Bury the Hatchet, Unbury the Axe (4)

Typing and displaying, *serially*

By **Wouter Spruit**
(Netherlands)

The previous articles in this series [1] brought to you by Elektor.POST showed how to program a PICAXE chip and how to build interfacing circuits to provide both digital and analog inputs and outputs to a PICAXE project. Now we can also allow the PICAXE to switch and control various other electronic circuits, and we know how to select adequate electronic components for our own circuit designs. Additionally, in Part 3 we explored various uses for pulsewidth modulation (PWM), including servo control and sound (by the way, did you recognize the tune?). This article concludes the series, and as promised we will be adding an OLED display and a PS/2 keyboard to a PICAXE. Finally, we will connect the PICAXE to a PC via a serial connection.

Déjà vu—déjà entendu

This time, the basic PICAXE setup is different from the one used in the three previous installments. Instead of the PICAXE 08M2 chip, the PICAXE 18M2 is used (the pinout is shown in **Figure 1**). It is still powered from the 5 V rail of an ATX power supply. The Circuits are built on the same solderless breadboards. Note that the download circuit breadboard adapter (AXE029) uses the "18" jumper setting, making the connections line up with the download pins on 18M2 parts. In this article, the USB-to-serial cable (**Figure 2**) is used for more than just programming the PICAXE—it will act as an interface for serial communications between the PICAXE and a PC. The PICAXE was programmed with LinAXEpad software version 1.5.0 for (Arch) Linux. To avoid confusions, any pin number in schematics refer to the physical pin number on the chip, and pin numbers in code listings refer to internal pin names according to Figure 1.

For examples on how to program a PICAXE chip (really?), please refer to the previous article [1], or the PICAXE Manual found at [2]. PICAXE chips and various peripherals are available through the Revolution Education webstore[3]. Theory behind the selection of components for electronic circuits can be found in the second article in the PICAXE series [1].

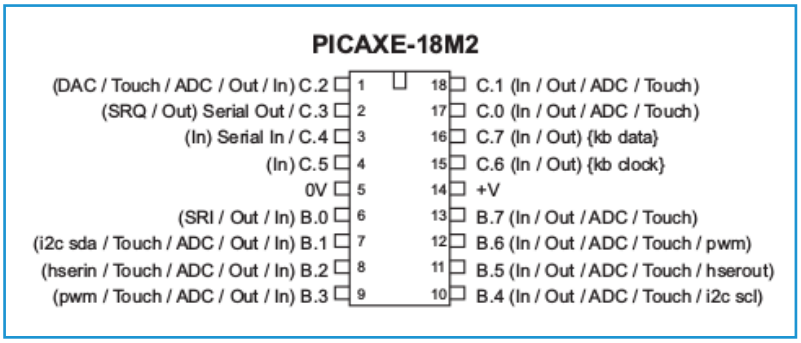

Figure 1.
PICAXE 18M2 pinout.

Serial communication

Most microcontrollers are able to communicate through a serial connection. “Serial” means the data is sent and received as a sequence of bits through a single wire. A common standard for serial communication is RS-232 (for instance, the serial port on a PC). The original specifications of RS-232 describe voltages of ±15 V, but most modern microcontrollers ignore the voltage specified in the original standard, only using the protocol for actual coding and timing of serial data. A common IC that provides “real” RS-232 communication, including voltage, is the MAX232 IC. PICAXE serial commands have separate modes for regular serial communications (baud rate starts with an “N”), and “true” RS-232 (baud rate starts with a “T”).

Serial communication with devices such as a PC or Raspberry Pi enables a PICAXE project to be integrated seamlessly into other projects, with the PICAXE acting as a buffer for any input and output circuits hooked up to it. The following paragraph explains how to set up a serial connection between devices, but you can try out the examples first if you like. The theory is useful for configuring connections between parts in your own designs. The PICAXE knows two types of serial input and output commands. The first set is used only on the pins (usually) connected to the serial cable: “`sertxd`” for output and “`serrxd`” for input. The second set of `serout` and `serin` are used for serial communications on other pins that support serial communications. The main reason we are using the PICAXE 18M2 instead of the 08M2, is the absence of pins for `serin`/`serout` and `kbdata`/`kbclock` on the 08M2.

A serial connection is set-up with a number of parameters, describing what characters to expect and how to interpret them. All data is sent in frames—a frame consists of (in sequence): 1 start bit, 5 to 9 bits of binary data, sometimes a parity bit, and (a) stop bit(s). The parameters of a serial connection tell the machine the number of frames per second (i.e. “baud rate”), the number of data bits per frame, if parity bits are used (“`n`” for no), and the number of stop bits. The properties of the serial connection from and to PICAXE are fixed at 8 data bits, no parity bit and 1 stop bit per frame. The baud rate has to be specified for the `serout`/`serin` commands. For communications through the download pins, the baud rate is fixed at 4800 (except on X2 parts, for those it would be 9600).

Figure 2.
USB-to-serial cable. Looks familiar, doesn’t it?

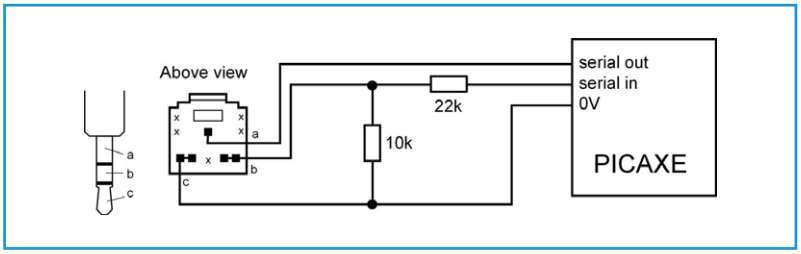

Figure 3.
Recapping: how to connect the download cable.

Connecting to a PC

Connecting the download cable has been discussed in previous articles in this series [1], but in case you missed it, the schematics from the PICAXE manual are also presented in **Figure 3**. A serial connection to a PC is already in place—it is used for downloading programs to the PICAXE! You can use this connection to communicate with the PC in your programs, too. The `sertxd` command allows you to send data to the PC—this works without any additional configuration. To view the data sent through this connection, a terminal program is required on the PC, and one is included in the LinAXEpad software in the menu PICAXE → Terminal... (F8). Data stored in a PICAXE register, for instance b0, is sent as raw binary data. This data is interpreted as ASCII code [4]. To send the value stored in b0 as readable text (i.e. a sequence of ASCII-encoded characters), instead of the raw ASCII value stored in b0, use #b0 instead (this does not always work with the obsolete PICAXE parts though). The code in **Listing 1** tells the PICAXE to increment the value of a register in a loop, then send the value first as an interpreted number (i.e. a readable sequence of ASCII

characters), then as the raw ASCII. Because only part of ASCII consists of readable text, the other characters are either interpreted as special or unprintable characters, or control sequences. The values “13” and “10” sent in the `sertxd` command are raw ASCII, used as control sequences to indicate “carriage return” and “line feed” respectively. This places the cursor at the start of a new line.

Listing 1: PC_OUT

```
main:
b1=0
do
  b1=b1+1
  sertxd(“The value of #b1 is “,#b1,13,10) ‘interpret  number as text
  sertxd(“The value of b1 is “,b1,13,13,10) ‘interpret number as ASCII
  pause 1000
loop
end
```

Listing 2: Loopback

```
init:
disconnect ‘PICAXE no longer scans for program downloads
main:
do
  serrxd [10000,timeout],b0  ‘wait 10 seconds for input, then goto timeout
  sertxd(“character received: “,b0,13,10)
  if b0 = “q” then
    sertxd(“q received, type quit to exit”,13,10)
    serrxd [5000,timeoutmain],(“quit”)
    goto quit
  endif
loop

quit:
reconnect
sertxd(“quit received. program done.” ,13,10)
end

timeout:
reconnect
sertxd(“Input timed out”,13,10)
‘goto main ‘only uncomment this if you know
‘how to reprogram the unconnected device!
sertxd(“Downloading of programs re-enabled.”,13,10)
end

timeoutmain:
sertxd(“no quit received within 5 sec. restarting program.”,13,10)
goto main
```

Receiving serial data through the “download pin” connection requires more configuration, because the pin is always in use by the PICAXE firmware (it scans for new downloads). In order to use it to receive serial input data, the pin has to be “disconnected” first. No new programs can be downloaded until the pin is reconnected—or a soft- or hard reset is performed. A timeout is available for the serial input commands on newer PICAXE parts. The program in **Listing 2** shows how to enable and disable serial inputting through the download cable, how timeouts are used, and how to receive a command from a PC terminal. We will open a terminal now (just hit F8) to communicate with the PICAXE.

Keyboard input

PS/2 keyboards send data over a serial connection, but they also require a clock signal. The pinout of a PS/2 keyboard connector is shown in **Figure 4**. Several PICAXE parts have pins marked as “kb data” and “kb clock” (pins 16 and 15 on the 18M2), meant specifically for connection to a PS/2 keyboard. The `kbin` command waits for data from a keyboard (with timeout), while the `kbled` command allows control of the Scroll Lock, Num Lock and Caps Lock LEDs on the keyboard. Note that the `kbled` command will halt program execution until a keyboard is connected. Keyboard data is sent as scancodes. An overview of scancodes may be found in the `kbin` command section in part 2 of the PICAXE Manual [5]. Most characters are sent as 8-bit serial data.

Now we will connect the keyboard according to the schematics depicted in **Figure 5**. Because USB keyboards are now more common than PS/2 keyboards, some stores don’t carry them anymore. Other stores, mostly second hand, still sell them, often at a very low price. If you can’t find a PS/2 connector, this step can be improvised, for instance by cutting of the PS/2 connector from the keyboard cable and connecting the four wires the correct way. The program in **Listing 3** receives input from the keyboard, and sends it to a PC, through the download cable. F8 opens a terminal interface from within (my version of) the PICAXE programming software.

Adding a display

In this example, the AXE033Y serial/I2C OLED 16x2 alphanumeric character display was used, connected to the “hserial” pins of the PICAXE. The AXE033Y comes with several extra features (7 memory slots for pre-programmed text, functions as stand-alone clock with optional clock module), but for now we will use it as a simple character display. Other serial character displays are also expected to work with this example. Data is sent to the display through a serial connection, encoding of characters is ASCII again. Several ASCII codes are interpreted as display commands, instead of characters to display. An overview of character codes and commands can be found in the PICAXE Manual part 3 [6], pages 32 and 41 respectively.

Connect the serial display according to the schematics of **Figure 6**. Before programming PICAXE with relatively complicated display code, we have to make sure the connected display works as expected. **Listing 4** shows how to prepare the display, and how make it show “Hello Elektor!”.
The display uses control sequences to position the cursor. Every time a character is sent to the display, it increments the character position automatically. However, in order to use the display to give feedback to the input from a keyboard, special characters like “Backspace” have to be coded differently. Note that

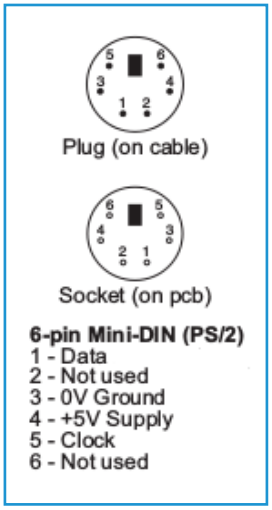

Figure 5.
Pinout of a PS/2 keyboard connector.

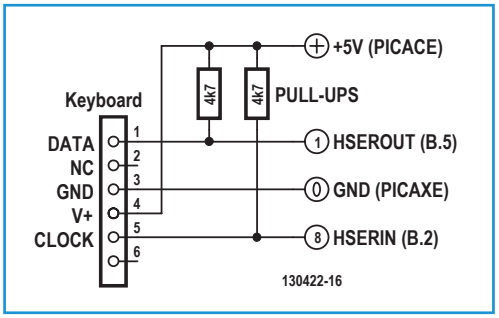

Figure 6.
How to connect a keyboard.

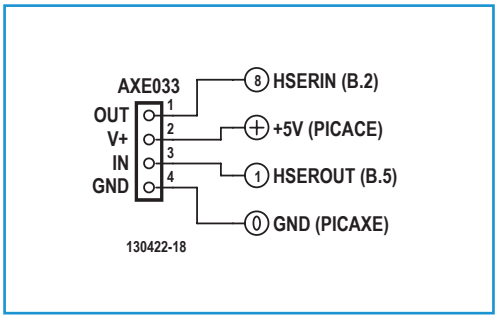

Figure 8.
Connecting the display.

Listing 3: Data from keyboard to PC

```
main:
do
  kbin b1' get keyboard input
  sertxd("Received scancode from PS/2 keyboard: ",#b1,13,10) 'interpret
number as text
loop
end
```

Listing 4: Hello ~~World~~ Elektor!

```
pause 500         `allow screen to initialize first
serout B.5,N2400,(254,1)   `send clear command to screen
pause 30        `wait for the screen
serout B.5,N2400,(254,128)   `move screen cursor to line 1, char 1
serout B.5,N2400,("Hello Elektor!") `send text to display
end
```

scancodes and ASCII are not directly compatible! An example on how to convert keyboard input in order to display it on the screen is shown in **Listing 5**; the conversion has to be done for all characters/keys you need (the codes are not sequentially compatible!). Connect both the display and keyboard as described previously. The only keys enabled are the characters "a", "b", "c", "h", "i" and "space"; a "Backspace" function is included and the "Enter" and "Escape" keys restart and end the program respectively.

Display memory actually consists of two lines of 40 characters, but only the first 16 characters of every line are displayed. The common scrolling effect can be achieved by moving the window of visibility relative to the text stored in memory, as demonstrated in the "quit" function in Listing 5.

Listing 5: KEYBOARD_DISPLAY

```
init:
pause 500  `allow screen to initialize first
gosub initscreen
serout B.5,N2400,("Enter text:")
gosub setcursor
gosub showcursor
do
  kbin b1
  if b1 = $1C then 'a
    b2=97   'convert to lower case ASCII a
    gosub printcharacter
  elseif b1 = $32 then 'b
    b2=98
    gosub printcharacter
  elseif b1 = $21 then 'c
```

```
    b2=99
    gosub printcharacter
  elseif b1 = $33 then ‘h
    b2=104
    gosub printcharacter
  elseif b1 = $43 then ‘i
    b2=105
    gosub printcharacter
  elseif b1 =  $29 then ‘space
    b2=32
    gosub printcharacter
  elseif b1 = $66 then ‘backspace
    gosub backspace
  elseif b1= $76 then ‘escape quits the loop
    goto quit:
  elseif b1= $5A then ‘enter resets the program
    goto init:
  endif
loop

quit:
gosub initscreen
gosub hidecursor
serout B.5,N2400,(“It is now safe to turn off the PICAXE”)
do
  serout B.5,N2400,(254,24) ‘ move the window left
  pause 100
loop
end

initscreen:
serout B.5,N2400,(254,1) `clear screen
pause 30
serout B.5,N2400,(254,128) `move to start of first line
symbol CURSOR_POS = b3
CURSOR_POS = 0
return

setcursor:
b4=CURSOR_POS+192   ‘192: start of second line
serout B.5,N2400,(254,b4)
return

showcursor:
serout B.5,N2400,(254,14)  ‘turn on cursor
return

hidecursor:
serout B.5,N2400,(254,12)  ‘turn off cursor
```

```
return

printcharacter:
if CURSOR_POS > 15 then return endif  'only use visible character space
serout B.5,N2400,(b2)
pause 200
CURSOR_POS=CURSOR_POS+1
return

backspace:
if CURSOR_POS = 0 then return endif 'already start of line
CURSOR_POS=CURSOR_POS-1
gosub setcursor 'move back one space
serout B.5,N2400,(32) 'overwrite with space character
gosub setcursor 'set cursor to correct position
pause 100
return
```

Evidently, serial displays come with various functions and options, and to cover all functionalities of such displays would not fit in this article. Details on the AXE033 display can be found in the manual [7]. Feel free to explore the possibilities further.

Terminal settings

In order to fully integrate the PICAXE with an existing (or new) system—like for home automation applications—the PC (and user) should be able to use a serial connection to communicate with the PICAXE, without having to rely on the terminal program included with the programming editor. "minicom" [8] is a terminal program that works on Linux and other POSIX-compatible systems (like Mac OSX), but this example will assume Linux. In order to use it, you have to make sure the (module for the) serial download cable is in use. LinAXEPad can help you enabling the correct module for the USB-to-serial cable (view → options → port → AXE027 modprobe), and showing the address of the device (probably "/dev/ttyUSB0"). To install "minicom" on Debian systems (including the Raspbian OS on the Raspberry Pi), we may enter `sudo apt-get install minicom` (in case we're using Arch, then it would be `sudo pacman -S minicom`). Let's configure "minicom" by entering the setup mode: `sudo minicom -s`. Select the option "Serial port setup" (arrow keys, enter), and type "A" to change the device path to the path of the USB download cable (in our case, "/dev/ttyUSB0"). Type "E" to change the serial speed, and use the "A" and "B" keys to select 4800 baud. Make sure the program expects 8 data bits, no parity and one stop bit (8N1). Use "enter" to return to the main menu, and select the save settings as "dfl" (default) option. After selecting "Exit" (NOT "Exit from minicom"), you can communicate with the PICAXE through "minicom" (now you can also start it with `sudo minicom`). If you want to exit "minicom", use CTRL+A, then X, then YES (enter).

Some examples will be shown to demonstrate how to use serial communications with PICAXE in programs and scripts on a PC. For this example, a bash shell under Linux was assumed, with a tested and working serial connection to the PICAXE. All commands require root access. First, the serial connection properties have to be set. We should enter `stty -F /dev/ttyUSB0 speed 4800 cs8 -cstopb -parenb`. To view data from the connection, we'll be using "cat": `cat /dev/ttyUSB0`. To get a single line into a variable: `read variable < /dev/ttyUSB0`. To view the data: `echo $variable`. And finally, to send data through the connection: `echo q > /dev/ttyUSB0` (though the quit command can be sent and received using "minicom", the quit

command does not seem to work if sent using echo in bash). To summarize, a bash script to communicate with a PICAXE (programmed with the code from Listing 2 with the `goto main` line uncommented in the `timeout` subroutine), is presented in **Listing 6**. Running the script requires the executable flag to be set (thus, `chmod +x scriptname.sh`).

Over 2 U now

We know how to make a PICAXE chip talk to peripherals through a serial connection! Now it's also possible for a PICAXE project to make use of an OLED display to show data, and to receive user input from a PS/2 keyboard. Two-way communication with a PC through the USB-to-serial download cable allows for easy debugging of PICAXE programs with the terminal application included in the PICAXE programming editor (or equivalent software). Through a serial connection, the PICAXE can be used to control your own interfacing electronics, possibly through a web interface, on a PC or a Raspberry Pi. The PICAXE system allows us to build complex systems, like home automation and robotics, very quickly and at a low cost. With the obvious exception of code and functionality provided by PICAXE-specific software, the principles shown in this series are also applicable other types of microcontroller. For additional resources you may also check the PICAXE series support page at Elektor.LABS [9] where, among others, all listings are hosted in single files for your convenience. So here ends our part, now it's your turn. Good luck with your projects!

(130422-I)

Internet Links

[1] "Bury the Hatchet, Unbury the Axe" (1, 2 and 3), Elektor.POST Projects Nos. 8, 16 and 19. www.elektor-magazine.com/extra/post

[2] www.picaxe.com/Getting-Started/PICAXE-Manuals

[3] www.techsupplies.co.uk/PICAXE

[4] www.asciitable.com

[5] www.picaxe.com/docs/picaxe_manual2.pdf

[6] www.picaxe.com/docs/picaxe_manual3.pdf

[7] www.picaxe.com/docs/axe033.pdf

[8] http://linux.die.net/man/1/minicom

[9] www.elektor-labs.com/PICAXE

Listing 6: BASH_SERIAL

```
#!/bin/bash
#run as root.
export DEVICE='/dev/ttyUSB0'  #the serial device to use
#initialize
echo setting up device $DEVICE
stty -F $DEVICE speed 4800 cs8 -cstopb -parenb
echo waiting for input...
read INPUT < $DEVICE
echo received: $INPUT
sleep 1
echo sending a "q" character
echo q > $DEVICE
echo waiting for response...
read INPUT < $DEVICE
sleep 1
echo response: $INPUT
echo done
```