

---

## **Once again – controlling servos: The servo expansion modules from Wilke Technology**

Gunther Zielosko

### **1. Basics**

The topic of servos has already been dealt with once before in application note 3 and is, when considered from today's perspective, a rather singular and complicated solution. A much better and simpler way is to use the special modules from Wilke Technology. The expansion modules EP16 and EP17, which carry 8 or 16 servo outputs as well as 24 digi-scan inputs, are just made for all sorts of motorized drives in industry for the moment, but also for experiments with robots, model train systems, camera controls, in car models of all types etc. Whilst servos in model constructions are normally radio controlled "analog" via some sort of pulse-width modulation, the EP16- and EP17-modules do this purely digitally via a serial V24-interface. Taking this into consideration nothing now stands in the way of using the modules within the BASIC-Tiger® system or in other systems with RS232-interface. Every PC can then immediately control up to 16 servos directly. Interested? Then proceed reading.

### **2. The modules EP16-Servo8 and EP17-Servo16**

As usual, the expansion modules come in a sealed BASIC-Tiger® casing and have 92 pins. Next to the servo controls there is also an almost independent digi-scan unit with 24 inputs, whose logical condition can be read via the bus system. A welcome addition for those who wish to read out switches, sensors or other contacts in a model. This section is only of marginal interest to us and will therefore not be dealt with further here.

Taking a look at the servo part, there are up to 16 servo outputs Servo-0 to Servo-15 and altogether four control pins, which are named `RxD_TTL`, `CTS_TTL`, `RxD_V24` and `CTS_V24`. For our experiments we would actually only need both V24 control pins, as these should be, according to their description, compatible to the PC's COM interfaces and the Plug-and-Play-Lab (Level -3...-15V and +3...+15V). Unfortunately this statement in the Tiger-Pocket-Guide is incorrect. Although the data input `RxD_V24` tolerates these levels and evaluates them correctly, the sending output `CTS_V24` in comparison only delivers TTL levels (although correctly negated to `CTS_TTL`). When working with the BASIC-Tiger® this seems to be working nevertheless, but with a PC program it does not. The CTS output of the servo part only allows a data transfer at specific times within a cycle, at the wrong level the PC's COM-interface refuses to send data at all. It is therefore more secure to use the TTL pins of the EP servo module and to ensure clear levels with an external MAX232.

Finally, it is of course important to connect mass (GND) and operation voltage ( $V_{cc} = +5V$ ) to the module. Do not forget to connect module GND with serial interface GND (Pin 5 at SUB-D plug) and servo GND. To connect the servo you need 3 lines, namely GND, +5V (if

possible from an independent source) and the servo control signal. The exact circuitry of all components will be presented later on.

That is the hardware – it could not be any simpler. Is it just as easy with the software? Unfortunately not – the processing of a command for the servo module is rather complicated and demands explanation. If you wish to use the BASIC-Tiger® as control computer, kindly read the explanations in section 6, where you will find program examples for different applications. In case that the PC should control servo motors there is a comfortable windows program, with which you can have access to at least 8 servos.

### 3. Processing the control commands

We know from the past that servos for model construction adjust their position via a pulse length control (0.552 ms to 2.596 ms). In common remote control systems this is done analog via monoflops. With the EP servo module the impulse length for each servo is set by a built-in processor and is updated every 20 ms. Data for the impulse lengths is submitted via a serial interface. A single RS232 interface controls up to 16 servos. This is possible by giving each servo an address, e.g. Servo 0 = 00h, Servo 1 = 01h...and Servo 15 = 0Fh. Aside from the actual positioning data, each control command contains the servo address of the servo that is to be adjusted. The positioning data itself are 10 bit values; so the servo can (in theory) be driven from position 0 (e.g. left stop) up to position 1023 (e.g. right stop), a very sensitive control. There is a peculiarity here as opposed to the analog servo control – position 1023 is not attainable, here there is a possibility to switch the servo power off, i.e. to save energy or to allow the servo to be moved manually. In practice the positions run from 0 to 1022, which analog impulse lengths for the servo from 0.552 ms to 2.596 correspond with (figure 1).

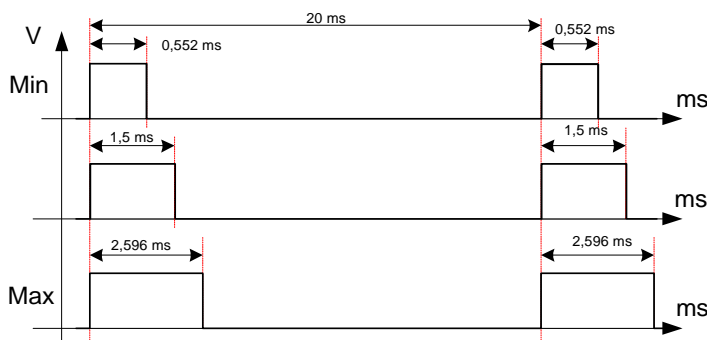


Fig. 1 Impulse concept of a servo output (Positioning values Min = 0, Max = 1022)

A command sequence consists of 2 bytes and contains, as explained already, the address of the servo and its new position. The low-byte is always transmitted first. Table 1 shows how the data is arranged. The servo addresses are shown in blue and the positioning data in violet. The lower rows contain examples of values, which will be explained in the following.

Byte	High-Byte							Low-Byte								
Data Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Position Bit						9	8	7		6	5	4	3	2	1	0
Servo-Adr. Bit		3	2	1	0											
Example FF 07	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
Example D2 0E	0	0	0	0	1	1	1	0	1	1	0	1	0	0	1	0
Example 96 11	0	0	0	1	0	0	0	1	1	0	0	1	0	1	1	0

Table 1 Distribution of data and address bits

Bit 7 of the Low-Byte is always set to “1”, bit 7 of the High-Byte always to “0”. This reduces the possibility of wrong settings in case of “mutilated” data or other transmission problems. The 10-bit positioning value of the servo is distributed amongst both bytes. Bits 3 to 6 of the High-Byte contain the servo address. Our 3 examples can be explained as follows:

```
FF 07          Servo 0 to position 1111111111 (dec. 1023 = switched off)
D2 0E          Servo 1 to position 1101010010 (dec. 850)
96 11          Servo 2 to position 0010010110 (dec. 150)
```

Is everything clear?

It is not necessary, by the way, to constantly reprocess all data and transmit it. It is sufficient to only supply the servo which is to be adjusted with new data; the other servos remain stable in their position in the meantime.

The following parameters must be set for the serial data transmission:

```
Baud rate:    38400 Bd
Data bits:    8
Parity:       No
```

It should be mentioned once again here that the module’s serial interface does not receive data at all times – this is only possible during 14.2 ms in every 20 ms cycle. This is controlled via the CTS signal, the BASIC-Tiger or the PC should always be connected to this pin, too. The intelligent time management of the BASIC-Tiger as well as respective PC programs have data buffers and mostly handle this reasonably well.

#### 4. The problem of mechanical servo limitation

Normal model construction servos are driven by the sender with monoflops, which in turn are adjusted with a resistor (joysticks, slider controls or control knobs) in their initial impulse length. In common remote control systems with sender, receiver and servo electronics the, according to definition, maximal possible stop positions are usually not realized. With the purely digital impulse length adjustment of the EP servo module a minimal or maximal impulse length is very possible. If these extreme values are set, it can happen that the servo tries to move beyond the mechanical stops. Although the motor is unable to do this, there is a considerable current flowing and the servo could be damaged. This means that the user must

somehow make sure that the positioning data is limited to values that don't let the servo reach the mechanical stops. This applies to the BASIC-Tiger as well as PC-based programs. Either the maximum and minimum impulse lengths are adjusted in such a way to guarantee that none of the servos gets into this awkward position, or an automatic calibration must be ensured. Ways to achieve this are shown in the following chapters.

## 5. Servo control with the PC and the program „SERVO01.TST“

In order to operate the servo module on the PC, a circuit according to figure 2 is needed. It goes from the COM interface at the PC (usually a 9-pole SUB-D male connector) through a cable with 9-pole SUB-D female connector to the level converter MAX232, which creates TTL signals from the RS232 signals. Both these signals and GND take over the control of the servo section of the extension module. From there only the individual servo control lines lead to the individual servos. If you are using servos from Graupner, the color codes of the circuit diagram are valid. Otherwise you have to evaluate how to connect your servo.

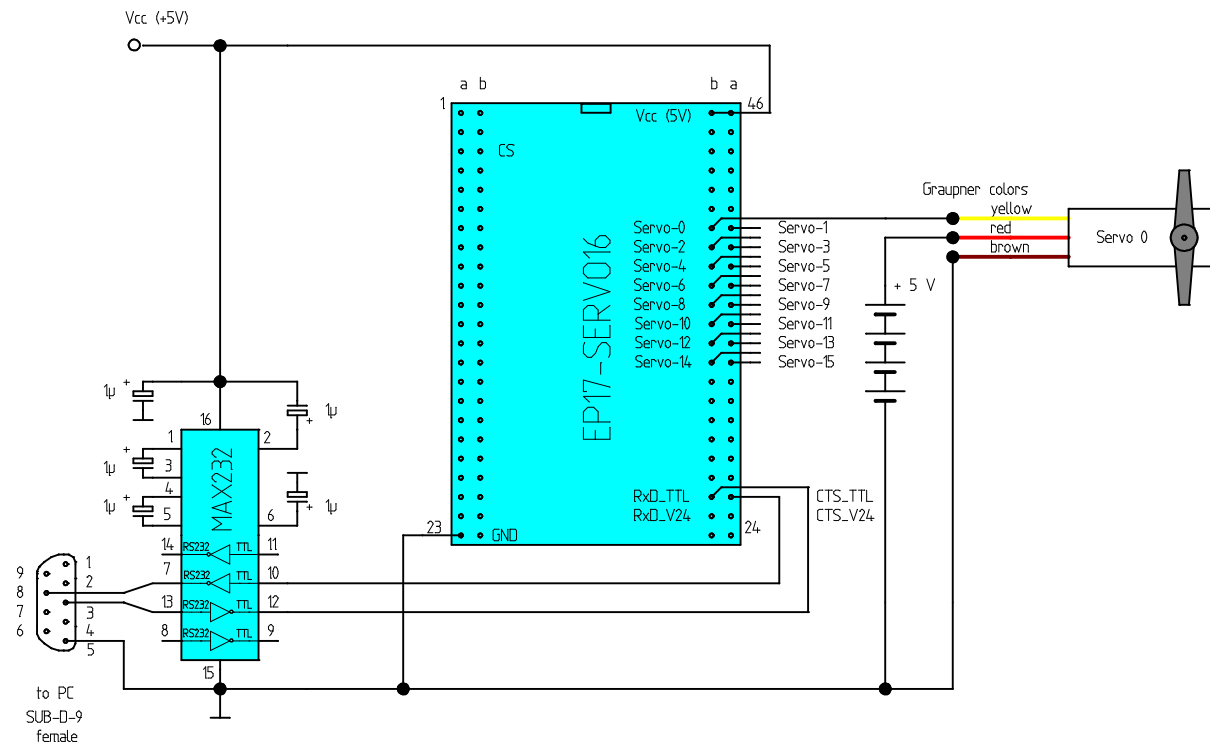


Fig. 2 Switch EP17-Servo16 for the operation with a COM-interface of the PC

It is now possible for a simple PC program to adjust several servos in their positions via the COM interface. A simple PC program is easier said than done. Such a program, of course under Windows, must initially be able to somehow comfortably set the adjustable values for the servos. These must then be brought into the form which we described in chapter 3. Thereafter they must be sent directly to an appropriately preset COM interface in the form of a 2 Byte-Word (so not as ASCII characters!). For non-Windows programmers this is rather

complicated. To make the use of the EP servo modules a little easier for these contemporaries, a comfortable windows program, based on TestPoint from Keithley, is provided with this application note, which you just have to install. Once installed, the program **SERVO01.TST** shows itself as Runtime-Module like this:

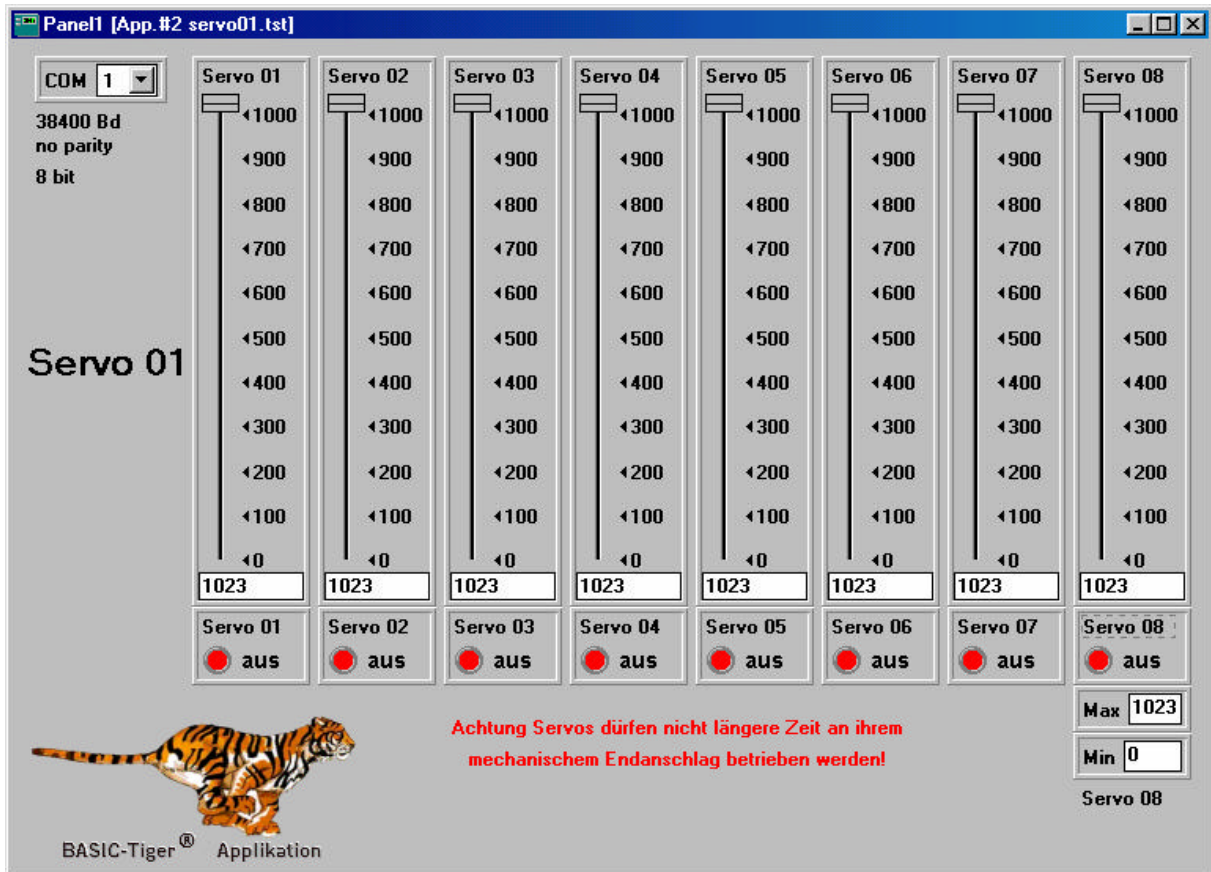


Fig. 3 The program **SERVO01.TST** for the positioning of up to 8 servos with the PC

A program such as **SERVO01.TST** is naturally not able to cover all conceivable applications for servos. Nevertheless it offers some possibilities for experimentation.

You will find 8 sliding controls with which you can comfortably set each servo individually. You can adjust the position in three different ways:

- Simply with the mouse (Cursor on the slider, left click and drag)
- Left click the requested control with the mouse and fine tune with up and down arrow keys (single steps are possible)
- Left click the requested control with the mouse and enter value directly into the text box.

Please ensure with all experiments that you do not drive any servo permanently beyond its mechanical end position, in which it draws large currents and might actually be destroyed (Chapter 4)!

The adjustment value 1023 is, as has already been mentioned, no real positioning value, but rather a special function for completely turning off the servo. You can achieve a real turn off

by driving it into the upper setting (1023). An active servo is symbolized in the program window by a green “LED”, a turned off servo by a red one. Below the slide control for servo 08 you will find 2 input boxes where you can experiment with the maximum and minimum levels (only for this servo!). Servo 08 has the same extreme values in the beginning, i.e. 0 and 1023, as the others. If you drive the servo close to the extreme values, you will notice that at one point it doesn't proceed anymore. Alternatively, you can switch an ammeter into the power supply line for the servos in order to measure the current. Once the servo reaches a mechanical end position, there is a considerable constant current. Now enter maximum and minimum values, which prevent the servo from reaching these “dangerous” zones. As the impulse time of servos is set digitally and therefore absolutely exact, the determined range for the servo in question is automatically applicable to all adjustment controls.

## 6. Servo control with the BASIC-Tiger<sup>®</sup>

The extension modules EP16-Servo8 and EP17-Servo16 were naturally developed to run with the BASIC-Tiger<sup>®</sup>. First of all the circuit here:

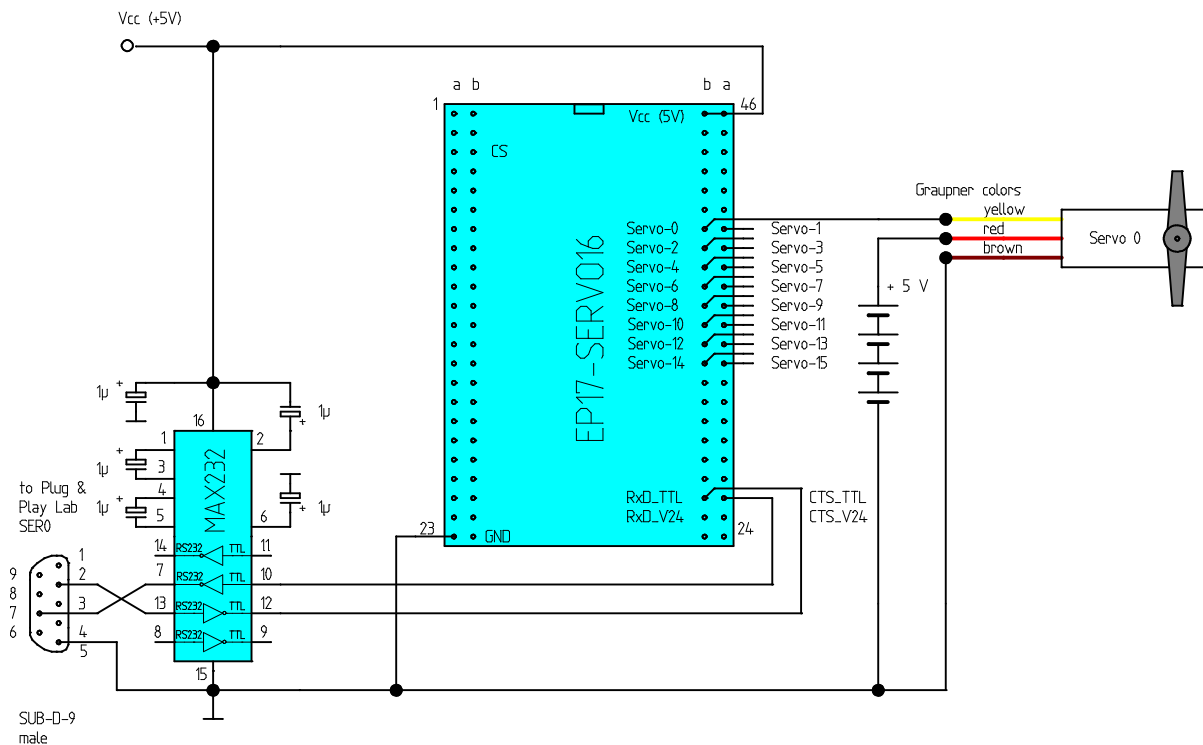


Fig. 4 Circuit EP17-Servo16 to run with the Plug and Play Lab (SER0)

You can see that operation with the Plug and Play Lab requires a different plug connector – here a 9-pole SUB-D-plug is being used. Furthermore the wiring differs in comparison with the version for the PC. Please note that due to the required CTS connection only the serial interface SER0 can function securely, as only this has a CTS (Clear to send) connection.

Other than that the Plug and Play Lab always operates with RS232 levels – no matter what type of BASIC-Tiger<sup>®</sup> you use. For stand-alone solutions with a BASIC-Tiger<sup>®</sup> you must think a bit more. For the (more expensive) Tigers with built-in RS232 interface you must sacrifice another MAX232. With the apart from that identical circuit as shown in figure 4, only the connections to the BASIC-Tiger<sup>®</sup> are displayed in figure 5.

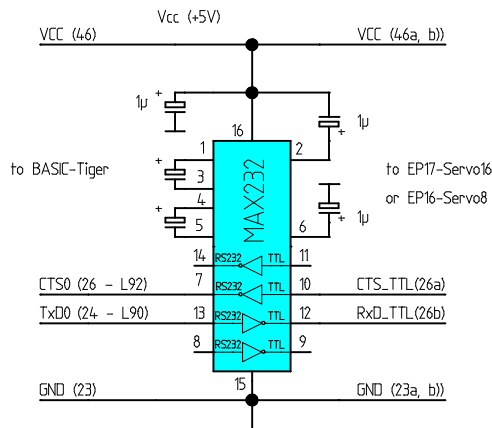


Fig. 5 Direct connection to BASIC-Tiger<sup>®</sup> with built-in RS232 interface

Tigers without RS232 interfaces are cheaper and you additionally save one MAX232. Figure 6 shows the differences, here too only the connection to the Tiger is shown.

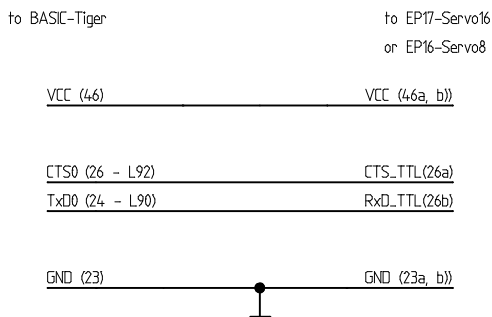


Fig. 6 Direct connection to BASIC-Tiger<sup>®</sup> without built-in RS232 interface

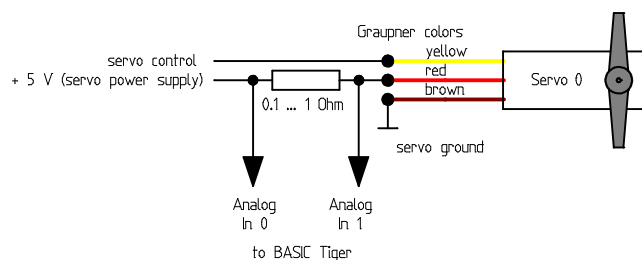
That much on the “hardware”, that is reduced to a few wires in the last case.

On BASIC-Tiger<sup>®</sup> software side, most of us have almost endless possibilities at our disposal, which makes a difference to the Windows programming at the PC. Let us begin with a simple test program.

The program **SERVO01.TIG** controls all theoretically possible 16 servos at the same time. This is done in a loop, meaning the servo address (Variable „devAdr“) is counted from 0 to the highest set address (Variable „LASTSERVO“). If you enter 0 at #DEFINE LASTSERVO, only one servo (Servo 0) is addressed – in the existing form of SERVO01.TIG it is set this way. Initially all servos are set to position 1023, which is the “off” position. No servo is in

function and no motor current is flowing. Thereafter all servos are driven to a very high position (MAX), here 700, in order to ensure that the dangerous operation at the mechanical stops is avoided. After a brief lingering a small position (MIN) is approached, that is set to 100 here. In the end all servos are switched off again with the value 1023. The whole process is repeated for all servos – a great duration test with the option to test, for example, the repetitive exactness of the adjustment values. Incidentally you can see the following on the LC display: the servo address (decimal), the current position (decimal) and the command word, which is derived from both values (hex., here High-Byte first!).

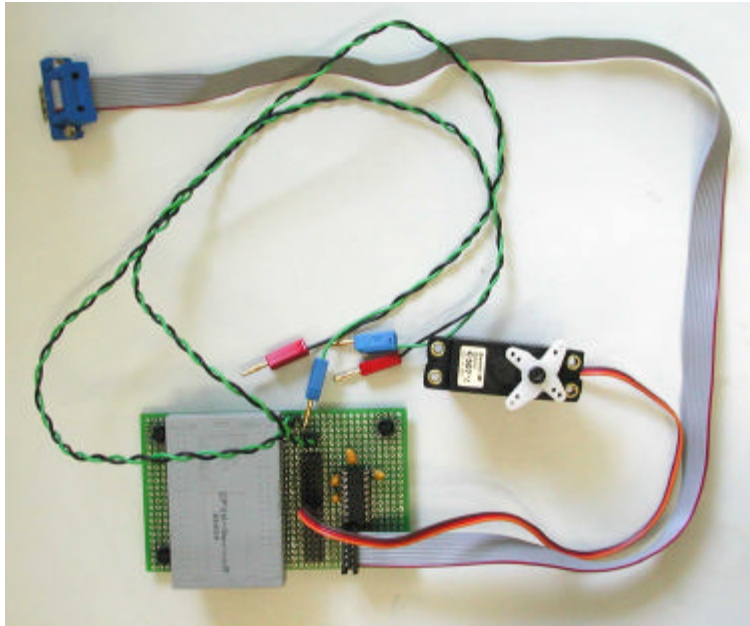
The program **SERVO02.TIG** serves for automatic detection of mechanical end positions of servos. We have learned that a servo tries with all its power to reach a position corresponding to the control impulse length. If it reaches the mechanical stop by doing so, the servo current is increased considerably. We are taking advantage of this effect. Initially **SERVO02.TIG** originates in the theoretically reachable positions 0 to 1022 (MIN, MAX). From the calculated middle position 512 onwards there is a slow increase of the positioning value. When reaching the mechanical stop the current in the servo increases. We measure that by simply fitting a resistor of 0.1 Ω to 1 Ω into the supply line of the separate +5 V source for the servo. With two wires we tap the voltage above the resistor and measure these with the two analog inputs AD0 and AD1 of our BASIC-Tiger<sup>®</sup> (Figure 7). To do so please connect at the BASIC-Tiger<sup>®</sup> Analog GND (Pin 44) to GND (Pin 23) and A/D Ref-volt-in (Pin 43) to Vcc (Pin 46). A measure for the current is the voltage difference between both ends of the resistor according to  $I=(U_1-U_2)/R$ . The BASIC-Tiger<sup>®</sup> calculates this and ceases the movement once a preset current limiting value has been reached. It stores the determined maximum value (MAXn). Naturally the determined value for the current is no real current value, as the voltage difference is only interesting as a current jump (end position reached). The same game begins for determination of the minimum value. Once both extreme values have been reached the new “real” maximum and minimum for the individual servo is displayed. Finally the servo moves back and forth between these two end points.



*Fig. 7 Servo current measurement*

A fantastic, self-learning servo, right? If you want, you can extend the program, which has been limited to one servo (Servo 0) for clarity purposes, to all possible channels and thus achieve a calibration of all connected servos...

Finally, once again the author's whole lab construction (Figure 8).



*Fig. 8 Lab construction with EP16-Servo8, MAX232, Servo, serial cable and power supply cables.*

Have fun with your servos!