a "Boolean To (0,1)" VI to convert the Boolean values into integer values.

As you can see, the structure of the code is very similar to what we have previously seen for the event case "MOSFET: Value change", and should look like this:
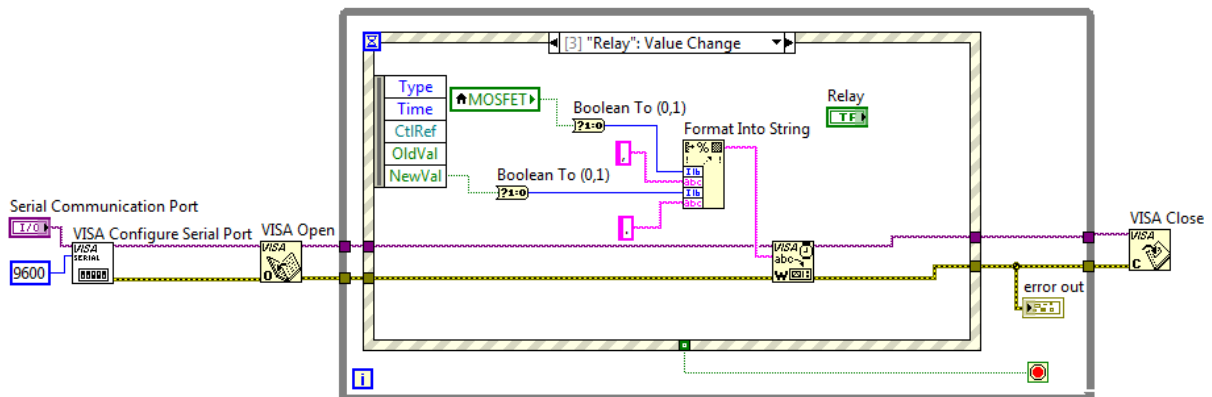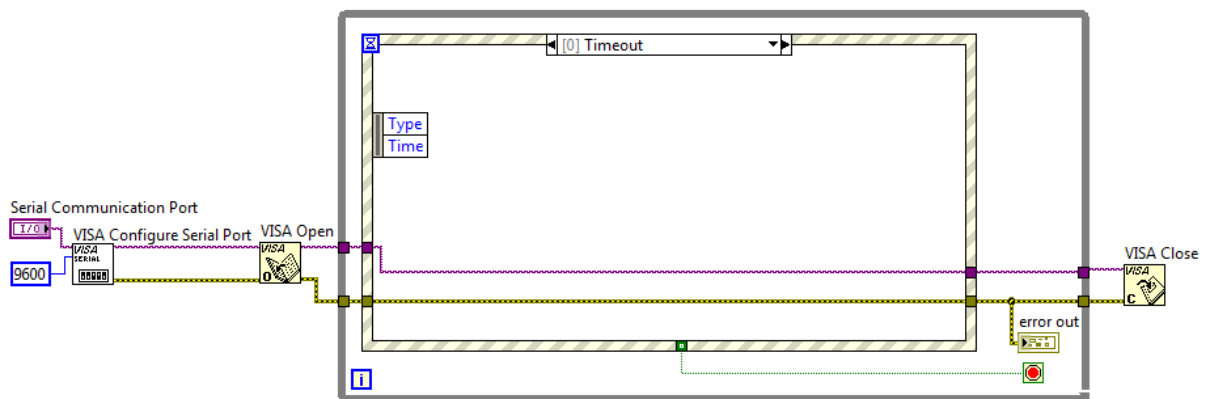

- Event case: "STOP: Value change"

This event case is very simple: simply move the STOP button inside the event case structure and wire its output to the conditional terminal of the While loop. This will terminate the while loop and close our programme.
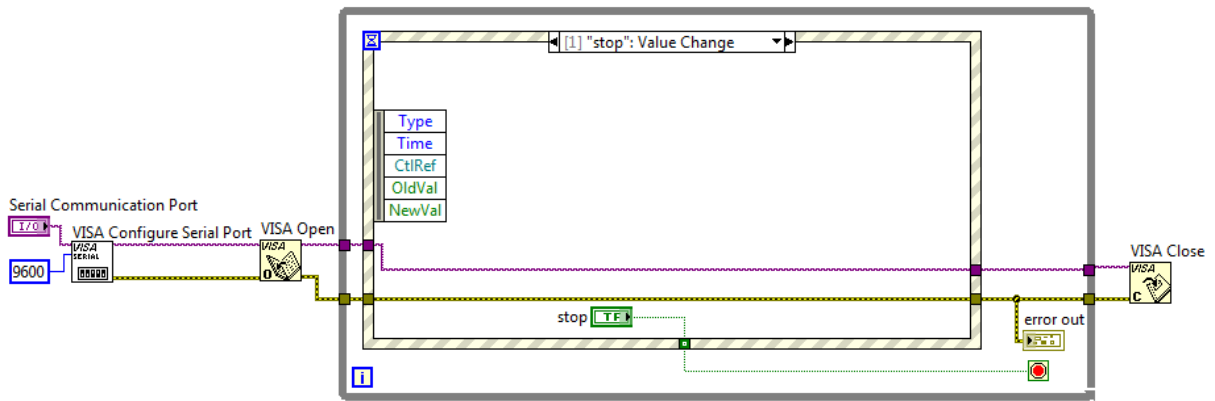
Do not write anything in the Timeout Event case:

- Testing the programme

If you have wired everything properly, you should be able to start the programme, select the Arduino COM port and turn on/off the modules by clicking on the "MOSFET" and "Relay" toggle switches. You should also be able to stop the programme by clicking the STOP Button.

8. *Examinations*

8.1 *Arduino programme*

The aim of this examination is to develop a programme that sets the speed of the DC motor of the conveyor belt using the MOSFET module and the Arduino PWM function. You can use and modify the functions presented in the previous examples to do so. Your Arduino programme shall:

- Receive a message sent via a serial terminal. This message shall contain the desired speed in meter per second.
- Print out the value that has been received on the serial terminal.
- Calculate the required duty cycle to set the right speed.
- Turn the red LED ON when the motors are running and OFF when the motor is not
- Make sure that your programme is constantly ready to receive new messages.
- Bonus: make sure the microcontroller does not execute when wrong data has been sent.
- You will also be evaluated on the clarity of your code and how well it is documented (commented).

Functions you will need:

```
atoi(char* string)
Serial.begin()
Serial.println()
Serial.available()
Serial.read()
XXX.on()// where XXX is the object you want to switch ON
XXX.off()// where XXX is the object you want to switch OFF
XXX.write(int duty)// with 0 <= duty <= 1023
or analogWrite(int pin_nr, int duty))// with 0 <= duty <= 255
```

Remember that your programme must follow the standard Arduino structure:

```
//make your variable declarations here

void setup(){

//write your setup instructions here

}

void loop(){

//write your loop instructions here
```

```
}
```

More details on the different functions are available at:

http://arduino.cc/en/Reference/HomePage

Good luck!

References

Arduino (2013), "Function declaration", Available online at:
http://arduino.cc/en/Reference/FunctionDeclaration [Accessed 06/01/2014]

Arduino (2013), "Array", Available online at: http://arduino.cc/en/Reference/array [Accessed 06/01/2014]

Arduino (2013), "For", Available online at: http://arduino.cc/en/Reference/for [Accessed 06/01/2014]

Arduino (2013), "Variables", Available online at:
http://arduino.cc/en/Reference/VariableDeclaration [Accessed 06/01/2014]

Arduino (2013), "if (conditional) and ==, !=, <, > (comparison operators)", Available online at:
http://arduino.cc/en/Reference/if [Accessed 06/01/2014]

Cplusplus.com (2013), "atoi", Available online at:
http://www.cplusplus.com/reference/cstdlib/atoi/ [Accessed 06/01/2014]

Durfee W. (2011) "Arduino Microcontroller Guide", University of Minnesota, Available online at: http://www.me.umn.edu/courses/me2011/arduino/arduinoGuide.pdf [Accessed 06/01/2014]

International Rectifier (2008), "IRFS3806 Power Mosfet specifications", Available online at:
http://ec.irf.com/v6/en/US/adirect/ir;jsessionid=7D19419E81140C82E8A30AE40BF0673A?cmd=catProductDetailFrame&productID=IRFS3806: [Accessed 06/01/2014]

Tinkerkit (2013), "TinkerKit Arduino Shield", Available online at::
http://www.tinkerkit.com/shield/ [Accessed 06/01/2014]

Tinkerkit (2013), "TinkerKit Mosfet module", Available online at:
http://www.tinkerkit.com/mosfet/ [Accessed 06/01/2014]

Tinkerkit (2013), "TinkerKit Relay module", Available online at: http://www.tinkerkit.com/relay/ [Accessed 06/01/2014]

Tutorialspoint.com (2013), "Data types in C", Available online at:
http://www.tutorialspoint.com/cprogramming/c_data_types.htm [Accessed 06/01/2014]