599 Menlo Drive, Suite 100
Rocklin, California 95765, USA
Office: (916) 624-8333
Fax: (916) 624-8003

Sales:sales@parallax.com
Technical: support@parallax.com
Web Site: www.parallax.com

PE Kit Tools: Measure Resistance and Capacitance

Propeller microcontroller applications that need to measure resistors or capacitors can use the RC Time object and a resistor capacitor (RC) circuit. Since there's a myriad of resistive and capacitive sensors that respond to physical properties such as light, rotation, humidity and force (to name a few), the simple, inexpensive circuits and the easy-to-use RC Time object featured in this PE Kit Tools article open up a world of measurement possibilities.

Web version: PE Kit Tools: Measure Resistance and Capacitance
Full PDF & source code: PE Kit Tools - Measure Resistance and Capacitance.zip
More info: PE Kit Labs, Tools, and Applications
Platform: Propeller Education Kit

In this Tool Chapter:

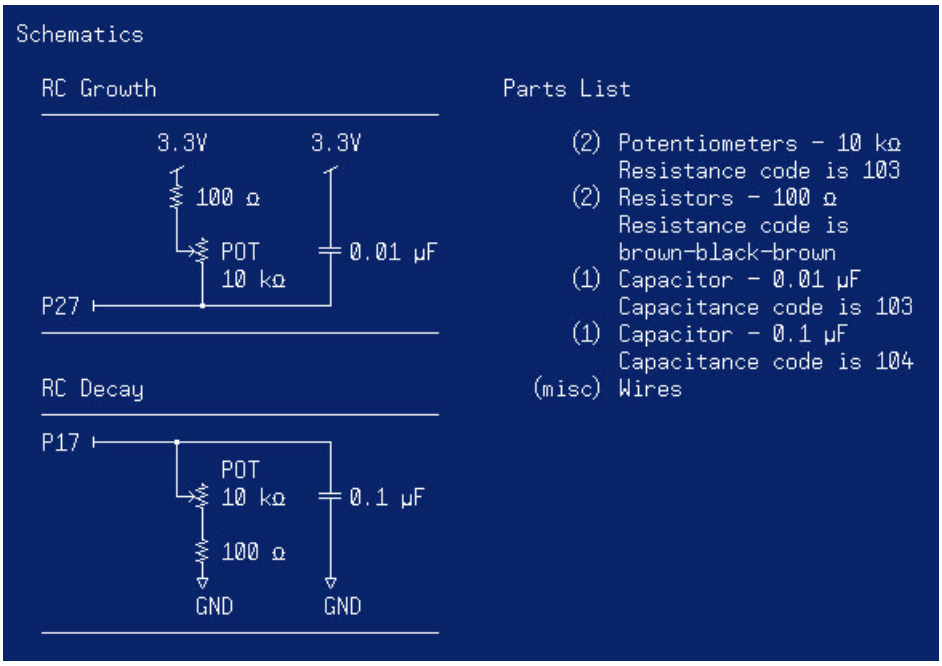
- RC Time Parts and Circuit
- How RC Time Measurements Work
- Simple Test Code
- RC Time object features
  - Timeout Setting
  - Charge Time Setting
  - Sequential vs. parallel measurements
  - Establishing Sampling Rates
- More PE Kit Sensor Examples
  - Ambient and Infrared Phototransistors
  - Direct Sunlight with LEDs
  - Proximate Flame with an Infrared LED
- Application Example: RC Resistance Meter
- Application Example: 200 kHz Sampling Rate

RC Time Parts and Circuit

One of the most common variable resistance sensors is the potentiometer, a.k.a "pot". As the knob on the pot is turned, its resistance varies. The Propeller microcontroller can use the RC Time object to measure the variable resistors (labeled POT) in Figure 1, which can in turn give the application accurate information about how far each potentiometer knob has been turned. The potentiometer can also be replaced by any number of other resistive sensors. For example, if the pot is replaced with a photoresistor, the circuit can instead be used to measure light intensity. If the pot is replaced with a fixed resistor, variable capacitor sensors that measure pressure or humidity can be measured. The examples in this chapter will use a couple of potentiometers to explore the RC Time object's features, and then demonstrate how other PE Kit parts like phototransistors and LEDs can be used with RC Time to measure a variety of physical properties.

✓ Build the circuits shown in Figure 1.

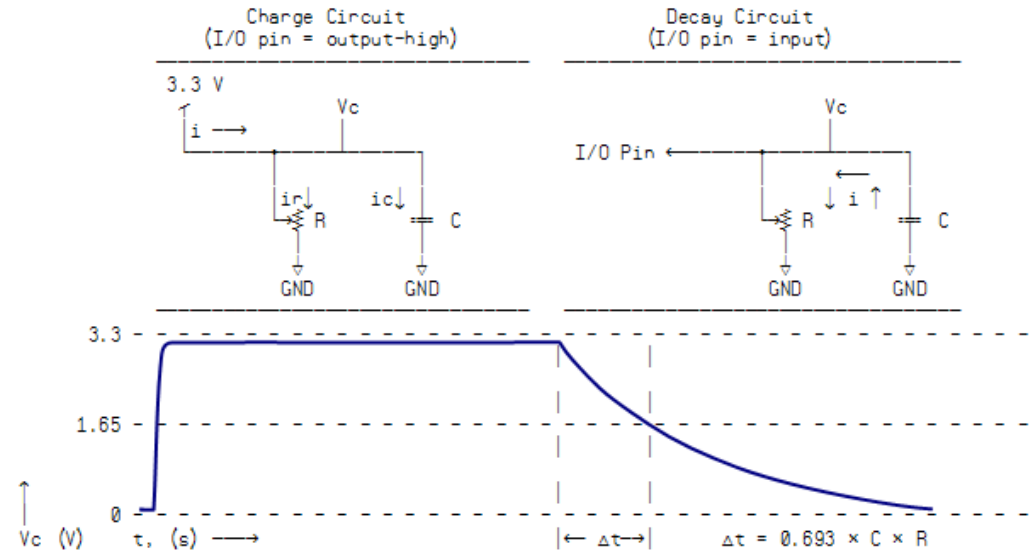
Figure 1: RC Test Circuits and Parts List



How RC Time Measurements Work

The RC Time object can be used to determine a variable resistor value by treating the capacitor in the circuit like a small battery. It charges up this capacitor (left side of Figure 2) by sending an output-high signal to the I/O pin. Then, it changes the pin to input and measures the time it takes the capacitor's voltage to decay as it looses its charge through the variable resistor (right side of Figure 2). The decay measurement time (Δt) starts at 3.3 V, and stops when the voltage to decays below the Propeller I/O pin's 1.65 V logic threshold. For larger resistances, it takes more time for the capacitor to lose its charge. For smaller the resistances, it takes less time for the capacitor to lose its charge.

Figure 2: Microcontroller RC Decay Measurement(1)



(1) Excerpt from *Propeller Education Kit Labs: Fundamentals*

The equation that describes the time it takes for the voltage to decay from 3.3 to 1.65 V is:

$$\Delta t = 0.693 \times C \times R$$

With a little algebra, the terms can be rearranged to solve for the value of R. This is great if the project is to make a simple resistance meter. On the other hand, if the application needs a sensor measurement, it may just scale the time measurement and compare it to some benchmark values. Other sensor applications need to compare the sensor measurement to complex equations, and others still use points from a graph in the sensors datasheet. The application can then check to find out which value in the list is closest to the measured value and so determine the value of the property the sensor measures.

Use the extra 100 Ω series resistor with the pot!

Figure 1 shows a 100 Ω series resistor in series with the potentiometer, but it is missing from Figure 2 for the sake of simplicity. When the potentiometer in the PE Kit is turned all the way in one direction, its resistance is 0 Ω. Without the 100 Ω resistor, what you have on the left side of Figure 2 is a short circuit to ground during the period of time that the Propeller I/O pin is charges the capacitor by applying 3.3 V. While this brief short circuit should not damage the Propeller chip or its I/O pin, it could cause a brief dip in the supply voltage, which could potentially cause problems with other circuits in an application. For example, if another cog is using that supply voltage as an A/D conversion voltage reference, the supply voltage dip might throw off the measurements.

Not all resistive elements can create short circuit's to ground. If you replace the potentiometer with a resistive element that never drops below 100 Ω, the extra resistor in Figure 1 is not necessary.

Although a series resistor between the I/O pin would provide the same protection, the math to provide a correction factor using the voltage divider equation is more complicated and would take more program memory. The 100 Ω resistor simply adds a constant offset to any measurement, but the relationship between R (or C) and decay time remains directly proportional (y = mx + b). At the time of this writing, the affect of initial current to the capacitor when the I/O pin switches high is not known. It might also affect circuits that are referencing the Propeller chip's supply voltage.

Simple Test Code

The RC Time object has lots of tools for measuring RC voltage growth and decay in circuits. In its simplest form, code that measures the circuits in Figure 1 resembles PBASIC RCTIME commands for the Parallax BASIC Stamp microcontroller. After declaring the RC Time object, the code passes the pin, voltage state of the circuit at the start of the measurement, and the address of the variable where the Time method should store the result. The RC Time object uses these parameters to charge the circuit, measure the growth/decay, and store the results in the appointed variables: tGrowth in the first method call, and tDecay in the second.

```
'Test Simple RCTIME.spin
' ...
OBJ
  rc : "RC Time"
  ' ...
PUB Go | tGrowth, tDecay
  rc.time(27, 0, @tGrowth)
  rc.time(17, 1, @tDecay)
  ' ...
```

The "PE Kit Tools – Measure Resistance and Capacitance.zip" file has both Spin and ASM versions of the RC Time object along with several test code examples. The first code example to try is "Test Simple RCTIME.spin". This object use the PST Debug LITE object to display the measurements in the Parallax Serial Terminal. For a primer on how to use this object to display variable names and their values, see Debug LITE for the Parallax Serial Terminal topic.

- ✓ Download and unzip "PE Kit Tools – Measure Resistance or Capacitance.zip".

- ✓ Open "Test Simple RCTIME.spin" with the Propeller Tool software.
- ✓ Open the Parallax Serial Terminal, and set the COM Port to the Propeller chip's programming port. (You can use F7 in the Propeller Tool to find out which port that is.)
- ✓ Set the Parallax Serial Terminal's Baud Rate to 115200.
- ✓ In the Propeller Tool, load the Test Simple RCTIME object into the Propeller chip with F11
- ✓ Wait just long enough for the Propeller Tool software's Communication window to report "Loading…" before clicking the Parallax Serial Terminal's Enable button. The Parallax Serial Terminal will wait for the Propeller Tool software to finish loading code into the Propeller chip before it connects to the COM port.

Figure 3 shows an example of how "Test Simple RCTIME.spin" displays decay and growth time measurements in the Parallax Serial Terminal. These growth and decay times are in terms of 12.5 ns units. That's because this program has the Propeller chip's system clock set to 80 MHz, and if the clock is ticking at 80 million times per second, the time between each tick is 12.5 ns.

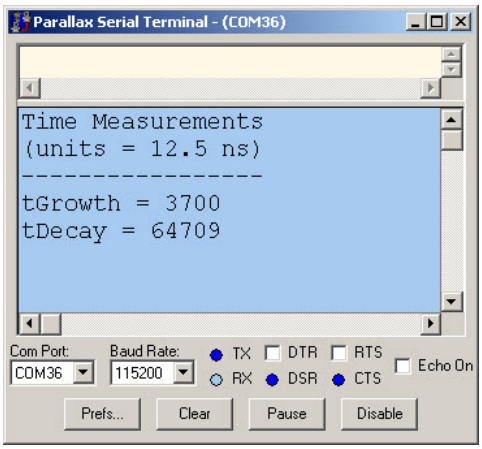


Figure 3: RC Decay and Growth Measurements

The range of decay times should be about ten times the range of growth times since the decay circuit has a capacitor that's ten times as large as the one in the growth circuit. Since the decay circuit's capacitor can store ten times the charge, the voltage will take ten times as long to decay through the same size resistor. This can be verified by setting the potentiometers to roughly the same position. The tDecay value should be about ten times the tGrowth value. There will be some variation, especially since the threshold voltage is not likely to be exactly 1.65 V. For example, if the threshold voltage instead 1.63 V, the decay time will be longer and the growth time will be shorter. Reason being, the decay will have to drop from 3.3 V down to 1.63 V, which is a 1.67 V decay. Meanwhile, measuring the growth will only be a 1.1.63 V voltage rise, and a different growth time measurement.

!	**Since threshold voltage can vary** between I/O pins and Propeller chips, some calibration may be necessary if you use this technique to make a resistance or capacitance meter. A resistance meter application example that demonstrates calibration is included later in this chapter.

RC Time Object Features

The RC Time object has a number of built-in features to make measurements simple and easy. For example, it has a built-in timeout that stops waiting for the decay (or growth) to cross the Propeller I/O pin's logic threshold after 0.01 seconds. It also has a charge time of 0.1 ms before it starts the measurement. The RC Time object has methods that allow you to adjust these values.

The RC Time object also defaults to taking measurements sequentially. In this mode, the cog that calls the object's Time method has to wait for the measurement to complete. There is also a method for configuring the RC Time object to take its measurements in parallel, without halting execution in the cog that called its Time method. Additionally, the RC Time object has Start and Stop methods for taking repeated measurements and storing the latest measurements in certain variables and specified intervals.

Timeout Setting

Let's say a circuit will decay under most circumstances, but not always. What happens then? Some objects that measure RC decay will wait indefinitely. In contrast, the RC Time object has a configurable timeout that defaults to 10 ms to prevent this problem. This configurable timeout also prevents the application from having to wait an unnecessarily long time for circuits that are responding slowly due to a large resistance or capacitance value. In many cases, a slow RC response indicates a throw-away measurement anyhow. A photoresistor in complete darkness is an example. It can really slow down an RC measurement due to large resistance values, but maybe the application only cares that it's beyond a certain level of darkness. At that point, the application might chose to hibernate until morning, or maybe turn on the lights!

Both the RC Time and RC Time.ASM objects have default timeout values of 10 ms. For RC Time.ASM, the timeout value is just as accurate as the decay measurement itself, good to the nearest clock tick. This is possible because RC Time.ASM uses assembly language to take the growth and decay measurements; whereas, RC Time takes its decay measurements in Spin. Since Spin is an interpreted language, it does not provide the same degree of control over timing that assembly language does. Keep in mind that both objects' growth and decay measurements are good to the nearest clock tick. The difference between the two objects is that the Spin language RC Time object's timeout value is approximate; whereas, assembly language RC Time.ASM's timeout value is exact.

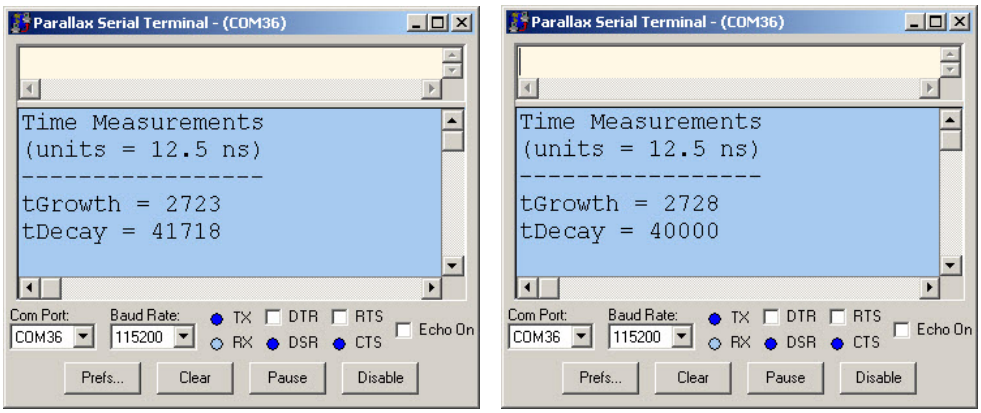
Changing either object's timeout value involves a simple method call to its TimeOut method. The examples below impose a 0.5 ms timeout on RC measurements for the Figure 1 circuits by passing clkfreq/2000 to the RC Time object's TimeOut method. This value is the number of clock ticks in half a millisecond, because clkfreq is the number of clock ticks in one second, and dividing 2000 into this value gives us the number of clock ticks in half a millisecond. It's equivalent to (clkfrreq/1000)/2.

```
OBJ
  rc : "RC Time"

PUB Go | tGrowth, tDecay
  rc.TimeOut(clkfreq/2000)
  repeat
    rc.Time(27, 0, @tGrowth)
    rc.Time(17, 1, @tDecay)
    '...
```

```
OBJ
    rc : "RC Time.ASM"

  PUB Go | tGrowth, tDecay
    rc.TimeOut(clkfreq/2000)
    repeat
      rc.Time(27, 0, @tGrowth)
      rc.Time(17, 1, @tDecay)
      '...
```

Figure 4 shows how the Spin RC Time object's tDecay timeout is approximate while the assembly language version is accurate to the clock tick. The code above configures both versions of the object to time out at 40_000 clock ticks (0.5 ms at 80 MHz). Keep in mind that for most applications, the actual decay measurement is the important part, and that both objects will return the exact same value, provided when the measurement is below the timeout.

Figure 4: Timeout Display for Spin (left) and Assembly Language (right)



- ✓ Try both "Test RCTIME Timeout.spin" and "Test RCTIME Timeout.ASM.spin"objects, and verify that they take the same measurements when the timeout values are below the 0.5 ms threshold.
- ✓ Set the potentiometer connected to P17 so that it causes each object to time out and verify that the assembly language version of the object can enforce the timeout more precisely.

Charge Time Setting

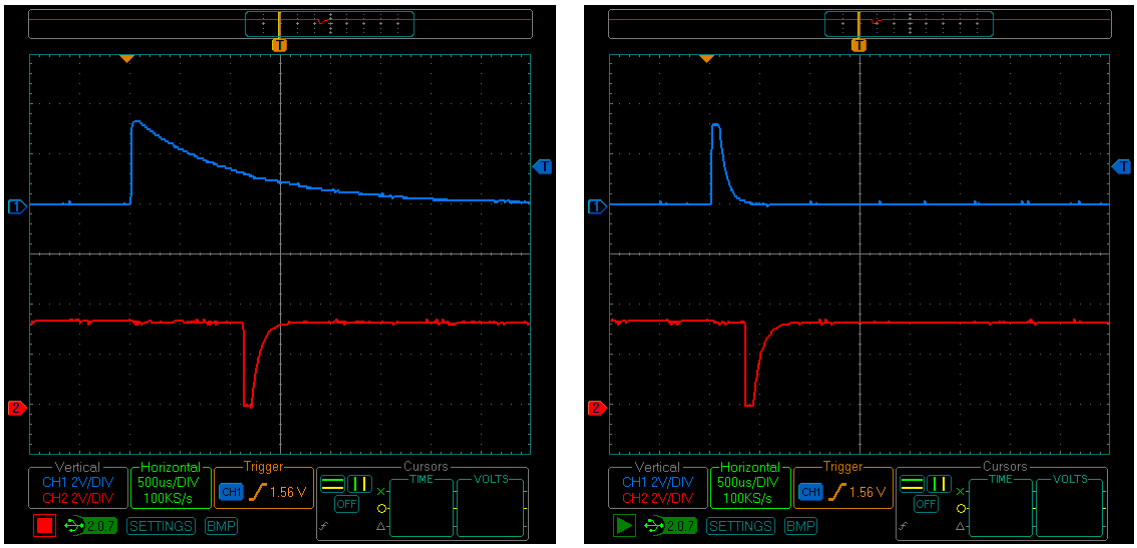
The RC Time object's default charge time is 0.1 ms, which works fine for the small RC values we are using in this chapter. However, larger capacitors may take longer to charge. To adjust the charge time, simply pass the object's ChargeTime method the number of clock ticks to wait and charge. For example, if you want to change the charge time from the default 0.1 ms to 5 ms, use this method call:

```
Rc.ChargeTime(clkfreq/200)
```

Sequential vs. Parallel RC Measurements

The RC Time objects default to sequential measurements. In sequential mode, the RC Time object does not return from the Time method call until either the measurement is complete or timeout is reached. Figure 5 shows what happens when the RC Time object's Time method gets called twice in immediate succession like it does in "Test Simple RCTIME.spin". On the left, the first measurement takes just over 1 ms to complete, and the second measurement does not start until the first measurement finishes. On the right, the first measurement takes less than 250 µs to complete, so the application moves on to the second measurement more quickly.

Figure 5: Sequential Measurements



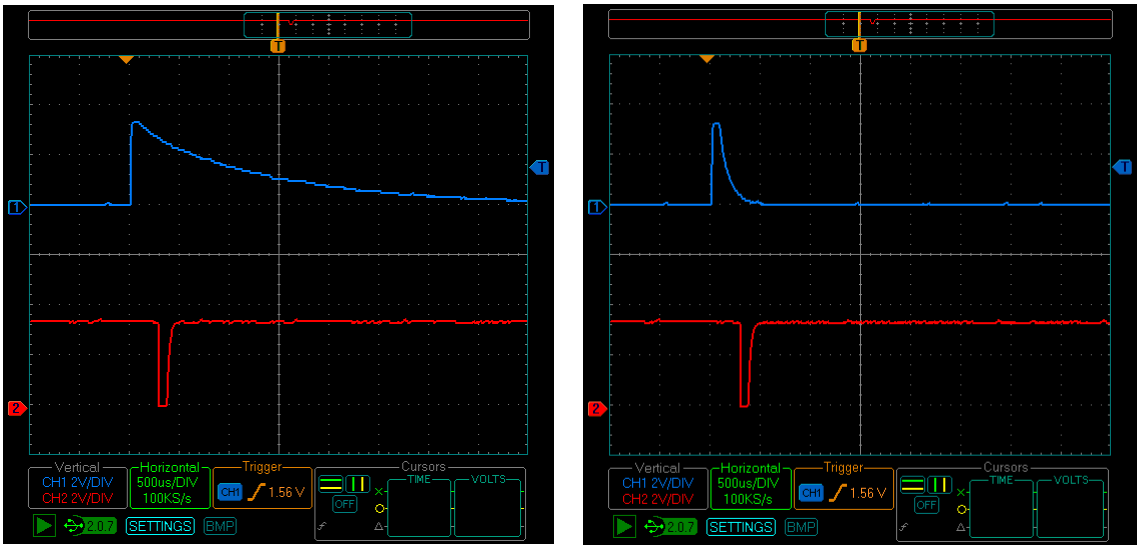
The RC Time object can be configured to launch multiple measurements in parallel. For parallel measurements, a copy of the RC Time object has to be created for each simultaneous measurement, and then each copy of the object has to be configured to return immediately after starting its RC measurement.

Here is an excerpt from "Test Parallel RCTIME.spin". It declares two copies of the RC Time object and then configures each copy for taking parallel measurements by passing rc#PARALLEL to its SetMode method.

```
OBJ
  rc[2] : "RC Time"

PUB Go | tGrowth, tdecay, i
  repeat i from 0 to 1
    rc[i].SetMode(rc#PARALLEL)
  '...
  rc[1].Time(27, 0, @tGrowth)
  rc[0].Time(17, 1, @tdecay)
  'Code here can work on other tasks while
  'waiting for the measurements complete...
```

With two copies of the RC Time object configured to take parallel measurements, Figure 6 shows how the second measurement starts immediately after the first one does, so the duration of the first measurement no longer affects when the second measurement starts. This approach can be useful for making sure measurements start at approximately the same time, and it can also be useful for saving time by taking multiple measurements in parallel. The drawback is that each measurement launches a separate cog for the duration of the measurement. This drawback is not severe because the RC Time object also shuts down a given cog immediately after the measurement is complete. Keep in mind that if your application launches four simultaneous RC decay measurements, there will four cogs occupied with simultaneous measurements for a brief period of time.

Figure 6: Parallel Measurements
The second measurement does not wait for the first measurement to finish.



If you have an oscilloscope:

- ✓ Examine how the duration of the first measurement can delay the start of the second measurement in "Test Simple RCTIME.spin" while the second measurement in "Test Parallel RCTIME.Spin" starts a brief, fixed time after the first measurement starts.

Assembly code optimization: The RC Time.ASM object is in its 0.70 revision and is currently a lot longer than it needs to be because it has not undergone any optimizations. Before it gets to v1.0, it will undergo additional testing and several iterations of assembly code optimization.

Establishing Sampling Rates

The RC Time object also has Start and Stop methods. The Start method makes it possible to make one or more copies of the RC Time object take growth/decay measurements at one or more different rates. When the application doesn't need any more measurements, the Stop method can be used to shut down the code and make it available for other tasks.

The RC Time object's Start method requires three additional parameters, charge time, timeout, and sample interval. The sample interval is the time between measurements, and it establishes the sampling rate. In the example code below, rc[0] is configured with a 0.1 ms charge time, a 1.0 ms timeout, and a 2 ms sample interval (clkfreq/500). rc[1] also has a 0.1 ms charge time, but its timeout is 0.5 ms and its sample interval is 1 ms (clkfreq/1000).

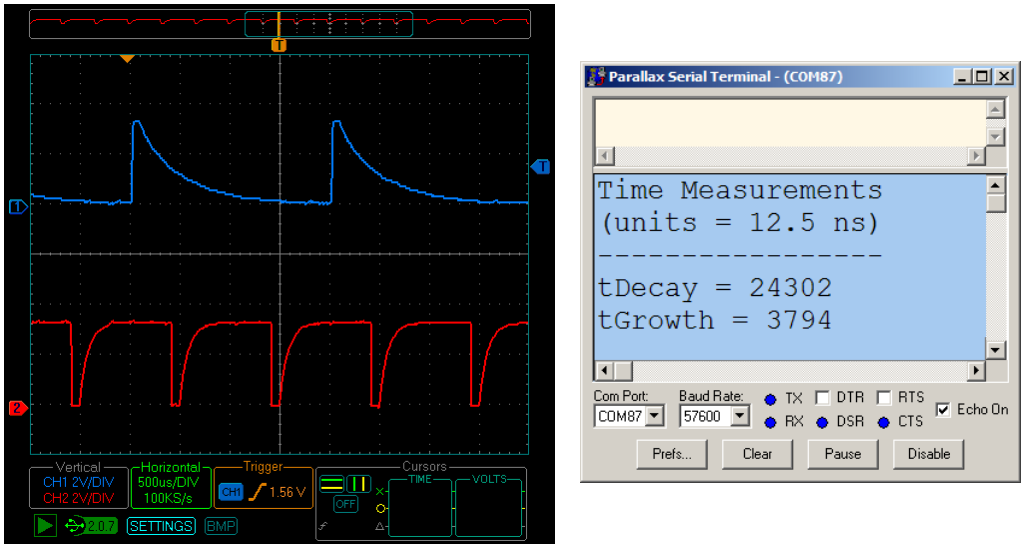
```
  ...

OBJ

  rc[2]       : "RC Time"                       ' RC Time.ASM is also an option
  debug       : "PST Debug LITE"                ' Variable & I/O display tool
  ...

PUB Go | tGrowth, tdecay

  debug.Start(115_200, debug#LIST)              ' 115.2 kbps, list display
  debug.Title(String("Time Measurements", debug#NL, "(units = 12.5 ns)"))
  debug.TitleAppend(String(debug#NL, "-----------------", debug#NL))
```

```
' Set up cogs to repeatedly take measurements and update tDecay & tGrowth.
rc[1].start(27, 0, clkfreq/1_0000, clkfreq/2000, clkfreq/1000, @tGrowth)
rc[0].start(17, 1, clkfreq/1_0000, clkfreq/1000, clkfreq/500, @tDecay)

repeat
  debug.VarsHome(@tDecay, String("| tDecay, tGrowth"))
  waitcnt(clkfreq/10 + cnt)
```

After the Start method for each RC Time instance gets called, the repeat loop can just send the latest measurement stored in each variable (tDecay and tGrowth) to PST Debug LITE for viewing in the Parallax Serial Terminal. One of the parameters in the start method is the address of the variable that the RC Time object should store its result in. Since both instances are taking repeated measurements, the tDecay and tGrowth measurements both store the latest value. tDecay gets updated at 500 Hz, and tGrowth gets updated at 1 kHz.

Figure 7 shows how the two instances of the RC Time object, configured to independent sampling rates, and both repeatedly take RC measurements. The upper trace shows rc[0], which repeats its measurements at a T = 2 ms sampling interval. Since sampling frequency is the inverse of sampling interval, (f = 1/T), f = 1/(2 ms) = 500 Hz. The lower trace shows the measurements rc[1] takes and stores in tGrowth at a sample interval of 1 ms and a sampling frequency (or sampling rate) of 1 kHz.

Figure 7: Two Different Sampling Rates and the Parallax Serial Terminal Results



- ✓ Open "Test Repeated RCTIME.spin" with the Propeller Tool software.
- ✓ Open the RC Time object and examine the minimum times allowed for the start method's chargeTimeTicks, timeOutTicks, and sampleTicks parameters.
- ✓ Try a variety of sampling rates, and if you have an oscilloscope, use it to examine the repeated RC measurements.

More PE Kit Sensor Examples

The RC decay technique can be used with a variety of sensors. Examples of parts in the PE Kit that can be used as RC Time compatible sensors include:

- Rotational position with a potentiometer
- Ambient light with a phototransistor or photoresistor*

- Infrared light with an infrared phototransistor
- Lightly filtered ambient light with LEDs
- Infrared light and flame with an infrared LED

* Whether your PE kit has a photoresistor or a phototransistor depends on its vintage. Older PE Kits have photoresistors. The newer RoHS compliant kits use phototransistors for light detection instead.

As we've already seen, the potentiometer's resistance varies with rotational position, i.e. where the knob is turned. Next, phototransistors regulate current passing through based on light intensity, which can also be measured with RC Time. LEDs that are reverse biased in a modified RC decay circuit behave like tiny solar cells that discharge their own junction capacitance and can also be measured with RC Time.

More RC Time Compatible Sensors at the Parallax web site

As mentioned earlier, there are lots of different sensors that are compatible with RC Time measurements. Here are a few examples from www.parallax.com:

- AD592 Temperature Probe
- Flexiforce Sensor
- HS1101 Capacitive Humidity Sensor
- QTI Sensor (for close-up line detection, but can also detect shades of gray)
- Blue Enhanced Photodiode

Nix the series resistor!

The documentation for these sensors typically includes a small resistor connected between the I/O pin and the capacitor. This series resistor protects a BASIC Stamp module's I/O pin from any brief current spike at the instant when the I/O pin starts charging the capacitor. Propeller I/O pins do not need these series resistors. Furthermore, if you remove the series resistor, sensors that vary linearly with the physical property they measure will yield linear RC Time measurement results. The series resistor changes it to a nonlinear function. If it's a resistive sensor that could drop below 100 Ω, make sure to add a resistor between the element and ground, like in Figure 1.

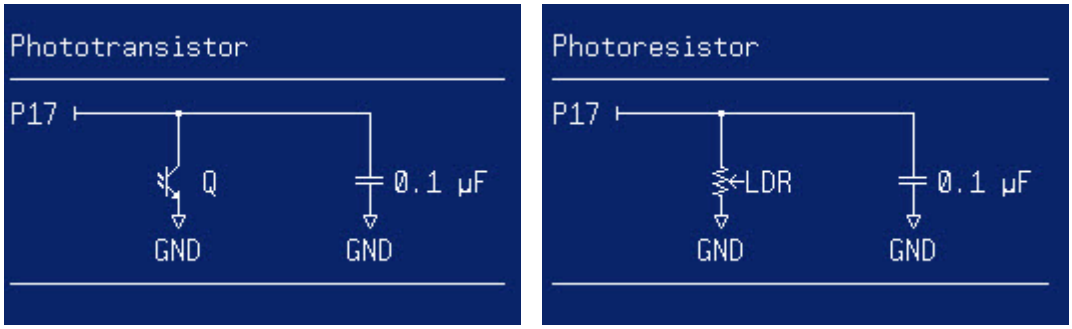
Ambient and Infrared Phototransistors

Newer PE kits have two phototransistors, shown in Figure 8 on the left and right. The ambient phototransistor has a clear plastic case and is sensitive to wavelengths from green to infrared, and is sometimes used in products to replace photoresistors for ambient light detection. The cadmium sulfide photoresistor in the center of Figure 8 is no longer included in the PE Kit due to recent RoHS (Restrictions on certain Hazardous Substances) rules. Its use in commercial products is also dropping off for the same reason. The infrared phototransistor looks like a smaller LED with a black coating. That coating filters for infrared. Strong sources of infrared include the sun, and the clear infrared LEDs included in the Propeller Education Kit.

Figure 8: Ambient Phototransistor, Photoresistor, Infrared Phototransistor

Phototransistors allow more current to pass with more light. Similarly, a photoresistor's resistance drops with more light. The result with either device is the same –more light leads to smaller decay times. The quickest way to test these parts is to remove the potentiometer and replace it with either the phototransistor or the photoresistor. Figure 9 shows RC Time circuit schematics for the phototransistors and also for the photoresistor. If you are using the ambient phototransistor, connect the longer pin to the I/O pin. If you are using the infrared phototransistor, the longer pin connects to ground. With the photoresistor, direction doesn't matter.

Figure 9: Phototransistor and Photoresistor Light Sensor Schematics



The photoresistor and ambient light phototransistor both work well with Test Simple RCTIME.spin. The infrared phototransistor needs a timeout that's longer than the default, so if you test that part, make sure to use Test RCTIME Timeout.spin, and set the timeout to something generous, like a fifth of a second (clkfreq/5).

- ✓ Substitute your light sensor of choice in place of the potentiometer.
- ✓ Run Test Simple RCTIME.spin (or a modified Test RCTIME Timeout.spin for the infrared phototransistor.)
- ✓ Examine tDecay under a variety of lighting conditions.

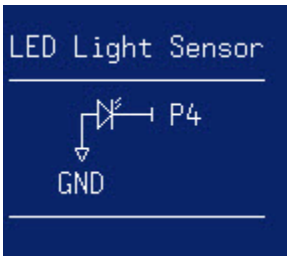
Measure Light with LEDs

LEDs as light sensors can sometimes serve as an inexpensive substitution for photodiodes, and have the advantage over phototransistors of an output that's linearly proportional to light level. They also respond well to direct sunlight, which tends to saturate phototransistors. Each color of LED also has some inherent filtering, and responds most the color of light it emits when it's forward biased. LEDs also have inherent capacitance across the junction of the two silicon materials with different impurities that electrons have to cross to emit photons. The capacitance across this junction is not surprisingly called junction capacitance. The LED's junction capacitance, along with the capacitance

inherent to the metal clips in your breadboard make it possible to use the Figure 10 circuit for LED light measurements, with no external capacitor.

!	**CAUTION – double check your LED connection before reconnecting power!** The LED in Figure 10 needs to be reverse biased, with the longer anode pin connected to ground and the shorter cathode pin connected to I/O pin P4.

Figure 10: LED Light Sensor Schematic



- ✓ Build the circuit shown in Figure 10. Make sure that the LED's longer anode pin connects to ground.
- ✓ Save Test RCTIME Timeout.spin as Test LED Light Sensor.spin
- ✓ Change the Timeout method call's parameter to half a second (clkfreq/2).
- ✓ Change rc.time(17, 1, @tDecay) to rc.time(4, 1, @tDecay).
- ✓ Load the program into the Propeller chip.
- ✓ Examine tDecay under a variety of lighting conditions.

Proximate Flame Detection with Infrared LEDs

You can replace one of the colored LEDs in Figure 10 with the clear infrared LED to detect nearby flames. For best results, point the dome on the top of the infrared led directly at the flame. Provided the detector is kept out of direct sunlight, a range of half a meter or more is feasible.

Application Example: Resistance Meter

The "RC Resistance Meter.spin" application circuit and Parallax Serial Terminal display are shown in Figure 11. The RC circuit is simply the P17 connection we have been using. The approach to taking measurements is somewhat different from a simple RC decay measurement. First, the RC Resistance object that the RC Resistance Meter application uses for measurements takes an average of 200 decay measurements to eliminate the effect of electrical noise. Second, the application needs to be calibrated. The calibration procedure is very simple, and corrects the cumulative effects of the various sources of error in the circuit, which include: actual capacitance value different from nominal (named) value, I/O pin threshold not exactly 1.65 V, stray resistances and capacitances in the circuit and prototyping board, and any deviation from nominal in the 100 Ω series resistor value. Since each of these errors contributes to one of the terms in $\Delta t = 0.693 \times C \times R$, they can all be corrected with the scalar and offset from our old algebraic friend $y = mx + b$.

Figure 11: Resistance Meter Circuit and Display (1.829 kΩ)

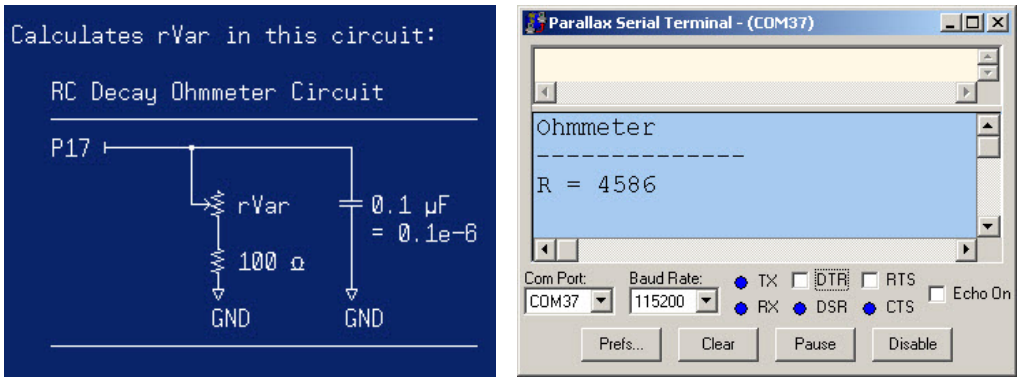


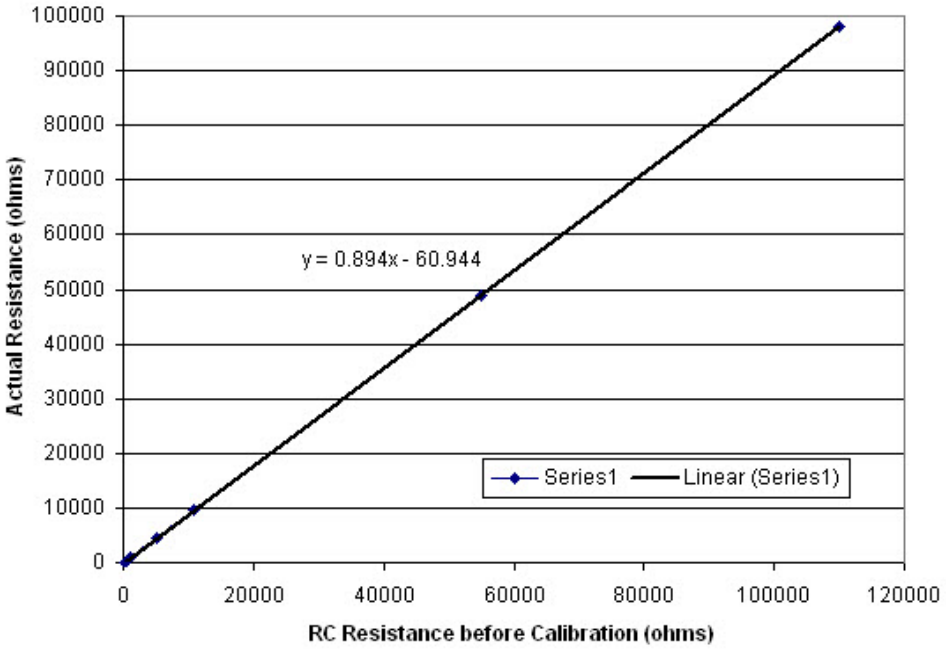
Table 1 shows errors before and after calibration. Note that calibration can reduce 50% errors down to below 1%. A minimal calibration can be accomplished by comparing two measurements against known resistor values. This will give you two (x, y) data points, where x is the measured value, and y is the known value. By solving for m and b in y = mx + b, you will get the scale factor and offset values that can be declared as constants in the example program.

Table 1: Plotted Points and Resistor Values before and after Calibration

	Plotted Points				
	y-axis values	x-axis values			
		Measured	Error	Measured	Error
Nominal	Actual	Before	Before	After	After
Values	Values	Calibration	Calibration	Calibration	Calibration
(ohms)	(ohms)	(ohms)	%	(ohms)	%
100	98.3	197	-50.10	98	0.31
470	461.6	583	-20.82	459	0.57
1000	980	1159	-15.44	974	0.62
4700	4605	5203	-11.49	4586	0.41
10 k	9778	10962	-10.80	9732	0.47
50 k	48980	54944	-10.85	49104	-0.25
100 k	98220	109900	-10.63	98140	0.08

A better calibration procedure is to take more (x, y) data points and then use a spreadsheet to calculate the average of m and b. For example, the seven x and y measurements in Table 1 are graphed with a Microsoft Excel as shown in Figure 12. Spreadsheets also have tools that can display a trend line and an equation that yields the average values of m and b for all the points in the graph. For the measurements in table 1, Figure 12 shows that m = 0.894 and b = –60.944 for the test circuit. Keep in mind that your test circuit will almost certainly have different m and b values.

Figure 12: Actual Vs. RC Measured Resistances Before Calibration



The application relies on the RC Resistance Meter.spin object for both calibration and testing. Before calibration, this object has its scalar constant set to 1.0 and its offset constant set to 0.0. The RC Resistance Meter.spin object passes these values, along with CAP = 0.1e-6 to the RC Resistance object. The RC Resistance object in turn uses this equation to calculate the resistance based on the time measurement along with the scalar, offset, and CAP values it received:

$$rVar = \frac{\Delta t \times Scalar}{0.693 \times CAP} - Offset$$

Δt is seconds, not ticks. The Resistance Meter.spin object also converts the RC Time measurement from clock ticks to a floating point representation of seconds before multiplying by scalar in the rVar calculation.

Before calibration, the measured resistance values will be linearly related to the actual values. You can then use the spreadsheet included with the example programs to calculate the new values for scalar (m) and offset (b). Or, for a quick and lower precision calibration, simply use a couple of data points, (like resistance measurements for1 kΩ and 10 kΩ) and y = mx + b for a rough calibration. Here is the procedure for more precise calibration with the spreadsheet:

- ✓ Open "RC Resistance Meter Calibration Example.xls". It's included in the "PE Kit Lab Tools – Measure Resistance or Capacitance.zip" file.
- ✓ Use a known good ohmmeter to measure the PE Kit resistors listed in Table 1's Nominal Values column. Record your measurements in the y-axis values column.
- ✓ Open "RC Resistance Meter.spin" with the Propeller Tool software.
- ✓ In the CON block, make sure that the scalar = 1.0 and offset = 0.0.
- ✓ Remove the Pot from the P17 RC decay circuit you built from Figure 1. Each test resistor should be plugged in where the pot's wiper and B terminal were in the RC Decay circuit in Figure 1.

- ✓ Run the program, and measure each resistor in the RC circuit, and record each measurement in the x-axis values column.
- ✓ Copy the updated trend line values for m and b that appear in the graph above the trend line into the scalar and offset CON block declarations.
- ✓ Load the modified program into the Propeller chip, and measure the resistors in the RC circuit again, this time, using the results to populate the Measured After Calibration column.

Application Example: 200 kHz Sampling Rate

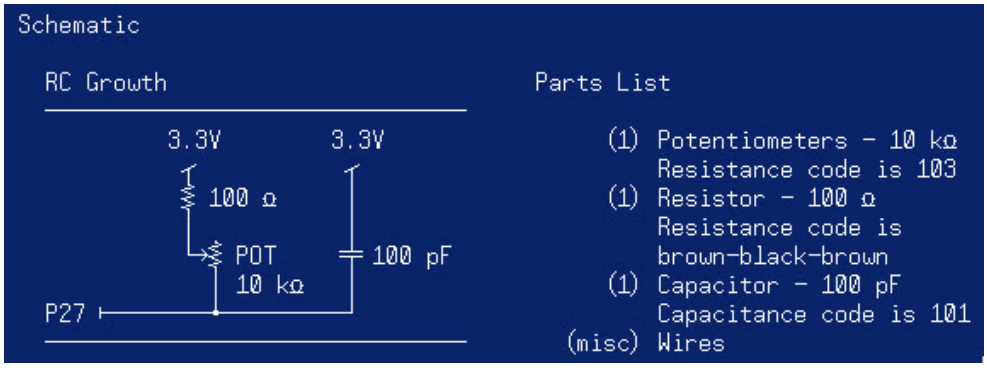
If you need a sampling rate that's faster than what RC Time.spin can provide, use RC Time.ASM.spin. It can support sampling intervals as short as a few hundred clock ticks, as opposed to the several thousand minimum in the RC Time.spin object's documentation. Let's say that your application requires a 200 kHz sampling rate. The sampling interval is T = 1/f = 1/200 kHz = 5 µs. Assuming the Propeller chip's system clock is running at 80 MHz, the number of clock ticks in the sampling interval would be:

```
sample interval clock ticks = ticks in 1 second × sample interval
                            = 80 MHz × 5 µs
                            = 400 clock ticks
```

Figure 13 shows an RC growth circuit that responds 100 times more quickly than the circuit in Figure 1. That's because the capacitor is 1/100 the size of the one in Figure 1. Instead of growth times in the 0 to 5000 clock tick range, the measurements will be in the 0 to 50 clock tick neighborhood.

- ✓ Modify the RC growth circuit connected to P27 according to Figure 13.

Figure 13: A Faster RC Circuit



These excerpts from "200 kHz Sampling rate.spin" configure the RC Time.ASM object to charge the circuit's capacitor for 50 clock ticks, and allow 125 clock ticks for the decay before timeout, repeating every 400 clock ticks.

```
  ...

OBJ
  rc        : "RC Time.ASM"                 ' RC Time.spin is also an option
  debug     : "PST Debug LITE"              ' Variable & I/O display tool
  ...

PUB Go | tGrowth, repsAddr

  debug.Start(115_200, debug#LIST)
  ...
```

```
repsAddr := rc.start(27, 0, 50, 125, 400, @tgrowth)

repeat
  debug.VarsHome(@tgrowth, String("| tgrowth"))
  debug.Vars(repsAddr, String("| @repsAddr"))
  waitcnt(clkfreq/10 + cnt)
```

According to the RC Time.ASM object's documentation, its Start method returns the address of the object's repsAddr variable, which stores the number of samples the object has taken. (Unless all cogs were busy, in which case the Start method returns zero.) Example code in the "200 kHz Sampling Rate.spin" object uses this variable address along with a modified versions of the Display object to list the number of measurements (reps) that RC Time.ASM has taken. Every second, the repetitions display increments by 200,000. Figure 14 shows the display at about the 10 second mark.

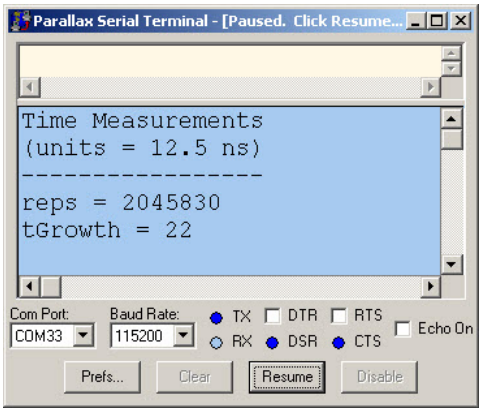
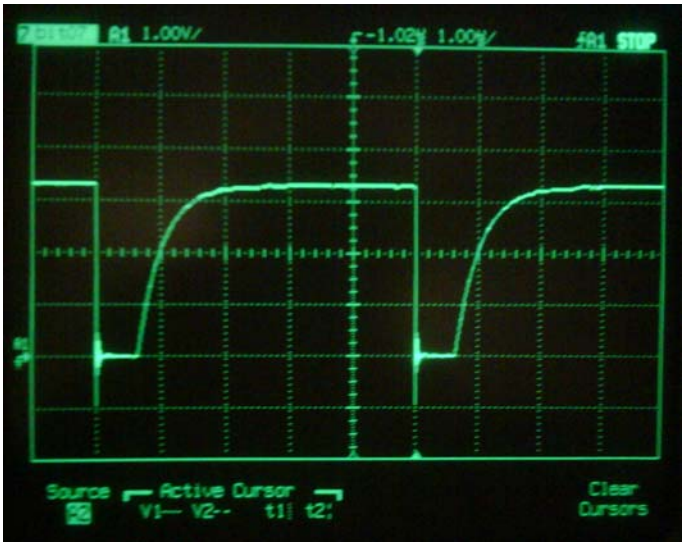


Figure 14: RC Decay and Growth Measurements

Figure 15 shows the RC measurements on a 100 MHz oscilloscope set to 1 µs/division. Note that the I/O pin switches to output-low to recharge the circuit every 5 µs demonstrating the 200 kHz sampling rate.

Figure 15: Oscilloscope View of 200 kHz Sampling Rate.Spin



- ✓ Open "200 kHz Sampling Rate.spin" with the Propeller Tool software.
- ✓ Open the RC Time ASM.spin object, and view it in Documentation mode. Read the Start method's documentation, and pay special attention to how many more clock ticks than the timeout the sample interval has to have.

- ✓ Try a variety of sampling rates, and if you have an oscilloscope, use it to examine the repeated RC measurements.