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

Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

Errata for What's a Microcontroller? Text v3.0 (#28123)

If you find what may be additional errata items not listed here, please email editor@parallax.com. We appreciate your sharp eyes!

Text errors are noted with ~~red strikethrough text~~, and corrections with blue text, in the sections below. Formatted PDF replacement pages for each correction are appended to this document.

Page 143

How the Potentiometer Circuit Works

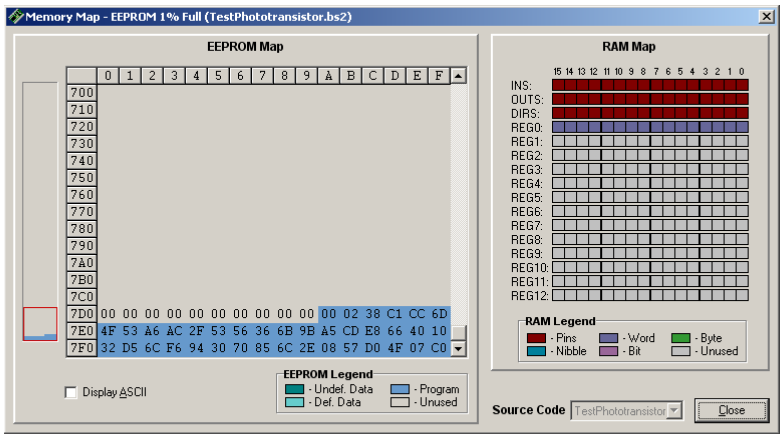
The total resistance in your test circuit is 220 Ω plus the resistance between the A and W terminals of the potentiometer. The resistance between the A and W terminals increases as the knob is adjusted further ~~clockwise~~ counterclockwise, which in turn reduces the current through the LED, making it dimmer.

Pages 166-167

```
' What's a Microcontroller - Ch5Prj01_ControlServoWithPot.bs2
' Read potentiometer in RC-time circuit using RCTIME command.
~~' The time var ranges from 126 to 713, and an offset of 330 is needed.~~
' Use RCTIME result in time variable to control servo position.
' Bicolor LED on P12, P13 tells direction of servo rotation:
' green for CW, red for CCW, off when servo is holding position.
' {$STAMP BS2}
' {$PBASIC 2.5}

PAUSE 1000
DEBUG "Program Running!"

time            VAR     Word                ' time reading from pot
prevTime        VAR     Word                ' previous reading

DO

  prevTime = time                           ' Store previous time reading
  HIGH 7                                    ' Read pot using RCTIME
  PAUSE 10
  RCTIME 7, 1, time
~~  time = time + 350                         ' Scale pot, match servo range~~
  time = time */ 185                        ' Scale by 0.724 (X 256 for */).
  time = time + 500                         ' Offset by 500.
```

Page 204

The memory map does not match the one for the program in Activity #1. This is the correct memory map display.



Memory Map - EEPROM 1% Full (TestPhototransistor.bs2)

EEPROM Map

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
700																
710																
720																
730																
740																
750																
760																
770																
780																
790																
7A0																
7B0																
7C0																
7D0	00	00	00	00	00	00	00	00	00	00	00	02	38	C1	CC	6D
7E0	4F	53	A6	AC	2F	53	56	36	6B	9B	A5	CD	E8	66	40	10
7F0	32	D5	6C	F6	94	30	70	85	6C	2E	08	57	D0	4F	07	C0

☐ Display ASCII

EEPROM Legend
- Undef. Data
- Def. Data
- Program
- Unused

RAM Map

RAM Legend
- Pins
- Nibble
- Word
- Bit
- Byte
- Unused

Source Code TestPhototransistor

Close

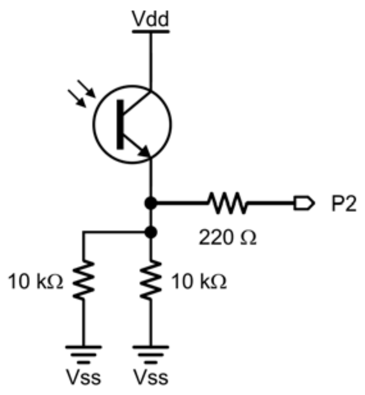
Figure 0-1a
Memory Map

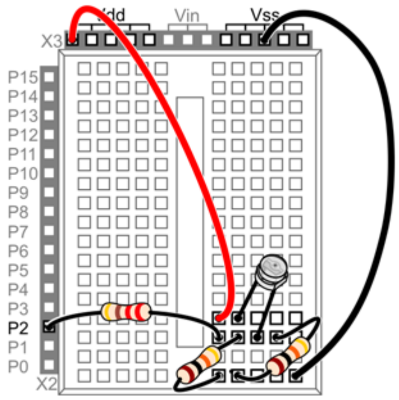
To view this window, click Run, and select Memory Map.

Pages 232-234

If your What's a Microcontroller v3.0 kit did not contain a 4.7 kΩ resistor, you can instead use two 10 kΩ resistors in parallel as shown in Figure 7-20a. The equivalent resistance of two 10 kΩ resistors in parallel is 5 kΩ. You can also use this approach with two 100 kΩ resistors in parallel for an equivalent resistance of 50 kΩ. (Since this is not a book error specifically, no replacement PDF page is provided.)

Figure 0-2a: Schematic and Wiring Diagram that Utilize two 10 kΩ Resistors in Parallel for an Equivalent Resistance of 5 kΩ.





Equivalent Resistance for Series and Parallel Values.

When two or more resistors are connected in series, the equivalent resistance is:

$R_{EQ} = R_1 + R_2 + R_3…$ (Equivalent resistance for resistors in series)

When two or more resistors are connected in parallel, their equivalent resistance is:

$R_{EQ} = 1 \div (1/R_1 + 1/R_2 + 1/R_3…)$ (Equivalent resistance for resistors in parallel)

For two 10 kΩ resistors in parallel, that's 1 ÷ (1/10k + 1/10k) = 1 ÷ (2/10k) = 10k/2 = 5 k. Two equal value resistors in parallel allow twice as much current through as one of them in its own would let through. So it stands to reason that the equivalent resistance for two equal value resistors in parallel would be one half of the value.

Equivalent Capacitance for Series and Parallel Values.

It's easiest to remember how to calculate equivalent capacitance if you think about it as the reverse of series and parallel resistor calculations. So, the equivalent capacitance for parallel capacitors adds up, and equivalent capacitance for series capacitors uses inverses.

$C_{EQ} = C_1 + C_2 + C_3…$ (Equivalent capacitance for capacitors in parallel)

In the previous activity, two capacitors were placed in parallel to double the capacitance. For two 0.01 μF capacitors, that's C_{EQ} = 0.01 μF + 0.01 μF = 0.02 μF

When two or more capacitors are placed in series, their equivalent capacitance is:

$C_{EQ} = 1 / (1/C_1 + 1/C_2 + 1/C_3…)$ (Equivalent capacitance for capacitors in series)

<<<<<END OF ERRATA LIST. FORMATTED CORRECTED PDF PAGES APPENDED>>>>>>

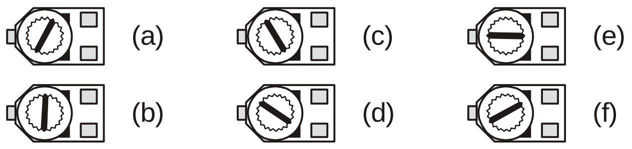


Figure 5-6
Potentiometer Knob

(a) through (f) show the potentiometer's wiper terminal set to different positions.

How the Potentiometer Circuit Works

The total resistance in your test circuit is 220 Ω plus the resistance between the A and W terminals of the potentiometer. The resistance between the A and W terminals increases as the knob is adjusted further counterclockwise, which in turn reduces the current through the LED, making it dimmer.

ACTIVITY #2: MEASURING RESISTANCE BY MEASURING TIME

This activity introduces a new part called a *capacitor*. A capacitor behaves like a rechargeable battery that only holds its charge for short durations of time. This activity also introduces RC-time, which is an abbreviation for resistor-capacitor time. RC-time is a measurement of how long it takes for a capacitor to lose a certain amount of its stored charge as it supplies current to a resistor. By measuring the time it takes for the capacitor to discharge with different size resistors and capacitors, you will become more familiar with RC-time. In this activity, you will program the BASIC Stamp to charge a capacitor and then measure the time it takes the capacitor to discharge through a resistor.

Introducing the Capacitor

Figure 5-7 shows the schematic symbol and part drawing for the type of capacitor used in this activity. Capacitance value is measured in microfarads (μF), and the measurement is typically printed on the capacitors.

The cylindrical case of this particular capacitor is called a *canister*. This type of capacitor, called an *electrolytic capacitor*, must be handled carefully.

- ✓ Read the CAUTION box on the next page.

Solutions

Q1. A potentiometer.

Q2. No, it's fixed. The variable resistance is between either outer terminal and the wiper (middle) terminal.

Q3. A capacitor is like a rechargeable battery in that it can be charged up to hold voltage. The difference is that it only holds a charge for a very small amount of time.

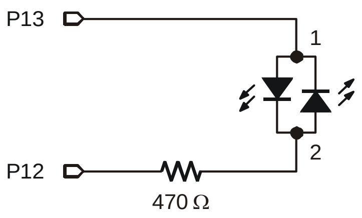
Q4. You can measure the time it takes for the capacitor to discharge (or charge). This time is related to the resistance and capacitance. If the capacitance is known and the resistance is variable, then the discharge time gives an indication of the resistance.

Q5. As R gets larger, the RC discharge time increases in direct proportion to the increase in R. As R gets smaller, the RC discharge time decreases in direct proportion to the decrease in R.

Q6. The **`CON`** directive substitutes a name for a number.

E1. New cap = (10 x old cap value) = (10 x 0.5µF) = 5 µF

P1. Activity #4 with bicolor LED added.



Potentiometer schematic from Figure 5-11 on page 151, servo from Chapter 4, Activity #1, and bicolor LED from Figure 2-19 on page 53 with P15 and P14 changed to P13 and P12 as shown.

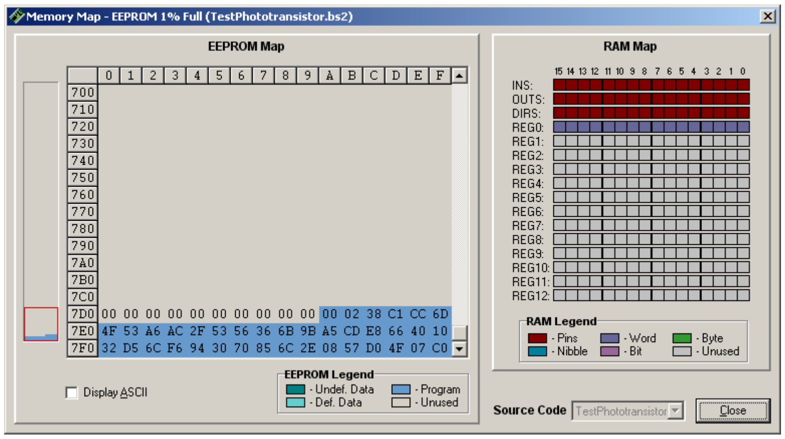
```
' What's a Microcontroller - Ch5Prj01_ControlServoWithPot.bs2
' Read potentiometer in RC-time circuit using RCTIME command.
' Use RCTIME result in time variable to control servo position.
' Bicolor LED on P12, P13 tells direction of servo rotation:
' green for CW, red for CCW, off when servo is holding position.
' {$STAMP BS2}
' {$PBASIC 2.5}

PAUSE 1000
DEBUG "Program Running!"

time           VAR     Word            ' time reading from pot
prevTime       VAR     Word            ' previous reading
```

```
DO

  prevTime = time                       ' Store previous time reading
  HIGH 7                                ' Read pot using RCTIME
  PAUSE 10
  RCTIME 7, 1, time
  time = time */185                     ' Scale pot, match servo range
  time = time + 500                     ' Scale pot, match servo range
  IF ( time > prevTime + 2) THEN        ' increased, pot turned CCW
    HIGH 13                             ' Bicolor LED red
    LOW 12
  ELSEIF ( time < prevTime - 2) THEN    ' value decreased, pot turned CW
    LOW 13                              ' Bicolor LED green
    HIGH 12
  ELSE                                  ' Servo holding position
    LOW 13                              ' LED off
    LOW 12
  ENDIF

  PULSOUT 14, time

LOOP
```

P2. The key is to add **`IF...THEN`** blocks; an example is shown below. **`CLREOL`** is a handy **`DEBUG`** control character meaning "clear to end of line."

```
' What's a Microcontroller - Ch5Prj02_ControlServoWithPot.bs2
' Read potentiometer in RC-time circuit using RCTIME command.
' Modify with IF…THEN so the servo only rotates from 650 to 850.
' The time variable ranges from 1 to 691, so an offset of at least
' 649 is needed.

' {$STAMP BS2}
' {$PBASIC 2.5}

PAUSE 1000
DEBUG "Program Running!"

time VAR Word

DO

  HIGH 7                               ' Read pot with RCTIME
  PAUSE 10

  RCTIME 7, 1, time
  time = time + 649                    ' Scale time to servo range

  IF (time < 650) THEN                 ' Constrain range from 650 to 850
    time = 650
  ENDIF
```

highlighted in blue, and only 35 bytes out of the 2048 byte EEPROM are used for the program. The remaining 2013 bytes are free to store data.



Memory Map - EEPROM 1% Full (TestPhototransistor.bs2)

EEPROM Map

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
700
710
720
730
740
750
760
770
780
790
7A0
7B0
7C0
7D0  00 00 00 00 00 00 00 00 00 00 00 02 38 C1 CC 6D
7E0  4F 53 A6 AC 2F 53 56 36 6B 9B A5 CD E8 66 40 10
7F0  32 D5 6C F6 94 30 70 85 6C 2E 08 57 D0 4F 07 C0
```

Figure 7-6
Memory Map

To view this window, click Run, and select Memory Map.

The EEPROM Map shows the addresses as hexadecimal values, which were discussed briefly in the Decimal vs. Hexadecimal box on page 183. The values along the left side show the starting address of each row of bytes. The numbers along the top show the byte number within that row, from 0 to F in hexadecimal, which is 0 to 15 in decimal. For example, in Figure 7-6, the hexadecimal value C1 is stored at address 7E0. CC is stored at address 7E1, 6D is stored at address 7E2, and so on, up through E8, which is stored at address 7EF. If you scroll up and down with the scroll bar, you'll see that the largest memory addresses are at the bottom of the EEPROM Map, and the smallest addresses are at the top, with the very top row starting at 000.

PBASIC programs are always stored at the largest addresses in EEPROM, which are shown at the bottom of the EEPROM Map. So, if your program is going to store data in EEPROM, it should start with the smallest addresses, starting with address 0. This helps ensure that your stored data won't overwrite your PBASIC program, which will usually result in a program crash. In the case of the EEPROM Map shown in Figure 7-6, the PBASIC program resides in addresses 7FF through 7DD, starting at the largest address and building to smaller addresses. So your application can store data from address 000 through 7DC, building from the smallest to the largest. In decimal, that's addresses 0 through 2012.

If you plan on storing data to EEPROM, it is important to be able convert from hexadecimal to decimal in order to calculate the largest writable address. Below is the