Bury the Hatchet, Unbury the Axe (2)

I/O circuits for PICAXE

In this article series we cover a range of topics relevant to the design of electronics projects based on the PICAXE range of microcontrollers. This second installment is about circuits to provide inputs and outputs for a PICAXE chip, using common electronics components. The story also aims to show you what factors to consider/calculate when picking components for your own interfacing circuits.

By **Wouter Spruit** (Netherlands)

The first article in this series brought to you by Elektor.POST was an introduction to the PICAXE system [1], including examples on how to build the PICAXE programming circuit, and how to program a PICAXE chip to control an LED with a pushbutton. The upcoming articles will cover more advanced, chip-specific interfacing capabilities available through PICAXE. The capabilities of a project based on a PICAXE chip can be enhanced by the addition of special ICs and peripherals, like extra memory, keyboards, LCD displays, or even a PC through a serial connection.

The truth is out there, isn't it?

A wealth of information is available on the Internet to anyone who is just starting to design their own microcontroller-based electronics. Unfortunately in the same place it can be very hard to find sufficient information on the underlying theory as well as specific examples for interfacing circuits. It is easy to see what components go where in sample circuits, but it's not clear why exactly those components got selected. For the following interfacing examples, some theory on the components is included to make sure you know exactly how to select the right components for your own designs that may be similar—but different.

Getting ready: the experimental setup

All experiments described next are performed on a PICAXE 08M2 (pinout is shown in **Figure 1**), operating at 5 volts thanks to an ATX power supply. Example circuits are built on solderless breadboards. Programming of the PICAXE chips is done through the USB-to-serial cable following the method for programming the PICAXE chip, described in the previous article [2], from LinAXEpad software version 1.5.0 for (Arch) Linux. To avoid any confusion, all pin numbers in schematics refer to the physical pin number on the chip (for

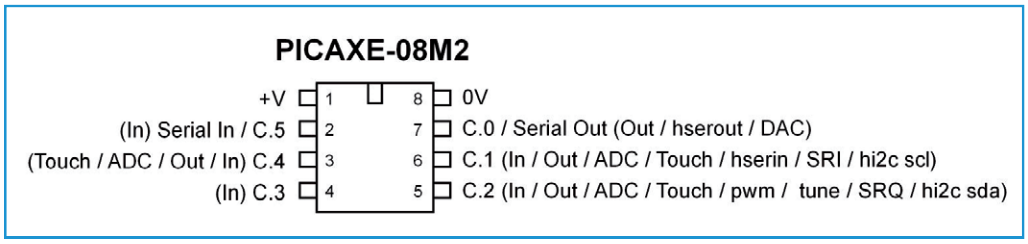
PICAXE-08M2
+V 1
(In) Serial In / C.5 2
(Touch / ADC / Out / In) C.4 3
(In) C.3 4
8 0V
7 C.0 / Serial Out (Out / hserout / DAC)
6 C.1 (In / Out / ADC / Touch / hserin / SRI / hi2c scl)
5 C.2 (In / Out / ADC / Touch / pwm / tune / SRQ / hi2c sda)

Figure 1.
PICAXE 08M2 pinout.

all examples: pin 3 is an output, pin 4 is an input), pin numbers in code listings refer to internal pin names according to Figure 1. For examples on how to program a PICAXE chip, you may refer to the previous article [2], or the PICAXE manual on the website [3]. PICAXE chips and various peripherals are available through the Revolution Education online store [4].

Protecting your components

Components have maximum ratings for voltage and current. Exceeding those maximum values will likely damage them permanently. Current, voltage and resistance are related according to Ohm's law:

Voltage (V) = Current (I) × Resistance (R).

Consequently the current through components is a direct result of the voltage potential between two points and the resistance between those points. With a fixed DC power source, for instance 5 V DC for a PICAXE chip, this means current can be limited by adding resistance in series. The resistance of resistors in series becomes higher for every resistor:

R = R1 + R2 + R... + Rn

while the resistance of resistors in parallel becomes lower for every resistor added:

1 / R = 1 / R1 + 1 / R2 + 1 / R... + 1 / Rn

The current through components in series is constant everywhere, while the resistance varies per component. From Ohm's Law it follows that the voltage potential gets divided across the components, proportional to their resistance. At this point, everybody with an interest in electronics should be familiar with it, but more information on Ohm's Law is readily available on the Internet, for instance on [5]. And it's never too late to learn more! Understanding how the following works makes calculating resistors in real circuits much easier. Consider the voltage divider depicted in **Figure 2**. Resistors R1 and R2 are connected in series, thus the current is constant everywhere from the voltage source to ground. However, the voltage drop across R1 and R2 differs according to their resistance. Because Ohm's Law applies to potential differences,

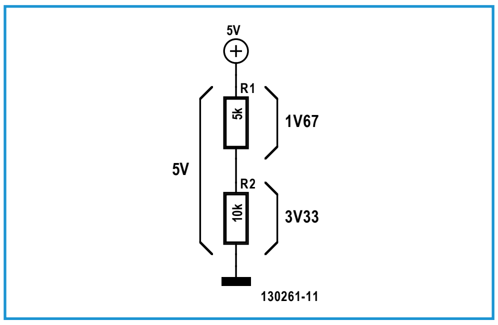

Figure 2.
Example of a voltage divider.

the correlation between voltage, current and resistance also works on just a part of a circuit. As it can be seen in Figure 2, the voltage across R1 + R2 is 5 V, and the voltage across R1 becomes:

5 × (R1 / (R1 + R2))
5 × (5 kΩ / 15 kΩ) = 1.67 V

this is implied by $V = R \times I$. Note that logically, the remaining voltage on R2 equals

5 – 1.67 = 3.33 V
= 5 × (R2 / (R1 + R2))

It's important to remember this is true for resistive components, but not necessarily for all others. For instance, diodes (including LEDs and optocoupler inputs—also LEDs—and transistors. To calculate the resistor to limit current to a LED, use the voltage drop of the LED instead of trying to measure its resistance with a multimeter! Example: to limit a red LED with a voltage drop of 1.8 V powered from a +5 V source with a current of 10 mA, we would calculate the resistor value as follows:

voltage across the resistor = 5 – 1.8 = 3.2 V.

Now we use $V / I = R$ to find 3.2 / 0.010 = 320 Ω. The closest standard resistor value from the E6 series is 330 Ω.

The principle of voltage division may be very useful combined with microcontroller inputs. Consider how R1 or R2 in the circuit of Figure 2 could be replaced with a variable resistor of some sort, for instance to measure temperature or light intensity. A microcontroller input pin would be connected between R1 and R2, measuring a voltage proportional to the sensor input through its A/D converter. Since there is a limit on the current every chip can sink (acting as ground) or source (voltage gets supplied from a pin), it is important to limit the current flowing to and from the chip in any (interfacing) circuit. However, in many cases, peripherals require more current to function than the chip could possibly supply without getting overloaded.

The transistor

Because discussing the specifics of all transistor types in existence is way beyond the scope of this article, the commonly available BC547B (or equivalent) NPN transistor will be used to show how to overcome the current limit on microcontroller pins. The idea is that a switching transistor is used because it is able to cope with the current required for peripheral operation, where current limits on pins are a problem.

An NPN transistor regulates the current flowing from its collector to its emitter pin, depending on the voltage applied on its base. A small current through the base pin leads to a larger current flowing through the collector/emitter pins, so the transistor works as an amplifier until the transistor is not limiting any current anymore. At that point the transistor is said to be saturated. The PNP transistor type has basically inverted switching action, but that is beyond the scope of this article.
Armed with this knowledge, we will build a circuit to test the transistor as a switch to control an LED: current from a PICAXE output pin controls a transistor, which in turn switches on the LED. The PICAXE output pin only needs to source enough current to switch the transistor—in this case, the current to light the LED comes directly from the power supply.
The transistor example uses the same push-button circuit as in the introductory article of the PICAXE series. **Figure 3** represents the transistor circuit used by the author. The setup with the simple switch in **Figure 4** should be added too. The required code for the PICAXE chip is included in **Listing 1**. It can also be downloaded as a text file, directly from the support page of this article series in Elektor**.** LABS [6]. **Figure 5** shows the whole circuit assembled in a breadboard, with the push-button. The LED will light up when you push the button.
If in your own designs you use the transistor to switch a load, make sure to check the maximum voltage (through the relevant pins) and maximum collector current (I_c) ratings for the transistor. Factors like switching speed and gain (h_{FE}) are less relevant for "slow" and all-or-nothing switching action, than for high-speed or accurately controlled amplification respectively. To calculate the resistor value to limit the current from an output pin

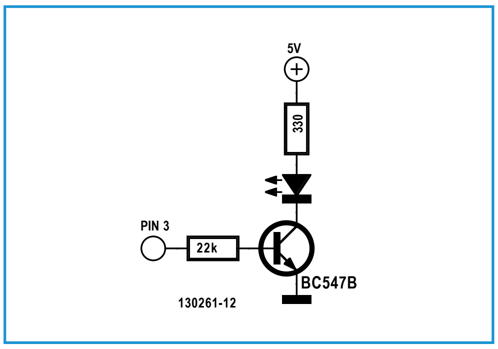

Figure 3.
Transistor circuit.

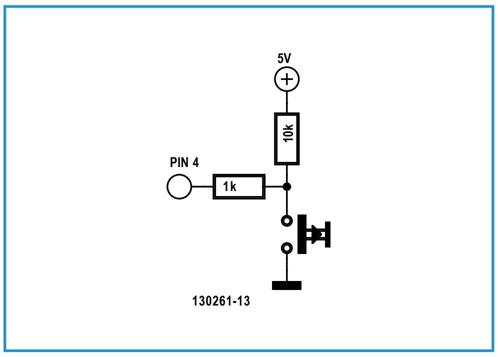

Figure 4.
Push-button circuit, as explained already in the introduction article.

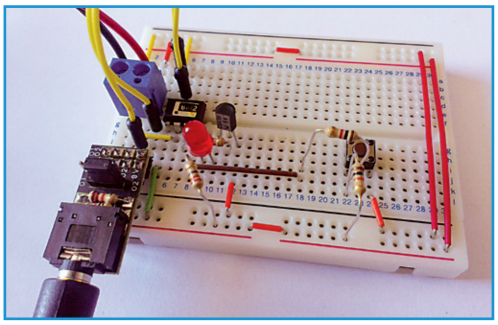

Figure 5.
Experimental setup with the transistor circuit and a pushbutton on a breadboard.

Listing 1: Switch code

```
do                    ;repeat forever
  if pin3=0 then      ;if the button is pressed
    high 4            ;output high
  else                ;if the button is NOT pressed
    low 4             ;output low
  endif               ;closes if statement
loop
```

through the transistor base, make sure it's lower than the maximum current the pin could provide, and high enough to saturate the transistor—this is where gain comes in. To calculate the minimum current through the base, divide your anticipated load current by the transistor gain (h_{FE} = 200), and if you want to be safe you may at least double it (even up to 5 times might be needed sometimes!), to make sure the transistor is fully saturated.

For an LED load drawing 10 mA, the current through the base pin should be 10 / 200 = 0.05 mA, doubling this gives 0.1 mA (which is 100 times less current for the microcontroller to source).
Now calculate the resistor value as done in the potential divider example. Starting from 5 V, the transistor takes 0.7 V like a diode [7], leaving 4.3 V across the current limiting resistor and with $R = V / I$, so

R = 4.3 / 0.0001 = 43,000 Ω (43 kΩ)

However, for a load heavier than 10 mA, the resistor should be smaller, and for convenience, a smaller resistor will be used, allowing the maximum current the transistor could take, 200 mA, because these microcontrollers can easily sink the 2 × 200 / 200 = 2 mA required. For this resistor: R = 4.3 / 0.002 ≈ 2.2 kΩ. A practical consideration: I could have many 1-kΩ resistors, but no 2.2-kΩ ones, I could solve this by adding two 1-kΩ ones in series, or, if I don't care about power (this is, the circuit is not running by means of batteries), I would just use 1 kΩ, as long as the microcontroller could source it easily. This might be why many examples use 1-kΩ resistors to limit current to base pins even for low-current loads. Also, other common transistors might have lower gains!

The Darlington pair

In 1953, Sidney Darlington came up with the idea of increasing a transistor's amplification with a second transistor [8], to make sure it behaves as a switch instead of an amplifier, as the transistors' gain factors are multiplied (for high gain transistors), see **Figure 6**. It is no surprise to see that several PICAXE project boards (like the AXE002U, PICAXE-18M2 Starter Pack) come with Darlington pairs as

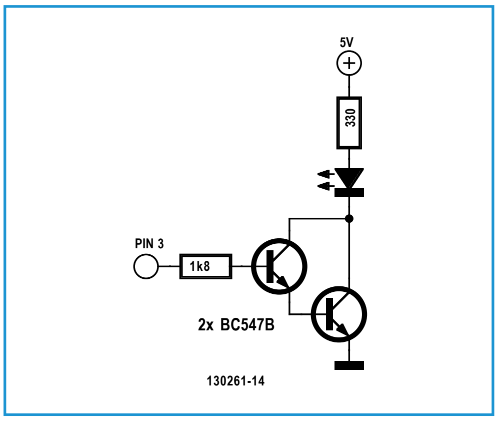

Figure 6.
Equivalent circuit of a Darlington pair.

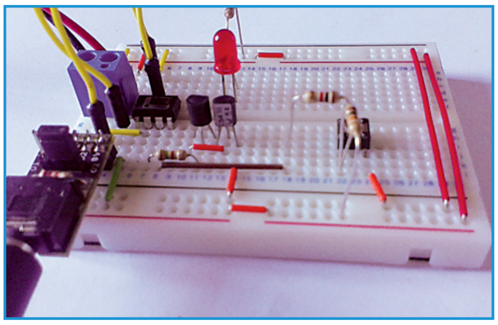

Figure 7.
Darlington pair circuit on a breadboard.

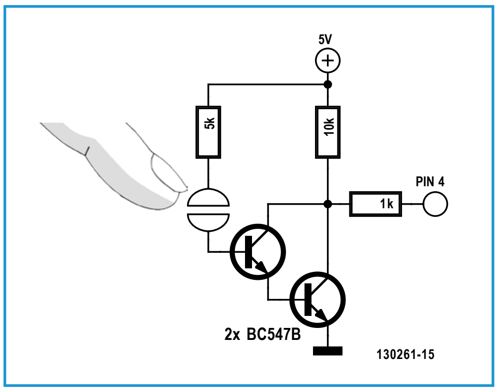

Figure 8.
A Darlington pair used as a touch-sensitive switch.

switches to control higher power peripherals with PICAXE.

Build the Darlington pair circuit from Figure 6 and add the switch from Figure 4. Afterwards program it with the now familiar code from Listing 1. The completed experimental setup of the circuit is shown in **Figure 7**.

Selecting transistors to make a Darlington pair is much like selecting single transistors, but consider that each B-E connection causes a voltage drop. Switching two transistors this way is a bit slower than switching one (whether switching speed is an issue depends on the application). Calculating the current limiting resistor to 2 mA like before, now means

5 – 0.7 – 0.7 = 3.6 V

yielding 3.6 / 2 = 1.8 kΩ. However, the current gain is much higher (200 × 200 = 40,000), so the 2 mA base current may drop by a factor of 200, requiring a resistor value of 1.8 kΩ × 200 = 360 kΩ.
Because such resistance values are well above the highest ever measured resistance of the skin of human fingertips [9], we can also use a Darlington pair as a touch-sensitive switch, according to the circuit depicted in **Figure 8**. The PICAXE code needed remains the same. You might want to limit current through the base pin and fingertip contacts, to prevent damage in case of a short-circuit by accidental contact with a conducting item, and to make sure no painful currents (>4 mA, for more info see [10]) can flow through the fingertip skin, although it's very unlikely ever to happen from a 5-V source under normal conditions [11]. The resistor should limit current through the finger to 1 mA to be sure: 5 / 0.001 = 5 kΩ. Build the Darlington pair touch switch circuit according to Figure 6 as the output and the one in Figure 8 as the input. This example again uses the same code from Listing 1. **Figure 9** shows the finished circuit on a breadboard, while in **Figure 10** it can be seen in action.

The optocoupler

In some cases, it is disadvantageous to power the microcontroller circuit from the same circuit or power supply as the device to control. An optocoupler is used to switch an optically isolated circuit: both circuits can have their own power supply, they are not physically connected in any way, they are switched by means of light. The device consists of a LED and a phototransistor in a single package. When the LED is on, the phototransistor allows current to flow through the second circuit, otherwise, it acts as a closed switch. Some types are only suitable as a binary switch, while others allow analogue signals. Selecting an optocoupler can be hard because there is no easy way (that the author is aware of) to find the perfect optocoupler suited to your needs from the wide variety of types (with wildly varying specifications) scattered all over the Internet and physical stores. If you can't ask someone with experience in selecting an optocoupler for your specific needs, you might end up spending hours browsing data sheets. For this example, I will use the 4N25. This optocoupler would probably work for most (low voltage) applications, but check the data sheet for LED voltage/current and photo transistor current and voltage ratings, as

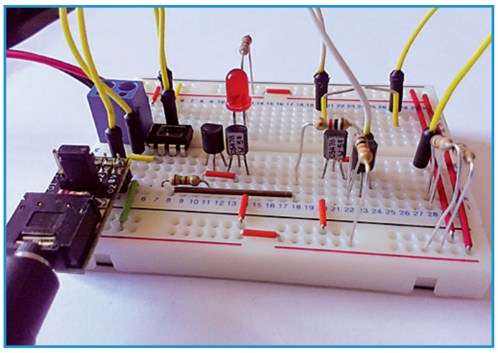
Figure 9.
Circuit of a touch switch on a breadboard.

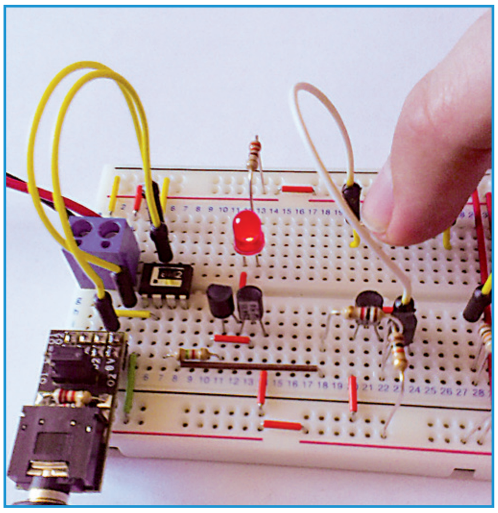
Figure 10.
The touch switch in action.

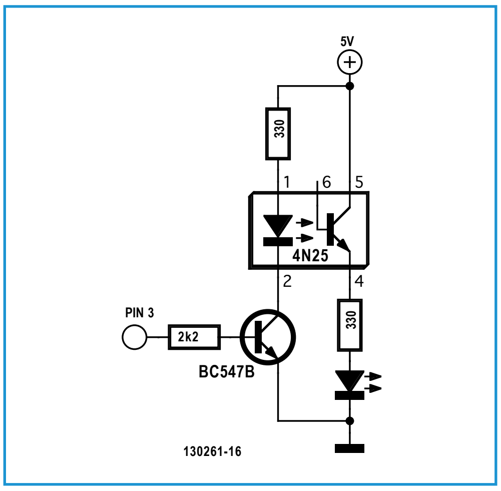

Figure 11.
Schematic of an optocoupler setup.

well as switching speed, if relevant to your application.
Build the example connecting the circuit of **Figure 11** to the output pin, and for the input choose between the basic switch of Figure 4 or the touch-sensitive version of Figure 8, using the same example code from Listing 1 for either. The completed circuit is shown in **Figure 12**.
In the example, a transistor is used to switch the optocoupler, but in most cases, the optocoupler could be connected directly to a microcontroller pin via a resistor to limit current through the optocoupler. To calculate the current limiting resistor, a forward voltage of 1.3 V was used, aiming for a current of about 10 mA [12], to give R = (5 – 1.3) / 0.01 = 370 Ω, normalized to 330 Ω.

Relays

If you want to switch mains-powered appliances with your PICAXE, the AC powerline circuit needs to remain isolated from the PICAXE power supply. Relays act by physically opening and closing a switch (for instance for switching on your desk lamp) by means of a solenoid: applying voltage to the solenoid changes the state of the switch, removing power from the solenoid restores the original state. Depending on how the relay is connected to the (mains) circuit, default switch position can either be On or Off. Bi-stable relays “remember” their state, so no power is consumed to maintain either state, only to switch. Relays being mechanical devices, they can be switched a limited number of times before they break down, making them less suitable to applications where rapid-switching is required. Solid-state relays (no moving parts, similar to optocouplers) can be used instead of traditional relays to solve the mechanical limitations, but they are significantly more expensive.
When a relay closes, it causes a voltage spike that is harmful to electronics, so a ‘back-emf’ suppressor diode is often fitted ‘ín reverse’ across the relay coil. The diode not conducting except when a spike occurs, it does not draw any significant current (check diode current and voltage ratings). Relays often require a higher supply voltage than microcontrollers. All things considered, it might be best to control a relay through an optocoupler instead of just a transistor, to keep the relay supply

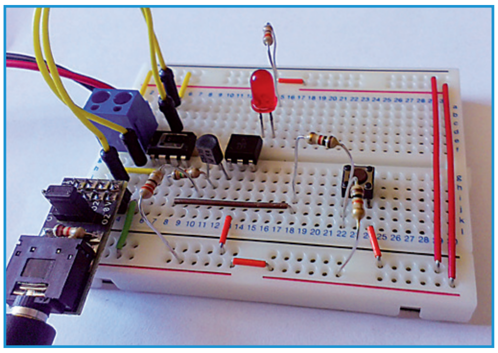
Figure 12.
Optocoupler circuit on a breadboard, including a basic push-button addition.

Listing 2: On/Off code

```
b0 = 0               ;set state 0 (OFF)
b1 = 0               ;reset last pin state
low 4                ;reset output
main:
do                   ;repeat forever
  if pin3=0 then     ;button is pressed
    gosub btnpress   ;jump to this label
  else
    if b1=1 then     ;check button release
      b1=0           ;reset pin state variable
    endif
  endif
loop

btnpress:            ;btnpress label (for gosub)
  if b1=0 then       ;is this the first visit since
                     ;button release?
    b1=1             ;mark as visited
    gosub setstate   ;jump to label
    pause 100        ;prevent switch bounce
  endif
return               ;return to gosub statement

setstate:            ;setstate label
  if b0=0 then       ;if current state is OFF
    b0=1             ;remember new state
    high 4           ;set pin 4 high
  else               ;current state is ON
    b0=0             ;set state OFF
    low 4            ;set output pin low
  endif
return
```

voltage (and unfortunate power spikes) separate from the microcontroller.
All previous examples have used a switch or button to give output only when the switch is pressed, but for the relay example we will need a toggle switch: pressing it changes the buttons state between On and Off. To do so, we must modify the pushbutton code from Listing 1 to store the state (On/Off) after the button is released. There are several ways to store data on the PICAXE chip, we will go through them in upcoming articles. For now, we will use the PICAXE general-purpose variables. Every PICAXE chip has a number of them, at least 14 (called b0 to b13), and each one can hold 8 bits. If you need to store a value in 16 bits it's possible to use two general-purpose variables together. In that case w0 would be b1 and b0 together, w1, b2 and b3, and so on. Note that writing w1 would overwrite b3 and b2! In some of the newer PICAXE chips, it's also possible to combine four 8-bit variables to obtain a 32-bit one. Some variable numbers are always used for storing important data for a specific pre-programmed PICAXE functionality.
The example code in **Listing 2** shows how to use the variable b0 to store either a 0 (OFF state) or a 1 (ON state) and toggle the state using a button. The output is set High or Low accordingly. Note the extra delay after changing the state in reaction to pressing the switch: this is to prevent the switch bounce phenomenon. This is a mechanical property of switches, when pressed and released they have brief moments of rapidly bouncing between the open and closed positions. The variable b1 is used to store the input pin state: this allows us to only change the switch state after a release/press cycle, otherwise ON/OFF toggling would continue as long the button was pressed. The code uses the "`gosub`" statement for readability. "`gosub`" jumps to a label, executes code from that point right up to a Return statement, then it continues from where it jumped.
The circuit for this example is built from the relay circuit represented in **Figure 13** and the switch of Figure 4, but this time the PICAXE should be programmed with the code from Listing 2. **Figure 14** shows the finished circuit assembled on a breadboard.
Again, the optocoupler does not need to be switched by a transistor, as long as the micro-

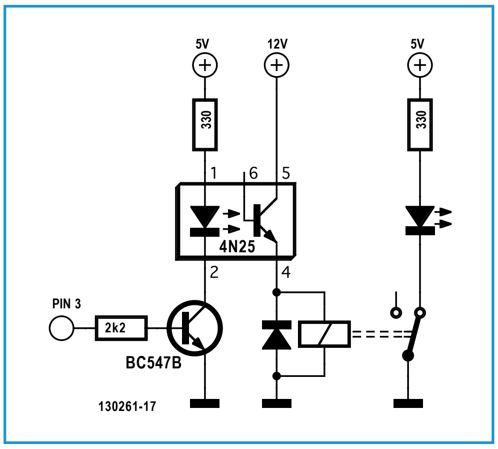

Figure 13.
Schematic of the relay circuit. It is also operated by way of an optocoupler.

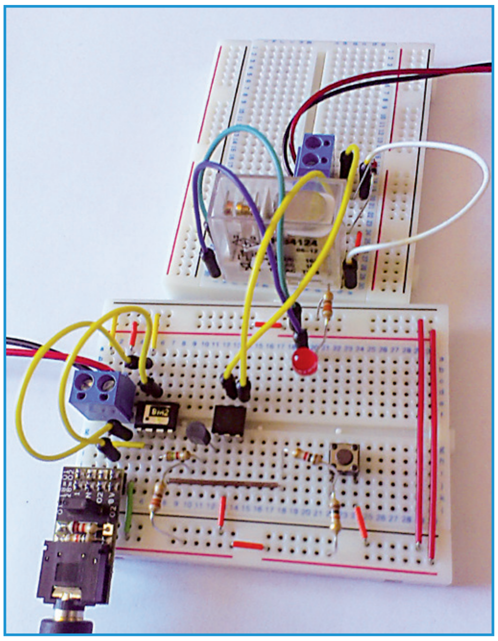

Figure 14.
The relay circuit in action—it's been wired on a second breadboard.

controller pins are able to provide the required current. In this example, the relay has its own power supply. Ground lines are not connected to the microcontroller circuit and the relay switching circuit after the optocoupler. The example uses the +5 V and GND lines (thus GND and GND3 are connected in this case) from the microcontroller circuit for the LED circuit after the relay—but depending on the relay, mains-powered appliances could be switched using the same circuit.
If you want to try this, do so safely (if you don't know what to consider when playing with mains voltage, you probably shouldn't do so). Remove the LED load and power supply from the relay output (and above all: make sure your relay can actually switch mains voltage!). To create a switch, you may "sacrifice" an

(unplugged) extension cord, cutting the wires and connecting the relay outputs between the wires, in order to reestablish the original cord connections when the relay output closes. After testing the PICAXE relay switching action without connecting the extension cord yet, connect the cord to some appliance drawing current within relay limits (like a lamp for an obvious effect) and to an outlet. Now you should be able to switch the mains appliance with a PICAXE button! A complete home-automation system is just a few steps away...
When selecting a relay, it is important to note the voltage and current required for switching as well as the maximum voltage and current it could switch. When switching a relay with an optocoupler, always make sure that the optocoupler is able to switch the relay current and voltage as well!

Let's call it a day

At this point we have covered how to wire up and program a PICAXE chip, as well as several input and output circuits with code. We know what to look for when selecting components for our own designs, and we know how to calculate resistor values to limit current to protect those components.

The next article will be about more advanced inputs and outputs, also covering servo controls, for instance to build robots. In the following installments we will be exploring the possibilities of adding integrated circuits to a PICAXE project, greatly increasing its capabilities, and even interfacing a PICAXE to a PC. Sounds great, no?

(130261)

Internet Links & Literature

[1] www.picaxe.com

[2] Elektor.POST Project No. 8: "Bury the Hatchet, Unbury the Axe"
www.elektor-magazine.com/extra/post

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

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

[5] http://hyperphysics.phy-astr.gsu.edu/hbase/electric/ohmlaw.html

[6] www.elektor-labs.com/picaxe

[7] www.farnell.com/datasheets/410427.pdf

[8] http://en.wikipedia.org/wiki/Darlington_transistor

[9] C.J. Poletto and C.L. van Doren,
"A high voltage, constant current stimulator for electrocutaneous stimulation through small electrodes",
IEEE Trans biomed. Eng., vol 46, no. 8, pp. 929-936, 1999

[10] F.A. Saunders, "Electrocutaneous displays"
in Proc. Conf. Cutaneous Commun. Syst. Devices, pp 20-26, 1973

[11] K.A. Kaczmarek, "Optimal electrotactile stimulation waveforms for human information display",
PhD thesis, Dep. Elev. Eng. Univ. Wisconsin-Madison, 1991

[12] www.vishay.com/docs/83725/4n25.pdf