Bury the Hatchet, Unbury the Axe (3)

Of Analog Inputs, PWM, Servos and PICAXE's Musical Skills

By **Wouter Spruit**
(Netherlands)

In the course of this article series [1] we first showed how to program a PICAXE chip and how to build basic input and output circuits (Elektor.POST Project No. 8). In the second installment (Elektor.POST Project No. 16) we covered the control of various types of switches, and how to calculate the appropriate component values for your designs. This time we will be looking at how to read and process values from analog inputs, describing multiple output types using circuits connected to a pulsewidth modulation (PWM) output, in combination with the already available PICAXE commands. We will also be able to position mechanical parts accurately using servos controlled by a PICAXE chip, as well as creating and playing monophonic ringtones!

A friendly reminder on how to get ready

As in previous installments, all experiments described herein are performed using a PICAXE 08M2 microcontroller (pinout repeated 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 a USB-to-serial cable following the method for programming the PICAXE chip, described in the previous articles [1], based on 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 all examples: pin 5 is a PWM output, pin 6 is analog input 1). Pin numbers in code listings refer to internal pin names according to Figure 1. For examples on how to program a PICAXE chip, please refer to the previous articles [1], or the PICAXE manual on the website [2]. PICAXE chips and various peripherals are available through the Revolution Education online store [3].

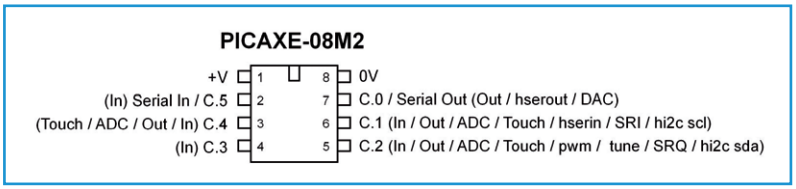

Figure 1.
PICAXE 08M2 pinout.

Analog input

All previous examples used a binary (all-or-nothing) switch to control output circuits. However, in many cases finer control is needed. The analog-to-digital converters (ADCs) on the PICAXE are able to read an analog voltage, usually ranging from +5 V (supply voltage) to 0 V, and convert it to an 8-bit (or, for some PICAXE chips, 10-bit) digital value. A potentiometer acts as an adjustable voltage divider: it is used as an analog input device. The wiper (middle terminal) is connected to

a PICAXE ADC pin, and the voltage depends on its position.
If we turn the wiper close to the 0 Ω end, the current through the ADC pin should be limited to avoid damage to components (as explained in part no. 2 [1]). However, simply adding resistors to limit current will also impact the ADC readings. A 330 Ω resistor between the wiper and the ADC pin will limit the current to

$$\frac{5V}{330\Omega} = 0.0152\ A$$

which is well within the 0.02 A sink limit for pins. For this example we will use a 10 kΩ linear pot. Thus, the maximum voltage over the ADC becomes:

$$5V \times \frac{10k\Omega}{330\Omega + 10k\Omega} = 5V \times 0.97 = 4.84V$$

This relatively minor effect can be compensated by software. Discussions on selecting the resistance of components to connect to the PICAXE ADC input can be found on the PICAXE forums [4]. The consensus on the forums is that a value of 10 kΩ should be optimal for a potentiometer connected to a PICAXE (or related Microchip PIC) ADC input. The example shows how to switch an LED on or off, using the pot's wiper position as a switch. The circuit is shown in the schematics represented in **Figures 2** and **3** (input and output respectively). The required code can be found in **Listing 1,** while **Figure 4** depicts the setup of this experiment on a breadboard. In this example, if the ADC value is between 2.5 V to 5 V, the LED is turned on, otherwise it's turned off. It's obvious that for this case it would have been better to use a simple on/off switch, but in the next example this same circuit will be used to regulate the brightness of the LED. Some cases are known where PICAXE ADC inputs behave unexpectedly, especially when the input impedance exceeds 20 kΩ or so. Should you want to know more, you may find info on the forums [4], and of course on the Internet.

PWM basics

Though the PICAXE does not integrate analog outputs, it does come with one (or more) PWM outputs. A PWM output can be used for a number of useful things, including something

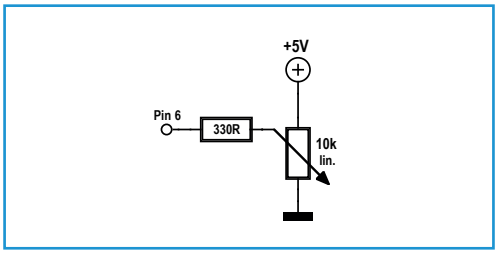

Figure 2.
Analog input schematic.

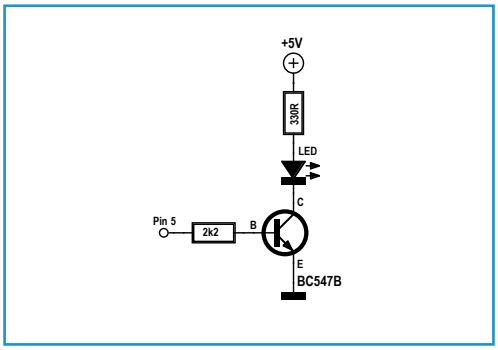

Figure 3.
PWM output schematic.

Listing 1: Analog In

```
init:
low 2
main:
do
  readadc 1,b1
  if b1 > 127 then
    high 2
  else
    low 2
  endif
loop
```

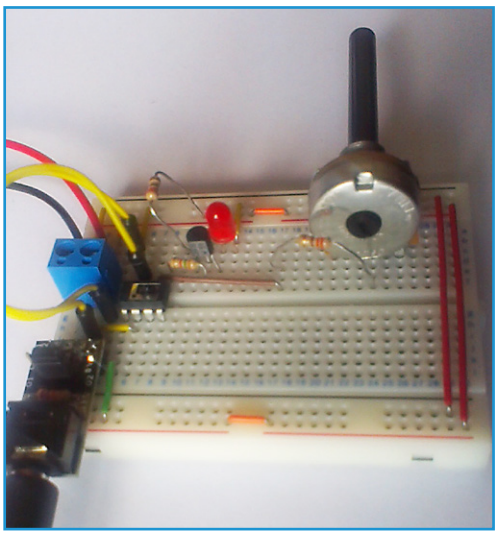

Figure 4.
PWM output breadboard setup.

very close to actual analog output. The PWM output pin will continuously output a square wave signal, and in contrast to normal PICAXE commands, the PWM output command runs in the background (in parallel to the rest of program execution). A PWM signal can be defined by frequency and duty cycle, as shown in **Figure 5**. Frequency is the number of repeating cycles per unit time (usually seconds); the period is the duration of one cycle and the duty cycle is the percent spent in an active state (High). Though some software like the RPi.GPIO Python library for software PWM on a Raspberry PI [5] would allow the user to set the PWM frequency and duty cycle directly, on PICAXE, setting up PWM is more challenging. Multiple commands exist, but the newest and most relevant one is "`pwmout`". Because this command is not that simple, we will base on the `pwmout` description available in [2], instead of trying to explain everything from scratch. The frequency of the PWM signal is based on the frequency of the PICAXE clock. Most PICAXE chips support multiple clock speeds, but the `pwmout` command only works with a limited number of frequencies. Commonly, the default clock frequency is 4 MHz, which will work with `pwmout`. The most simple uses for the `pwmout` command are '`pwmout pin,period,duty cycles`' to turn the signal on and '`pwmout pin,OFF`' to turn it off again. The variables `period` and `duty cycles` are used to set the PWM frequency (f_{PWM}), period (T_{PWM}) and duty cycle (D_{PWM}) respectively, through the following relations:

$$\frac{1}{f_{PWM}} = T_{PWM} = (period + 1) \times 4 \times resonator\ speed$$

$$D_{PWM} = duty\ cycles \times resonator\ speed$$

Where `resonator speed` is 1/4000000 for a 4 MHz clock.

Note the `pwmout` command is using a time period instead of a percentage to specify the duty cycle! Luckily, the PWMout wizard will ease things out. A number of wizards are present in the (LinAxePad) software, the one for PWM may be found under PICAXE ➞ Wizards ➞ PWMout. The wizard allows us to specify the PWM frequency in Hz, and the duty cycle as a percentage (the way we are used to), and it outputs a `pwmout` command complete with

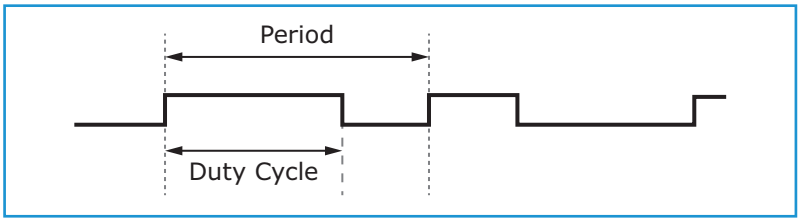

Figure 5.
Structure of a PWM signal.

variables to produce a PWM signal, approximating your specifications (limited by the `pwmout` command resolution).

PWM is useful for dimming LEDs or controlling DC motor speed. A LED does not obey Ohms law (see previous article). To control its brightness without using a current source (a power supply that varies its output voltage to deliver a constant output current), PWM can be used to rapidly switch the LED on and off. With a duty cycle of 50%, the LED is only switched on for 50% of the time. The perceived LED brightness changes accordingly. Because the frequency of the PWM signal is very high, only the average result of the switching can be seen by human eyes.

The PWM frequency to use for dimming an LED should be high enough to be visually imperceptible. Consider the output resolution though—selecting a frequency closer to the PICAXE maximum frequency will reduce the number of bits available to set the duty cycle.

This example will use a LED to show the principle behind controlling output intensity through PWM, but the same goes for controlling the speed of a DC motor. The PICAXE cannot source enough current for a motor, so a transistor should be used to operate the motor, as mentioned in the previous PICAXE article (Elektor**.**POST Project No. 16) [1]. There we also mentioned the transistor switching speed: this becomes something to keep in mind when using PWM. If the transistor cannot switch fast enough to keep up with the PWM frequency, the PWM output signal will become distorted and therefore unusable. Lowering the PWM frequency or selecting a faster transistor will solve this problem. Note that Darlington pairs are even slower than single transistors, as explained also in the previous installment.

The circuit will be identical to the previous example in Figures 2 and 3 (for input and

output respectively), but this time the software is a bit more complicated, as shown in **Listing 2**. The calculations performed fit the range of input values (0-255) to the range of output values (0-100). The next part will make use of this principle.
The new program uses the 8-bit ADC input signal from the pot's wiper position to set the duty cycle of a PWM output signal, toggling the LED. You are now able to control the LED brightness with the potentiometer!

Let's move those servos

A servomotor is a mechanical actuator meant for precise control of position, among others. A typical servo is connected via three wires, black for ground, red for supply voltage, and white (usually) for data. A servo will try to assume a position specified by a signal through the data pin. If the signal is lost, it will no longer hold its position. To set the position of a servo, it expects a square wave carrier signal with a period of 20 ms (frequency = 1 / 0.020 = 50 Hz); the duty cycle (periods of 0.75 ms to 2.25 ms) determines the angle. We don't have to use the `pwmout` command for this, as PICAXE comes with two other commands. The first command, '`servo pin,pulse`' initializes a PWM-capable output as a servo output. Next, the '`servopos pin,pulse`' command is used to change the servo position.
When connecting the servo, the data line is connected to the PICAXE servo (PWM capable) output pin via a current limiting resistor (for example 330 Ω), but the servo needs a separate power supply for a number of reasons. A servo can draw large currents and cause a lot of noise on the power line. In practice, this means the PICAXE will behave erratically if the power supplies are not separated! Also, some servos may require a higher voltage than the rest of your electronics. The 0 V of the PICAXE power supply and the servo power supply should be interconnected to provide a common reference rail. A capacitor is connected between servo V+ and V0 to reduce ripple. For the servo power supply in the example, four 1.5-V 'AA' dry cells have been used (not rechargeables, they have a lower nominal voltages of about 1.2 V) for a total of 6 V.
The values for the variable `pulse` are expected to be within the 75 to 225 range, corresponding to the periods of 0.75 to 2.25 ms the servo kind of expects.
An example circuit on how to connect a servo is provided in the schematic of **Figure 6**, and once again, the analog input circuit

Listing 2: PWM Out

```
init:
low 2
main:
do
  readadc 1,b1  ;get analog value
  if b1 > 250 then  ;clamp value
    b1 = 250
  endif
  w1=b1*10   ;w1 is a 16-bit register (b2 with b3)
  w1=w1/25   ;needed for integer arithmetic
  pwmout 2,24,b2;b2 is the significant part of w1
loop
```

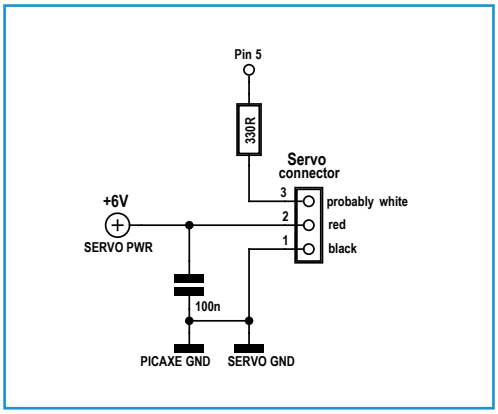

Figure 6.
Servomotor schematic.

Listing 3: Servo control

```
init:
servo 2,100   ;init servo
main:
do
  readadc 1,b1  ;get analog input
  if b1 > 250 then  ;clamp value
    b1 = 250
  endif
  w1=b1*3  ;scale value
  w1=w1/5
  b2=b2+75
  servopos 2,b2 ;update servo position
loop
```

remains the same of the previous examples (Figure 2). The code to control the servo through the pot connected in the analog input is found in **Listing 3**. The example setup could look like the one shown in **Figure 7**. Because the PICAXE registers are 8-bit or 16-bit integers, we have to work around that limitation to make the ADC input (0-255) correspond to the output range (0-100). In this case the range of the output values is 75-225, which means the range to scale to the analog input is 225 – 75 = 150. Because 150 / 250 = 3 / 5, we can get the scaled value by multiplying the ADC value by 3 (this requires 16-bit resolution because 3 × 255 does not fit within the maximum 8-bit value of 255), followed by a division by 5. Addition of the minimum value of 75 gives a scaled output value `b1` within the correct range to position the servo, according to the value in the wiper terminal of the potentiometer. Note that `b2` is used in the code to represent the lowest 8 bits of w1 because w1 is a 16-bit register made up from the two 8-bit registers b2 and b3, as described on the previous part [1].

A precise control of the angle is very useful in robotics, but such projects would probably require many more servos than the PWM outputs available on a single PICAXE chip. A solution to this problem would be including dedicated servo driver chips. The PICAXE could talk to these peripherals, and the servo driver board would remember and set positions for more than 20 servos per board. For instance, the AXE031 allows to control up to 21 channels [3]. In the next article we will focus on how to make the PICAXE "talk" to peripherals.

The Singing PICAXE: generating melodies and sounds

Another built-in PWM-based PICAXE functionality is the '`tune`' command. The command syntax is different for some versions of the chip, because of the number of (PWM) outputs. We will use the version suitable for 8-pin PICAXE chips. The `tune` command is used to play a monophonic tune (or monotone) made of speaker bleeps, similar to old-fashioned mobile phone ringtones. For some reason, a functionality to switch other outputs (intended to make LEDs flash to match the tune) is integrated in the command too. Obviously, to use `tune` we will need a PWM-capable output, and in the case of the PICAXE 08M2 chip there is only one. The command '`play`' is also available, which plays one of the pre-programmed tunes.

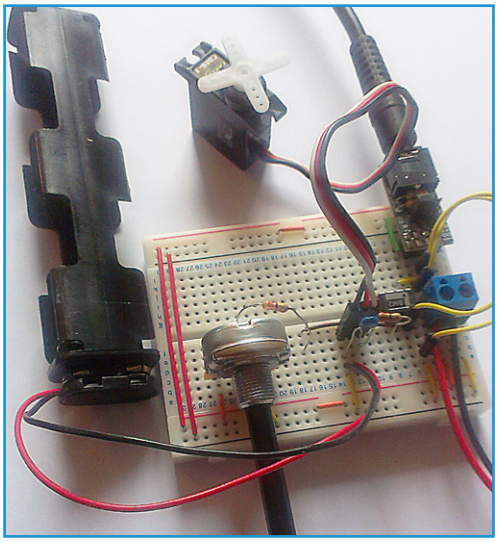

Figure 7. Servomotor breadboard setup.

The PICAXE manual part 2 [6] includes extensive documentation regarding the `tune` command, as well as sample circuits to use it with a piezoelectric buzzer, a speaker, or an external audio amplifier. For the `tune` example, the author used a pair of recycled old headphones. The circuit in the manual to connect a speaker recommends to use a 40-80 Ω speaker. If you measure a lower speaker resistance, then a series resistor must be added in order to get at least 40 Ω in total.

Another wizard is available (under PICAXE ➞ Wizards ➞ Ring Tone Tunes) to aid in the creation of melodies with the `tune` command (and import ring tones in the RTTTL format, for instance from this site [7]). Even though this command allows us to create tunes easy and fast, we can also use the `pwmout` command to make tunes manually, or, to create sound effects similar to the ones used in old Atari games. Hint: change the PWM wave parameters rapidly in for-loops with varying STEP parameters. You can also nest different for-loops.

The circuit to connect a speaker (from the PICAXE manual part 2 in [6]) is presented in **Figure 8**, and the code for a single tune

may be found in **Listing 4**. It is important to introduce this code exactly, there should only be one line of code for it to work. The experimental setup is shown in **Figure 9**. The listing uses a simple demo melody (do you recognize it?), but feel free to experiment with the `tune` (or `pwmout`) command to create new tunes and sounds. Note this example does not require the input circuit, but don't remove it yet, we will need it again for the next part.

This PWM output wants to be analog

PICAXE only integrates digital output pins. However, it is still possible to output an analog voltage using PWM with a low-pass filter. In the previous parts [1], we also covered the principle of the voltage divider. When a potential is applied to a number of resistors in series, then the voltage over the individual resistors is divided proportionally to their respective resistances. A PWM (square) wave through a low-pass filter effectively works the same way. The second resistor in the voltage divider is replaced by a capacitor, so the voltage over the capacitor will be proportional to the duty cycle of the PWM signal. However, this only works if the frequency of the PWM is significantly larger than the cutoff frequency of the low-pass filter. The formula to calculate the cutoff frequency is:

$$f_C = \frac{1}{2\pi\ RC}$$

A bigger RC value results in a smoother DC output voltage, but it is also slower: If the PWM frequency changes, the analog output voltage takes more time to change accordingly. To select components, first we may calculate the resistor value to limit the current, then choose a value for C using the high-pass filter formula

$$C = \frac{factor}{2\pi\ R\ f}$$

where "factor" acts as a multiplier to move the cutoff frequency of the filter away from the PWM frequency. To visualize the effects of the PWM frequency and RC value, a website [8] provides a simulation that shows how the shape of the analog output changes with different wave and circuit parameters. When calculating the RC value it is advisable to use at least a factor of 100 to get an output signal with relatively minor ripple.

We will not use the highest PWM frequency available, because the PICAXE loses its finer control over the duty cycle at higher PWM frequencies. Instead, to control the duty cycle in 100 steps, a frequency of 4 MHz / 100 = 40 KHz. The parameter `duty cycles` passed to the `pwmout` command is now conveniently the same as the duty cycle percentage! With a PWM frequency of 40 KHz and a value of 10 KΩ for R (to limit current as explained),

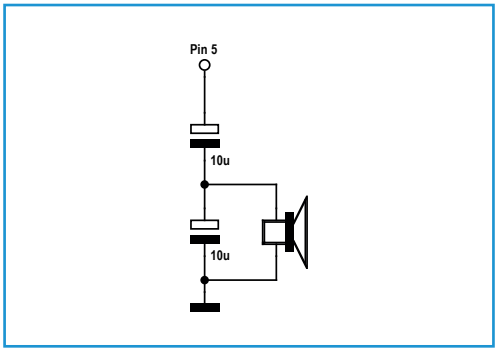

Figure 8.
Tune generating schematic.

Listing 4: Generating a Tune (one line)

```
tune 1,4,($67,$69,$40,$69,$04,$4C,$04,$22,$4C,$67,$69,$40
,$69,$02,$4C,$02,$00,$4C,$6B,$29,$67,$69,$40,$69,$C0,$02,
$2B,$29,$27,$27,$C2,$E0,$67,$69,$40,$69,$04,$4C,$04,$22,$
4C,$67,$69,$40,$69,$C7,$2B,$20,$6B,$29,$67,$69,$40,$69,$C
0,$02,$2B,$29)
```

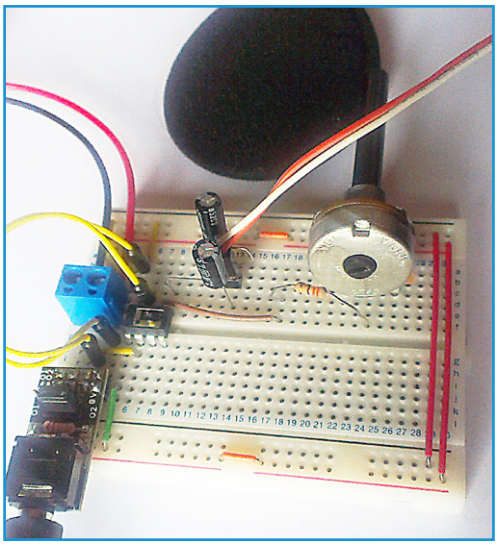

Figure 9.
Speaker breadboard setup (using recycled headphones).

$$C = \frac{100}{2\pi \times 10000 \times 40000} =$$

$$3.98 \times 10^{-8} = 39.8nF$$

The closest standardized capacitor value would be 47 nF. In this case, the PWM signal will not be changed, so a (much) larger C value could be used to get a smoother output without side effects.

This output circuit is shown in **Figure 10**, while the input circuit stays the same (Figure 2). We will use the same code as in Listing 2 because the PWM frequency is already 40 KHz, and the correct scaling code is also included. The breadboard setup can be seen in in **Figure 11**.

The analog input is used to control the analog output voltage via the PWM duty cycle. The smoothness of the output signal can be visualized using an oscilloscope, or, if the output is smooth enough, a standard voltmeter. It would be possible to connect the output directly to the analog input of the PICAXE, to measure the output voltage using the PICAXE ADC, but at this point we have no way to display the results yet.

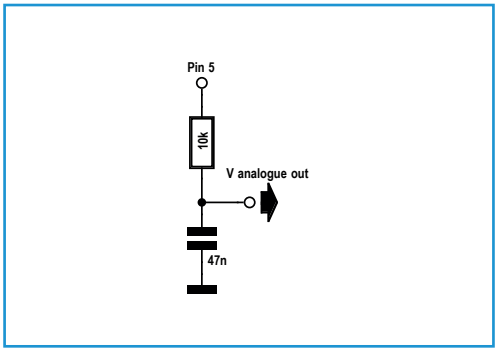

Figure 10.
Analog output schematic.

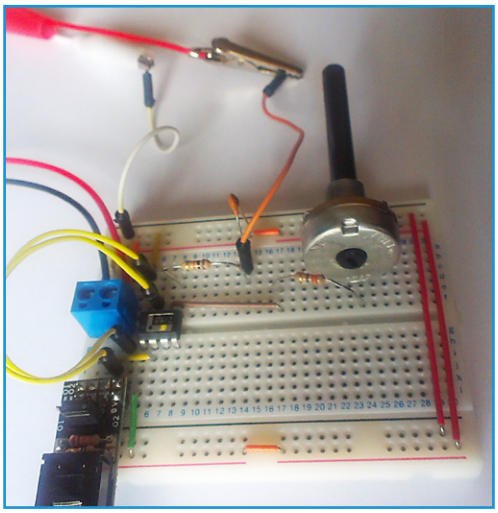

Figure 11.
Analog output breadboard setup.

Summary

In addition to programming a PICAXE chip and controlling simple electronics, we can now use analog inputs, and we have gained a more refined control over our outputs. We can now use a PWM output to generate an analog output signal, design a dimmed output, and even generate old-fashioned sounds and control the position of a servo accurately. However, in the last example we encountered a problem: We cannot display any data yet, for instance as part of an interface to more complex PICAXE programs, or to show data collected from the (analog) input. But don't panic! In the next installment brought to you by Elektor**.**POST we will show how to use the PICAXE function to communicate with different peripherals, greatly increasing the possibilities of a PICAXE-based project, including an OLED character display.

(130262)

Internet Links

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

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

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

[4] www.picaxeforum.co.uk/forum.php

[5] https://code.google.com/p/raspberry-gpio-python/wiki/PWM

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

[7] www.picaxe.com/RTTTL-Ringtones-for-Tune-Command/

[8] http://sim.okawa-denshi.jp/en/PWMtool.php