

Author: Jeremy Leach Email: ukc802139700@btconnect.com

September 2006

Introduction

This article describes an experimental RC Timer to use with a PICAXE. The principle is very simple: Start a capacitor charging and then use the charge voltage to calculate the elapsed time it's been charging.

The method used fits into around 90 bytes of code. It's been implemented on a PICAXE08M, which allows use of only a single pin, however it could be implemented as a two-pin solution on any PICAXE.

Tests so far indicate that the RC Timer is fairly accurate at room temperature - around 2% error. However the accuracy in different conditions will be limited by the chosen components and is not expected to be that good.

This little project has been done mainly for interest, and I'm not sure if there is a real use for this idea. However the concept is quite interesting because in effect the RC pair are acting as a self-contained timing unit, and the calculated times will be independent of the PICAXE clock speed.

The solution might also have some value from it's simplicity and low board space / cost.



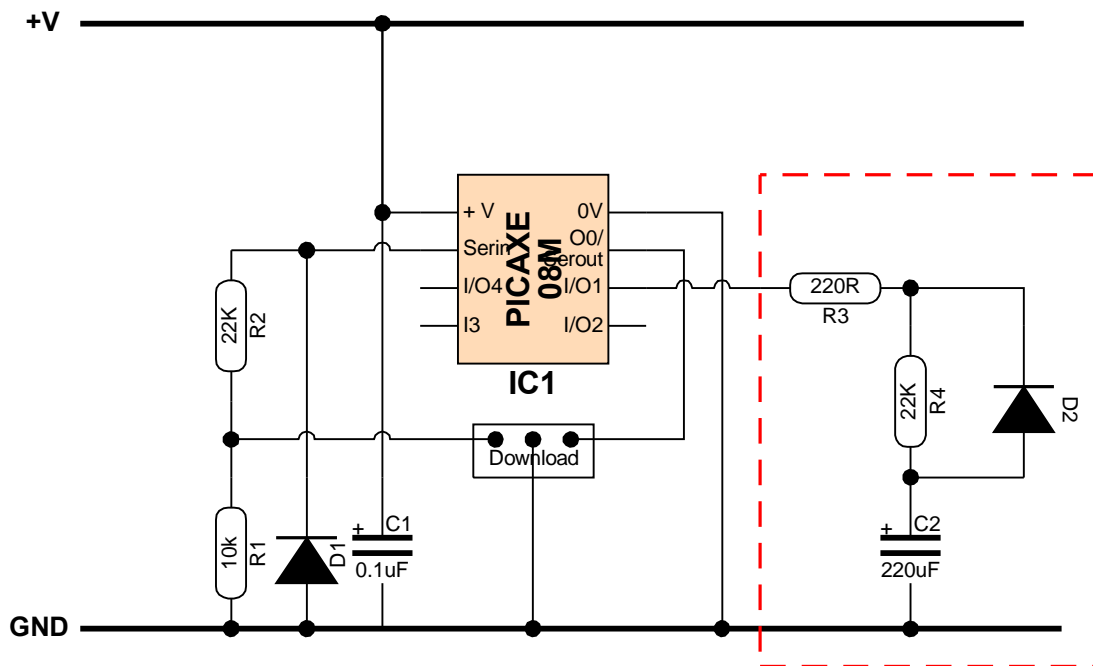
IN THIS ARTICLE

- 2 Principle of Operation
- 3 Implementing in Code
- 4 Results
- 4 Further Detail
- 5 Code listing

ACKNOWLEDGEMENTS

- PICAXE is a trademark of Revolution Education Ltd.
- PICAXE Forum <http://www.rev-ed.co.uk/picaxe/forum/>

Figure 1 : RC timer test circuit



Principle of operation

The test circuit I've bread-boarded is shown in Figure 1. The RC Timer element is within the red dotted line.

The Timer connects to pin I/O1, which is also an ADC input.

Operation of the Timer is as follows:

Timer Reset

To use the Timer, the PICAXE pin is first taken low, and held low for a set time to ensure the capacitor discharges.

Starting with a fully charged capacitor: as soon as the pin is taken low, diode D2 is forward biased and effectively shorts out resistor R4. The capacitor rapidly discharges through resistor R3. This is a low value resistor purely to limit the maximum discharge current to under 25mA, which is the maximum output current of the PICAXE pin.

When the capacitor voltage reaches the forward-drop voltage of the diode, R4 comes into play and so the discharge rate suddenly slows down. This isn't ideal and so it's a good idea to use a low forward-drop Schottky diode for D2.

The capacitor won't actually discharge to 0 voltage because the PICAXE output will be around 0.2V in the 'low' state. However this doesn't matter too much as long as it's consistent.

Once the capacitor is discharged the PICAXE pin is immediately taken high, and timing begins...

Timing

Diode D2 is now reverse-biased and so the capacitor charges via R4 and R3. The values of R4 and C2 are chosen to give the desired time-constant.

Note that while the capacitor is charging the PICAXE can get on and do whatever it needs to as long as the pin is kept high.

The theoretical charge formula for the voltage across the capacitor (V_c) using a 5-volt supply is:

$$V_c = 5(1 - e^{-T/(R3 + R4)C2})$$

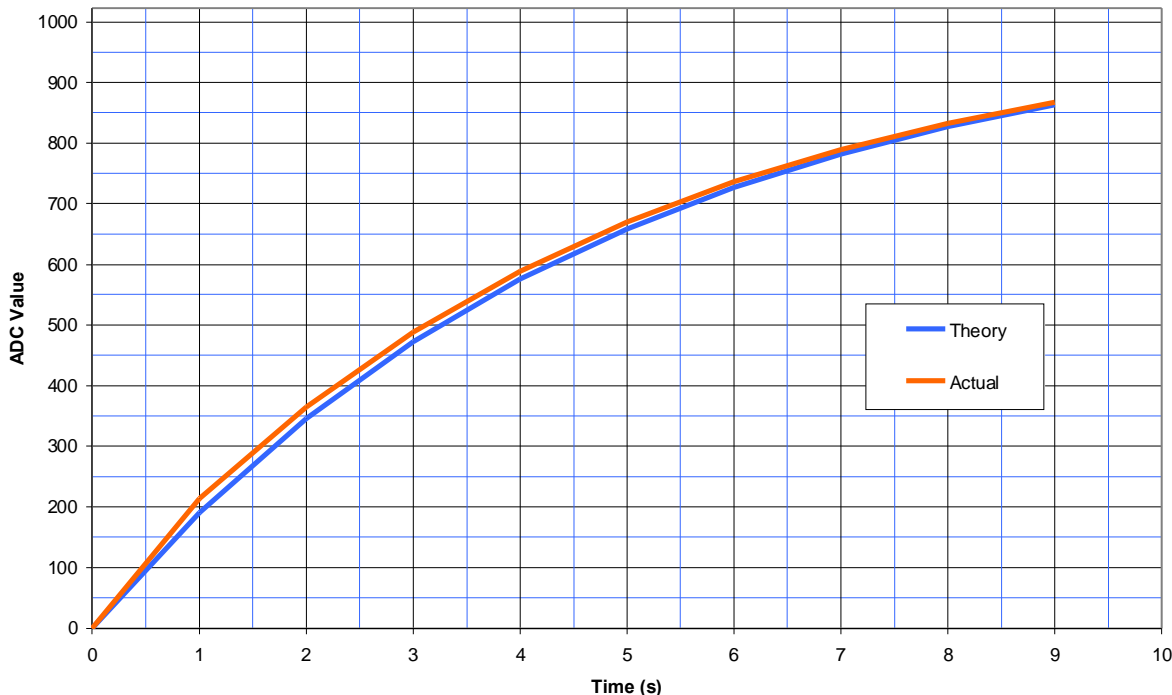
It's easy to measure the voltage using the `ReadADC10` command to check this charge formula.

My results are shown in Figure 2 and show that the actual charge graph is only slightly different from the theoretical graph for the components I've used.

Note that the graph refers to ADCValues and not

Figure 2: Charge Graph

Capacitor charge curve : Theory and Actual



voltages. The supply voltage is not relevant because everything can be calculated in terms of the ADC readings, which span from 0 to 1023 across the supply voltage range.

Reading the timer

To read 'the Timer' the capacitor voltage (ADC Value) is read using the `ReadADC10` command. However to do this, the PICAXE pin has to be momentarily made an input. While it's an input the charging stops, so if the timer is going to be used again, using the same charge cycle then the pin must be made high again as soon as possible, otherwise subsequent Timer results will be affected.

Note that the input impedance of the PICAXE pin is high while it's designated an input, so R3 and R4 have insignificant effect (as long as they aren't too high).

Once the ADC value is obtained (representing the capacitor voltage) it then needs to be translated to a time by effectively automatically reading off the graph.

Implementing in code

The whole aim is to implement the code routines in as least bytes as possible while still providing a useful Timer output.

The method I've used is to approximate the charge curve by a set of line segments, and find which line segment the ADCValue falls within and calculate the corresponding time value.

The line-segment information is stored in EEPROM. The line-segments span a set time interval of `SegmentTimeWidth`. Each segment spans an ADC Range of `RangeHeight` with limits of `RangeLow` and `RangeHigh`.

Instead of storing all this data, it's most efficient to only store `RangeHeightDiff` values. This is a series of byte values representing the difference in height of each successive line-segment range. From this all other values can be calculated.

When an ADCValue is read, the line segments are scanned to see which one applies to the ADC

Figure 3 : Detailed values

Line Seg	Start Time (s)	Theory Range Low	Theory Range High	Theory Range Height	Theory Range Height Diff	Actual Range Low	Actual Range High	Actual Range Height	Actual Range Height Diff
				255				255	
0	0	0	191	191	64	0	214	214	41
1	1.00	191	346	155	36	214	364	150	64
2	2.00	346	473	126	29	364	488	124	26
3	3.00	473	575	103	24	488	589	101	23
4	4.00	575	659	84	19	589	670	81	20
5	5.00	659	727	68	16	670	736	66	15
6	6.00	727	782	55	13	736	789	53	13
7	7.00	782	827	45	10	789	832	43	10
8	8.00	827	864	37	8	832	867	35	8
9	9.00	864	893	30	7	867	896	29	6

Value.

Results

Tests so far indicate a fair accuracy at room temperature (Error<2%). The accuracy will be affected by a number of factors including temperature and the age of the capacitor. However a thorough analysis is beyond the scope of this initial experiment!

Further Detail

Discharge calculations

Discharge to 0.3V:

$$0.3 = 5(1 - e^{-t/(0.00022 * 220)})$$

$$t/(0.00022 * 220) = 0.061875$$

$$t = 3 \text{ ms}$$

Discharge from 0.3V to 0.01V:

$$0.01 = 0.3(1 - e^{-t/(0.00022 * 22000)})$$

$$t/(0.00022 * 22000) = 0.034$$

$$t = 164 \text{ ms}$$

Detailed line-segment values

Figure 3 shows the values I worked out on a spreadsheet, based on the components shown in Figure 1.

The column on the right is the list of values used in the EEPROM statement. They all comfortably fit into byte values. The less line segments used, the higher these values will be, the initial values being the highest due to the rapid rate of change of the initial capacitor charge.



PICAXE RC Timer

```
*****
!*
!* PICAXE RC Timer
!*
!* Version: 1
!* Author: Jeremy Leach August 2006
!*
*****

*****
***** EEPROM *****
*****

'Lookup table for RangeHeightDiff.
Eeprom 0, (41,64,26,23,20,15,13,10,8,6)

*****
***** GLOBAL VARIABLES *****
*****
'Word0 (b0 and b1)
Symbol RangeHeight = b0
'Word1 (b2 and b3)
Symbol RangeLow = w1
'Word2 (b4 and b5)
Symbol RangeHigh = w2
'Word3 (b6 and b7)
Symbol ADCValue = w3
Symbol SegmentADCOffset = w3
'Word4 (b8 and b9)
Symbol Segment = b8
Symbol RangeHeightDiff = b9
'Word5 (b10 and b11)
Symbol LSW = w5
Symbol ResultWord = w5
Symbol Time_MS = w5
Symbol Time_MS_LSB = b10
Symbol Time_MS_MSB = b11
'Word6 (b12 and b13)
Symbol MSW = w6

*****
***** CONSTANTS *****
*****
'Pin assignments
Symbol RCPin = 1

'Other Constants
Symbol SegmentTimeWidth = 1000 'ms
Symbol Segments = 10

*****
***** MAIN *****
*****

'Demo of RC Timer
Symbol TestRun = b1
Symbol PauseTime = w5
Output 4
Output 2
```

PICAXE RC Timer

```
Demo_1:
  For TestRun = 1 To 10
    Gosub RESET_RC_TIMER
    PauseTime = TestRun * 1000 'milliseconds
    Pause PauseTime
    Gosub GET_TIME
    Sertxd ("Time = ",#Time_MS_MSB,":",#Time_MS_LSB," milliseconds",13,10)
  Next
End

'*****
'**** RC TIMER ROUTINES ****
'*****

'NOTE1: Modelled on: R = 22.22KOhm, C = 220uF. For best accuracy plot your own
'graph and use those values in the EEPROM table.

'NOTE2: The most efficient way to store the line segment data is to store the
'RangeHeightDifference values, because these can all be contained in byte values.

GET_TIME:
  'ON EXIT: Time_MS contains the time in milliseconds since the timer was
  'reset. If the time is greater than (Segments * SegmentTimeWidth) then
  'a value of 65535 will be returned.

  'Initialise
  Input RCPin
  Readadc10 RCPin,ADCValue
  High RCPin 'quickly return the pin to high, to carry on charging the cap.
  RangeHeight = - 1
  RangeLow = 0
  Segment = 0
  Time_MS = - 1

  CheckSegment:
  'Search for the line segment that contains the ADCValue.

  Read Segment,RangeHeightDiff 'Get the RangeHeightDiff value from EEPROM.
  RangeHeight = RangeHeight - RangeHeightDiff
  RangeHigh = RangeLow + RangeHeight

  If ADCValue <= RangeHigh Then CS_1
  'The ADC Value is greater than the upper range of the current line-
  'segment, so keep searching.
  Segment = Segment + 1
  If Segment < Segments Then CS_2
  'The time is too high, and the ADC Value is higher than the upper-
  'limit of the highest line-segment.
  Return
  CS_2:
  RangeLow = RangeHigh
  Goto CheckSegment

  CS_1:
  'The ADC Value lies on the current line segment.
  'Calculate ADCValue * SegmentTimeWidth / RangeHeight without overflow.

  SegmentADCOffset = ADCValue - RangeLow
  MSW = SegmentADCOffset ** SegmentTimeWidth
  LSW = SegmentADCOffset * SegmentTimeWidth
  ResultWord = LSW / RangeHeight
  ResultWord = - 1 / RangeHeight * MSW + ResultWord
```

PICAXE RC Timer

```
'ResultWord = MSW / RangeHeight + ResultWord. Not necessary because in  
'this case MSW will always be < RangeHeight.  
Time_MS = Segment * SegmentTimeWidth + ResultWord  
Return
```

RESET_RC_TIMER:

```
Low RCPin 'Discharge cap rapidly through diode.  
Pause 200 'To make sure it's fully discharged  
High RCPin 'Start charging the cap.  
Return
```