Awkward keyboard commands and uncomfortable mouse control can spoil even the most attractive PC game. The ideal solution would be simple operation, as offered by the Nintendo 64, combined with the processing power of a PC. This article describes how to connect a Nintendo 64 controller to the PC game port.

design by K. Schuster

PC interface for Nintendo joystick

using the Nintendo 64 to run PC games

This circuit allows a Nintendo 64 controller to be connected to the PC game port (or a sound card), without requiring any additional drivers to be installed. The Nintendo 64 controller is a widely-used unit that combines high quality with a low price. With this approach, you can run PC games with the comfortable Nintendo 64 controller instead of using the PC keyboard and mouse.

What the Nintendo 64 controller offers...

In addition to a few membrane switches, the controller contains a precise analogue electro-optical joystick module that works like a mouse. On demand, the controller unit reports the status of the switches and the position of the joystick. Bidirectional communication takes place over a single line that has a High level in the rest state. This line is used both to send commands to the controller and to receive the requested data from the controller. A command byte must be sent before data can be received from the controller. If the line is free, as indicated by a persistent High level, the command byte can be transferred. The controller responds to the command $01 with the status information for all pushbuttons and the position of the analogue joystick. The transmission time for each bit is 4µs in both send and receive modes. A Low bit is indicated by a 3-µs Low phase followed by a 1-µs High phase, while a High bit is indicated by a 1-µs Low phase followed by a 3-µs High phase. In order to delay the response to a command, the last transferred bit of the command can be held Low. If the line is returned High at the end of the command transmission, the response should occur within 2 to 3 microseconds. The response time is not fixed, since the controller and the N64C2PC IC operate asynchronously, each with its own clock.

The first experimental circuit, with a 8051 clocked at 12 MHz (corresponding to a 1 µs cycle time), was obviously too slow to meet the critical timing requirements of the Nintendo 64 controller. Reliable communication was only possible after the microprocessor was replaced by an AT89C2051-24PC with a 24-MHz clock. Regarding the hardware, you can see that two clock sources are shown in **Figure 1**, in addition to the microcontroller and a pair of current-limiting resistors. This is because 24-MHz crystals are normally only available for series-resonant operation. Such ‘overtone’ crystals are not suitable for this application! If you cannot obtain a fundamental-frequency crystal, you can use a self-contained 24-MHz oscillator (see the list of components).

Returning to the communications with the controller, the answer to the command $01 is four bytes of controller status information, transmitted MSB first, as shown in **Table 1**.

...is not what the game port expects

A simple PC game port does not need any active circuitry. The two pushbuttons simply make connections to earth. The PC game port, or a suitable sound card, simply polls the switch levels to see whether they are High or Low.
With the analogue joystick, the situation is a bit more complicated. The joystick contains two potentiometers (X and Y), whose resistances are around 100 kΩ, connected to the supply voltage. Capacitors located on the card are charged via these potentiometers. These capacitors determine the time constants of a pair of monostable multivibrators. The positions of the potentiometers can thus be derived from the lengths of the pulses produced by the monostables. All analogue elements are addressed or polled at the same time. A normal PC game port provides connections for two joysticks, which means that it has four 'digital' and four 'analogue' inputs. Sometimes only one joystick can be connected, but this is very rare.

Two worlds join together

It is not difficult to see that these two worlds do not really fit with each other. Requesting and interpreting the status data from the Nintendo 64 controller should not be difficult, but how can the expectations of the PC game port be satisfied without a lot of complicated circuitry? Handling the pushbuttons is relatively easy; the relevant bits from the Nintendo 64 controller can simply be periodically output on the microcontroller leads. However, what should be done with the digital values for the analogue joystick? Here we can use a trick: the interface microcontroller waits for a short Low level on one of the potentiometer lines, which goes along with the cyclic charging of the capacitors of the PC game port card. Following this, the microcontroller holds all of the potentiometer lines Low, to prevent any further charging of the capacitors, and starts its timer. Each of the potentiometer lines is subsequently allowed to go High at a time that depends on the data received from the Nintendo 64 controller. The corresponding capacitors are charged briefly via the microcontroller outputs, and the associated monostables report what they assume to be the potentiometer positions. If you observe the relevant outputs of the AT89C2051 with an oscilloscope, you will see pulse-width modulated signals with a period of around 840µs and a duty cycle of 50% to 90%, depending on the potentiometer position. When the potentiometer is at the midrange position, the duty cycle is 70%.

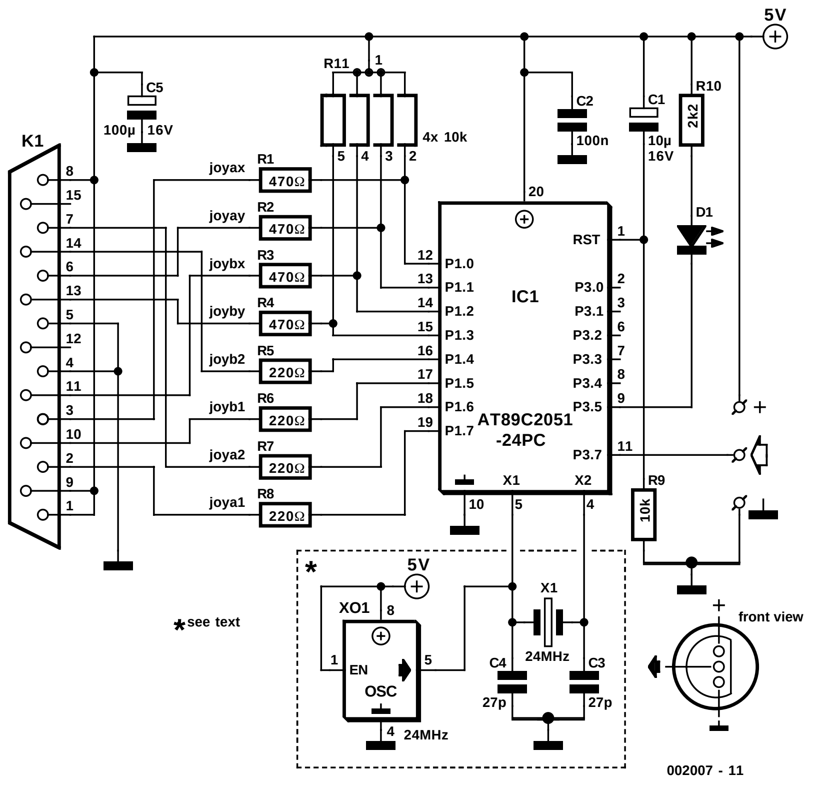


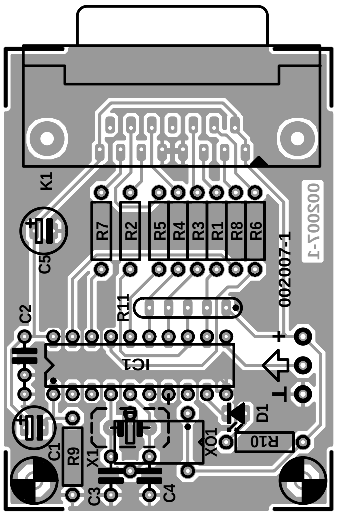
Figure 1. A microcontroller, a pair of resistors and an oscillator are all you need for the adapter circuit.

Details — the program

The software, including the source code, is available from *the Elektor Electronics* web site (www.elektor-electronics.co.uk). If you cannot program the microcontroller yourself, you can obtain a ready-programmed device from our Readers Services under order code **006504-1**.
The main loop of the program starts after the stack and the two timers have been initialized, the timers have been started and their interrupts have been enabled. First, the timing for the analogue joystick modules A and B (B is the control cross or C button) are established by the routines prepajoyt and prepbjoyt, respectively. Timers T0 and T1 are responsible for the timing of joystick A, with T0 used for the X axis and T1 for the Y axis. Timer T0 also manages the Timeout Mode, which prevents the program from getting stuck in a polling loop if the Nintendo 64 controller is unexpectedly disconnected or there is an intermittent contact. In such a situation, it would otherwise not be possible to initialize the Nintendo 64 controller once it was reconnected without first manually resetting the microcontroller.
The entire program is synchronized with the slowest and least-flexible element, the PC game port. The instruction jnb JPYAX,* waits for the capacitors to be discharged. Once the game port has

Table 1. Nintendo 64 serial status information

Byte 1	
Bit 7	button A
Bit 6	button B
Bit 5	button Z
Bit 4	start button
Bit 3	control cross up
Bit 2	control cross down
Bit 1	control cross left
Bit 0	control cross right
Byte 2	
Bit 7	unknown, always 0
Bit 6	unknown, always 0
Bit 5	button L
Bit 4	button R
Bit 3	button C up
Bit 2	button C down
Bit 1	button C left
Bit 0	button C right
Byte 3	
analogue stick x	
Byte 4	
analogue stick y	



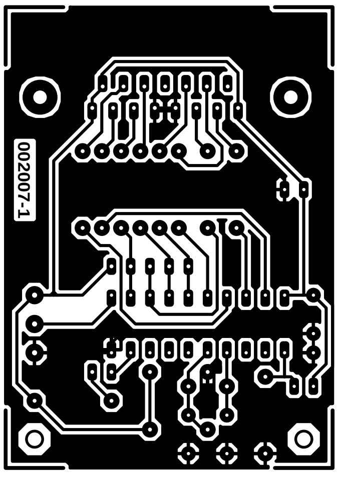


Figure 2. The printed circuit board for the Nintendo-64/PC adapter.

COMPONENTS LIST

Resistors:
R1-R4 = 470Ω
R5-R8 = 220Ω
R9 = 10kΩ
R10 = 2kΩ2
R11 = SIL-resistor array 4x10kΩ

Capacitors:
C1 = 10µF 16V radial
C2 = 100nF
C3,C4 = 27pF (*)
C5 = 100µF 16V radial

Semiconductors:
D1 = LED, low current
IC1 = AT89C2051-24PC (order code **006504-1**)

Miscellaneous:
K1 = 15-way sub-D plug fr board edge mounting
X1 or XO1 = quartz crystal, 24MHz, fundametal resonance or 24MHz-oscillator module (Seiko-Epson SG531P-24MHz) (*)

(*) = see text

done this, the microcontroller sets the four potentiometer lines JOYAX/Y and JOYBX/Y Low and starts timers T0 and T1 for JOYAX/Y, since these are assigned to the analogue joystick. The control cross or C button is assigned to JOYBX/Y. Analogue values are also expected here, so the timing is handled by the routine joybtiming, due to the lack of additional timers in the microcontroller. With the help of a few NOPs and nested loops, the game port receives what it expects here as well, and the JOYBX/Y lines are set high again after appropriate delays. The rate of advance in the Y direction can be set to one of three different levels by simple ‘switch-on, switch-off’ logic. If the control cross or the C button is used during a game for forward or reverse motion, the L button can be used to switch between ‘creeping’, ‘walking’ and ‘running’. The duty cycle range is thereby switched from its default range of 58%–78% to either 48%–88% or 40%–97%.
After both software timers have timed out, the program waits until the hardware timers T0 and T1 have completed their jobs and generated interrupts. Once they have timed out, the JOYAX/Y outputs are again set to High. Since the program can easily get hung in the subsequent time-critical portion, the timer T0 interrupt is used as an ‘emergency brake’ timeout in the routine Inittom. If the Nintendo 64 controller does not respond within a predefined interval, the program is restarted from the beginning. The routine sendbyteA sends the command $01 (Status Information), and the following routine getbytes reads the four status bytes from the Nintendo 64 controller. Bytes 1 through 4 land in registers R4 through R7 for further processing. Before each byte is read, precise bit synchronization is established, following which the Timeout Mode of Timer 0 is again deactivated and the values that have just been read in are interpreted in the routine handlebuttons. This works according to the arrangement shown in **Table 2**.
Repeatedly pressing the L button changes the advance rate of the control cross up/down buttons or C button in three steps.
Once the switch states have been evaluated and their status has been passed on to the PC game port, the loop starts from the beginning with the evaluation of the analogue values that have been read in. The routine calctiming normalizes and scales these values in terms of processor cycles, and the resulting data form the inputs for the next round, which begins with the discharging of the capacitors.

Playing around

In order for the new joystick to be used with the PC under Windows 95/98, it must be made known to the operating system. You should find a joystick or game controller icon under

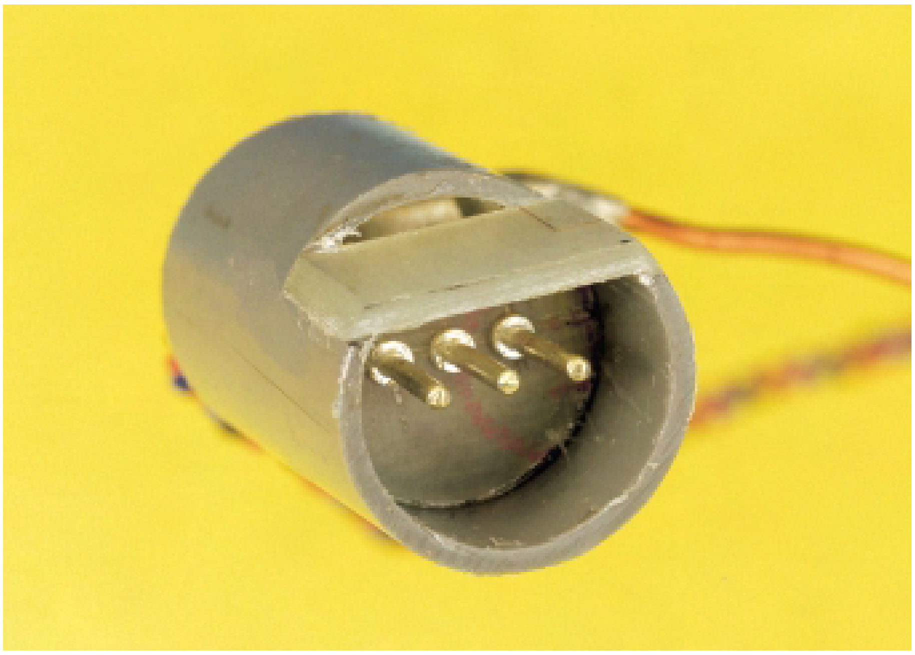


Figure 3. How to prevent an incorrect connection.

Settings/Control Panel. If you do not, the necessary software must first be installed. After this, the best approach is to configure a new joystick with four axes and four pushbuttons. During the subsequent calibration, make sure that the up/down buttons of the control cross, or the C button, have been set to the highest speed using the L button (recognizable by the largest displacement on the screen). The settings can be saved with the name ‘N64’, for example. This name may be needed later to configure certain games. Older (DOS) games only require calibration.

Some games (such as Unreal) offer an extensive range of joystick settings, which you will have to carefully study and try out. In some cases, such as with Half-Life, you will need a small joystick configuration file that contains the configuration data. The game looks for this file in a particular folder when it is started (for example, c:\Sierra\HalfLife\valve). You can usually find tips in the ReadMe files of the games as well. **Table 3** shows two typical configuration files.

Construction hints

Constructing the circuit, using the printed circuit board shown in **Figure 2**, should not present any difficulties. This PCB is unfortunately not available ready-made through our Readers Services. Mount the microcontroller in a good-quality socket. The choice between a quartz crystal and an oscillator module has already been discussed. If an oscillator module is used, omit capacitors C3 and C4 (and of course X1). Difficulties may arise with the (various) controller plugs, since matching sockets are hard to come by. There are three possible solutions: (a) cannibalize an old Nintendo 64 console, (b) cut off the plug and make up an adapter cable with a three-way DIN or Mini-XLR plug (with a mating connector on the end of the cable), or (c) improvise a solution using 1.3-mm diameter solder pins to which short lengths of wire are soldered, which in turn can be soldered to the inputs of the AT89C2051 (see **Figure 3**). To protect against a reverse-polarity connection, you should solder the pins to a piece of prototyping board with a hole spacing of 3.75 mm, and then use an additional part (for example, a piece of 3/4-inch plastic pipe, as shown) to prevent the plug from being connected incorrectly.

(002007)

Table 2. Arrangement of the Nintendo-64/PC game port signals

N64 Controller	PC Gameport	Line
button A	Joy A button 1	JOYAB1
button B	Joy A button 2	JOYAB2
button Z	Joy B button 1	JOYBB1
button Start/R	Joy B button 2	JOYBB2
analogue X-axis	Joy A analogue x	JOYAAX
analogue Y-axis	Joy A analogue y	JOYAAY
K/C left/right	Joy B analogue x	JOYBAX
K/C up/down	Joy B analogue y	JOYBAY

K=control cross, C=buttons

Table 3. Two typical joystick configuration files

```
// name joystick.cfg
// analog turn and look version
//
// x analog turn left/right
// y analog look up/down
// C move left/right
// C move forward/backward
// configure in game: A alternate fire, B duck, Z fire, R/Start jump
//
joyname ”N64”
joyadvanced 1
joyadvaxisx 4
joyadvaxisy 2
joyadvaxisz 1
joyadvaxisr 3
joyadvaxisu 0
joyadvaxisv 0
joyforwardsensitivity -1.0
joysidesensitivity 1.0
joypitchsensitivity -1.0
joyyawsensitivity -1.0
joyforwardthreshold 0.1
joysidethreshold 0.1
joypitchthreshold 0.1
joyyawthreshold 0.1
joyadvancedupdate
```

Alternative version:

```
// name joystick.cfg
// analog turn and move version
//
// x analog turn left/right
// y analog move forward/backward
// C look up/down
// C move left/right
// configure in game: A jump, B alternate fire, Z fire, R/Start duck
//
joyname ”N64”
joyadvanced 1
joyadvaxisx 4
joyadvaxisy 1
joyadvaxisz 2
joyadvaxisr 3
joyadvaxisu 0
joyadvaxisv 0
joyforwardsensitivity -1.0
joysidesensitivity 1.0
joypitchsensitivity 1.0
joyyawsensitivity -1.0
joyforwardthreshold 0.1
joysidethreshold 0.1
joypitchthreshold 0.1
joyyawthreshold 0.1
joyadvancedupdate
```