Amicus18 Companion Shield

Code and text edited by
Dave (mackrackit) McCormick
for use with
microEngineering Labs
PICBASIC Pro
August 2011

Amicus18 Companion Shield

Amicus18 Companion Shield

Amicus18 Companion Shield

A shield is a PCB that fits over the Amicus18 board and provides extra functionality, such as Ethernet, Motor control, LCD, Smartcard, GPS, GSM etc…

All Arduino shields will physically fit on the Amicus18, however, Arduino source code is not compatible with Amicus18, as they differ in two very crucial aspects. First, the Amicus uses a Microchip PICmicro[tm] for it's microcontroller, while the Arduino uses an Atmel AVR micro-controller. The Arduino uses a subset of the language C, where as the Amicus18's supplied language is BASIC. However, there is no reason that any PICmicro[tm] language cannot be used with Amicus18, in fact, it's encouraged.

The entry level shield, and in the authors opinion, the most useful, is the Companion shield. This is a PCB laid out in the pattern of a solderless breadboard. The holes are single sided, which means that components can easily be removed using solder mop braid, or a solder vacuum tool, if a mistake is made, or components need to be re-used.
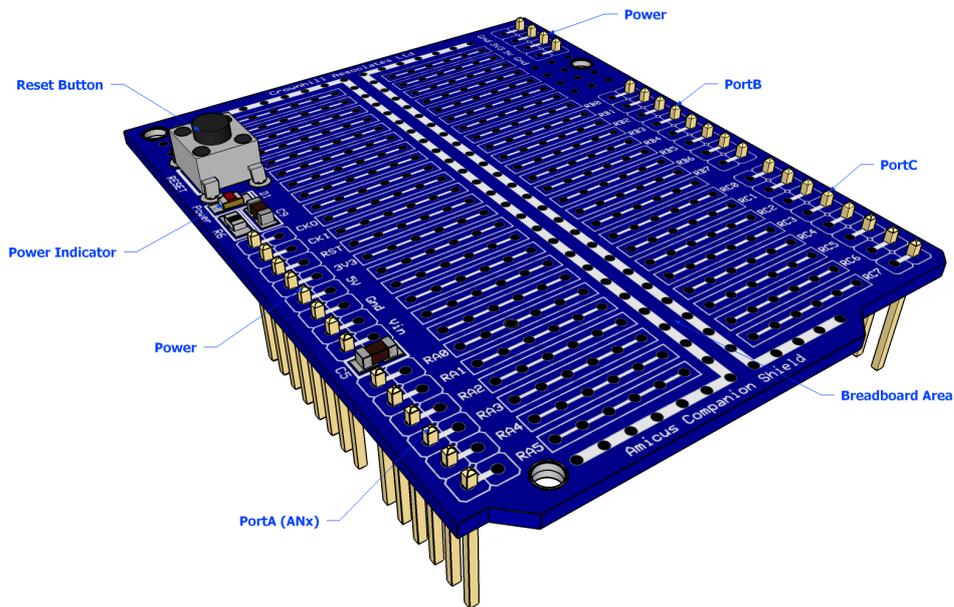

Another solution is to add a solderless breadboard to the Companion shield, thus allowing the full re-use of components without the need for a soldering iron. Notice the use of header sockets instead of header pins.

3

06-10-2009
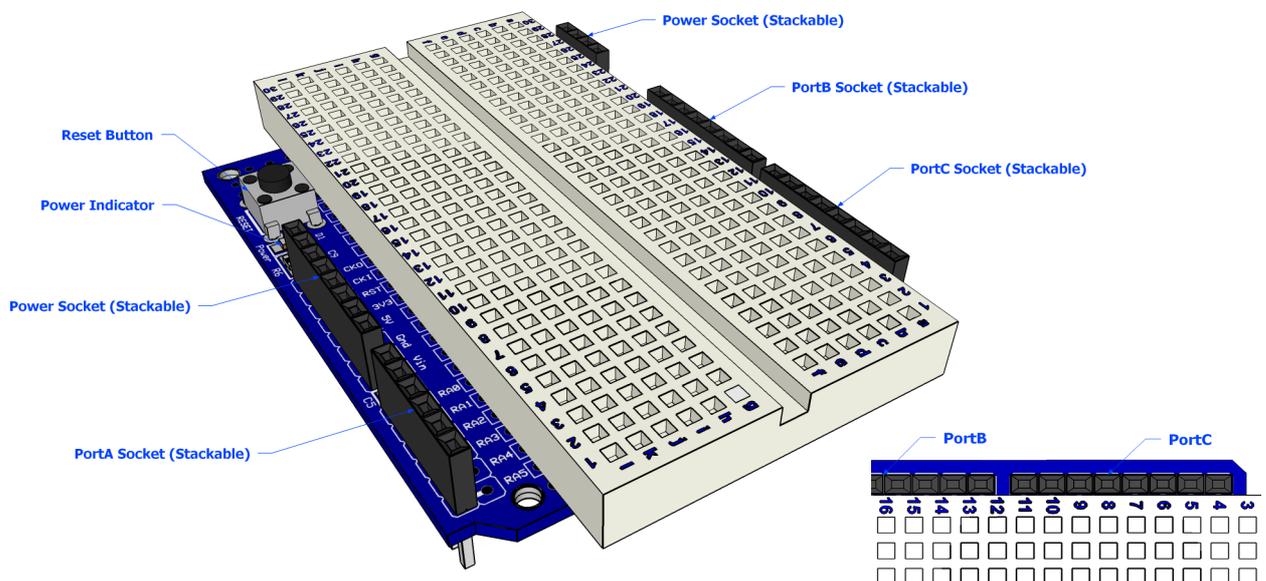
Amicus18 Companion Shield

Companion Shield Options

The companion shield is available as a blank PCB or ready built. However, there are two fla-
vours of the ready built boards, one with header sockets, and one with header pins. It all
depends on what you need to do with the companion shield. The illustrations below show
the various flavours:
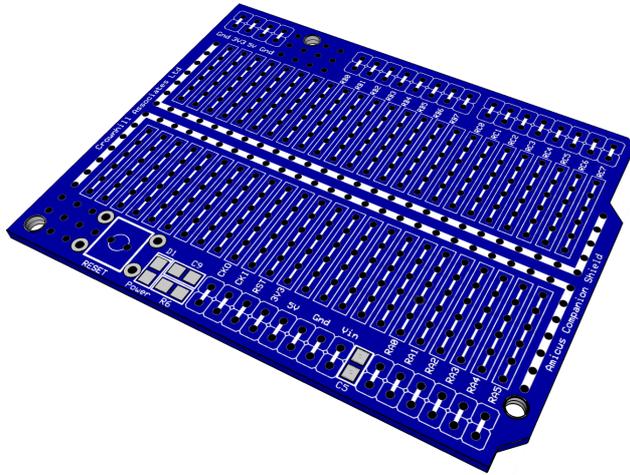
Blank Companion Shield
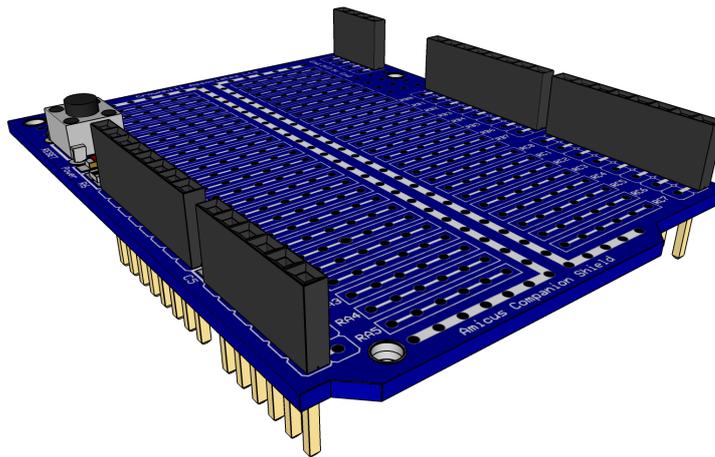
Companion Shield with Header Sockets
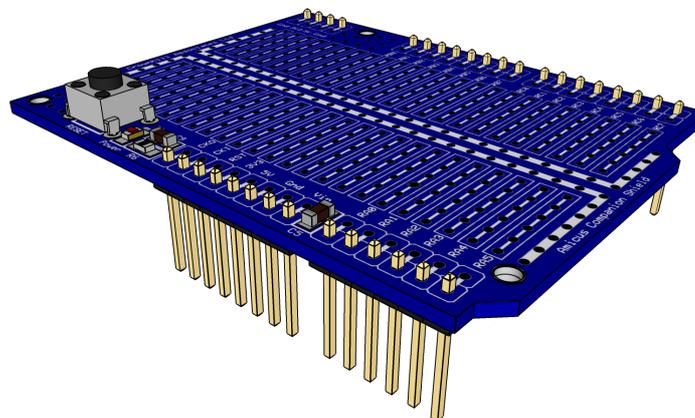
Companion Shield with Header Pins

4

Amicus18 Companion Shield

The two flavours of the shield allow the boards to be stackable or at the top of the stack:
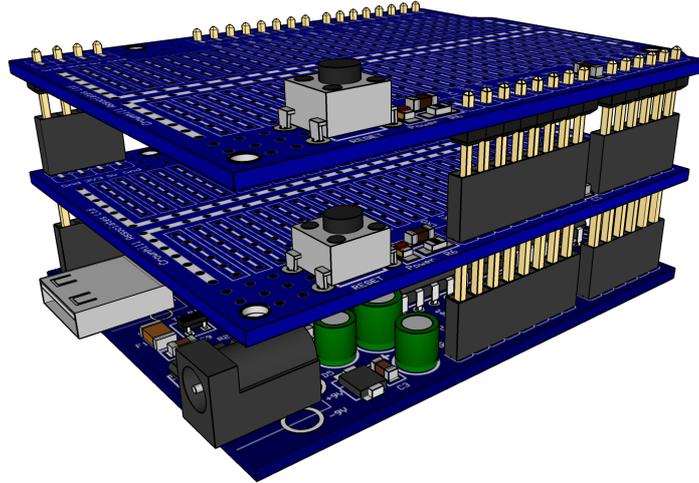

The illustration above shows the Amicus18 board at the bottom of the stack, then a socketed shield, then a pinned shield. A pinned shield could carry an LCD or other user interfacing device that would not suit being stacked between other PCBs. The socket and pin headers used for the companion shield have long legs, thus allowing plenty of clearance between the stacked PCBs, 12mm for the pinned header, and 14mm for the socketed header.

Building the Companion Shield

If you are going to choose the blank companion shield, it must be pointed out that it contains surface mount components (not supplied with it). These components are purely optional, but if you are considering using them, make sure you have the required skills to solder surface mount devices. It's not difficult, and there are plenty of SMT soldering tutorials on the internet.

Start by soldering the decoupling capacitors C5 and C9 on the board, both are 100nF 50 Volt ceramic capacitors with an 0805 casing:
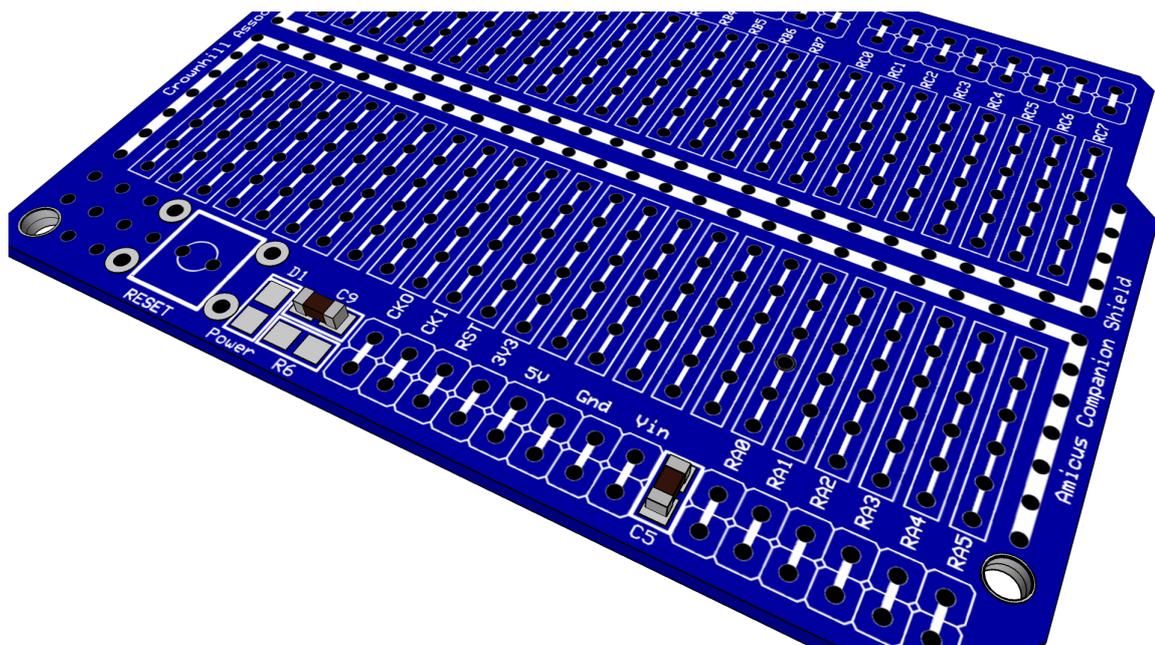
Next solder on the resistor R6 which is a 1KΩ 1% 0805 casing type:
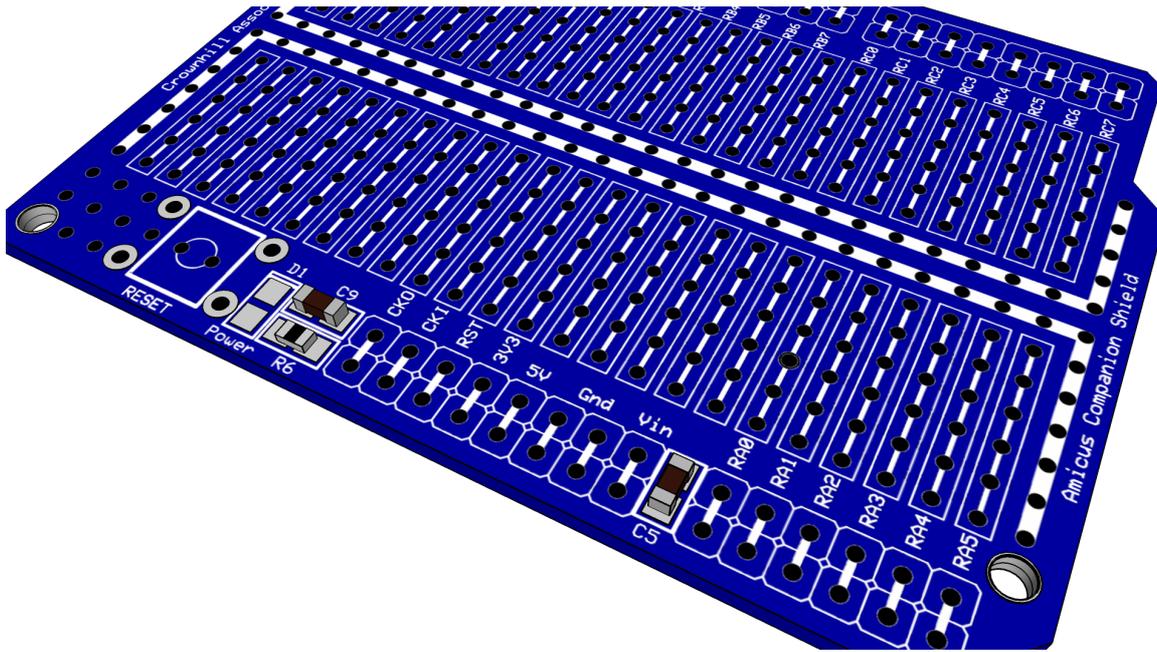
Next to solder is the power indicator LED, this is a red type 0805 casing, but any colour will do. Note that resistor R6 is not required if the LED is omitted:
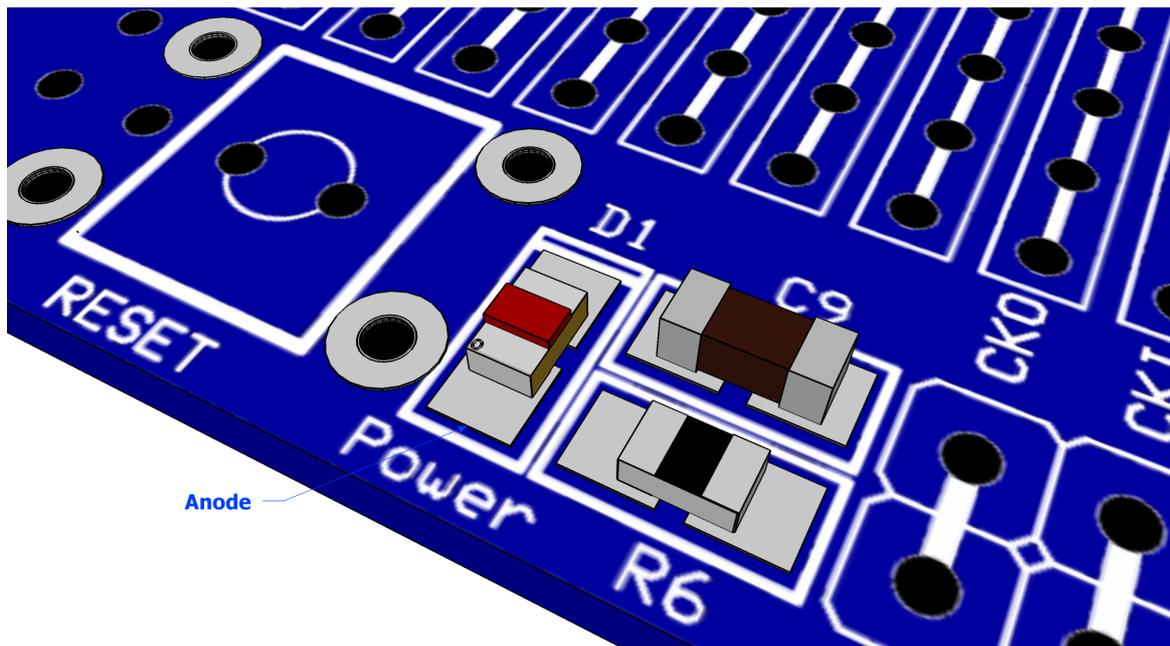
Take note of the orientation of the LED, make sure the Anode is located as in the above diagram. Reversing the LED won't harm it, it just won't illuminate.

The next component is the reset button, this is a standard PCB push to make type:
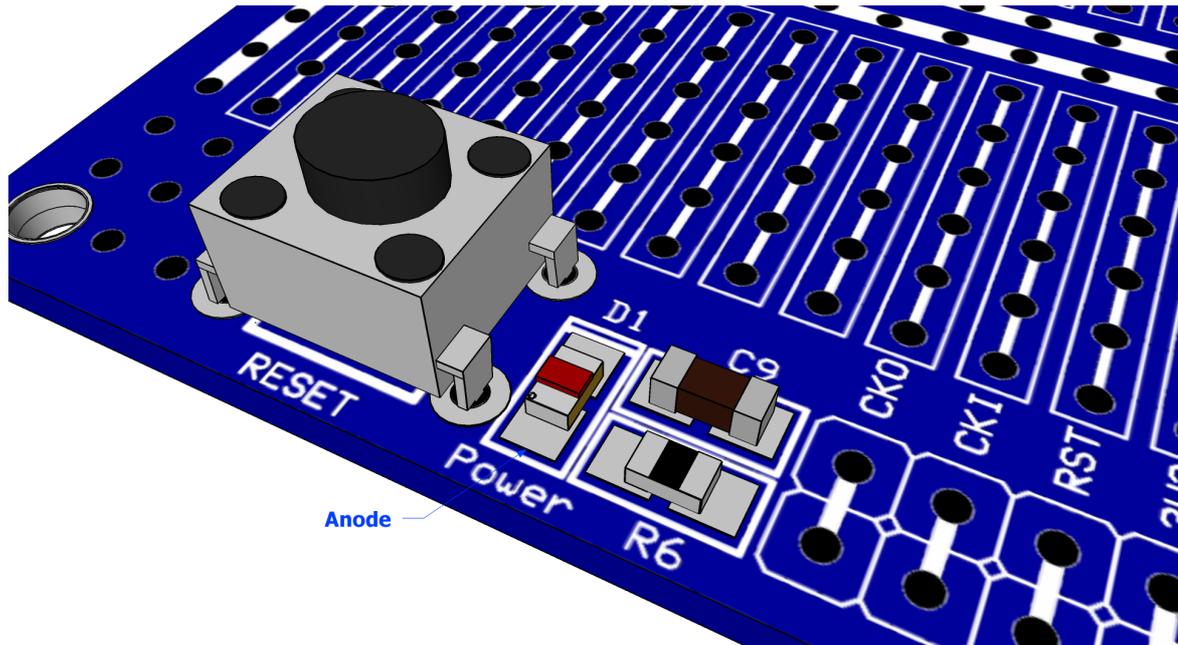
Then place either the header pins or the header sockets as the earlier diagram illustrate. These are standard 2.54 (0.1") spacing Single Inline types (SIL).

You will require 5 of these:

1 x 4 way
1 x 6 way
3 x 8 way

Amicus18 Companion Shield

First Program

We'll jump straight in at this point and produce our very first program that does something, but not using the companion shield just yet.

Open Micro Code Studio and type in the following code. Note that it is *not* required to type in the commented texts. i.e. *blue texts*:

```
' FLASH AN LED CONNECTED TO RB0
' MAKE SURE THE AMICUS18 BOARD'S JUMPER Q3 IS SET TO THE GND POSITION
    WHILE 1 = 1          ' CREATE AN INFINITE LOOP
       HIGH PORTB.0      ' BRING THE LED PIN HIGH (ILLUMINATE THE LED)
       PAUSE 500         ' WAIT 500MS (HALF A SECOND)
       LOW PORTB.0       ' PULL THE LED PIN LOW (EXTINGUISH THE LED)
       PAUSE 500         ' WAIT 500MS (HALF A SECOND)
    WEND                 ' CLOSE THE LOOP
```

Move jumper **Q3** to the **Gnd** position, and place an LED into PortB pins RB0 and RB1, with the Cathode connected to RB1, and the Anode connected to RB0. The Cathode is identified by being the shorter of the two wires, and also the body of the LED has a flattened side.
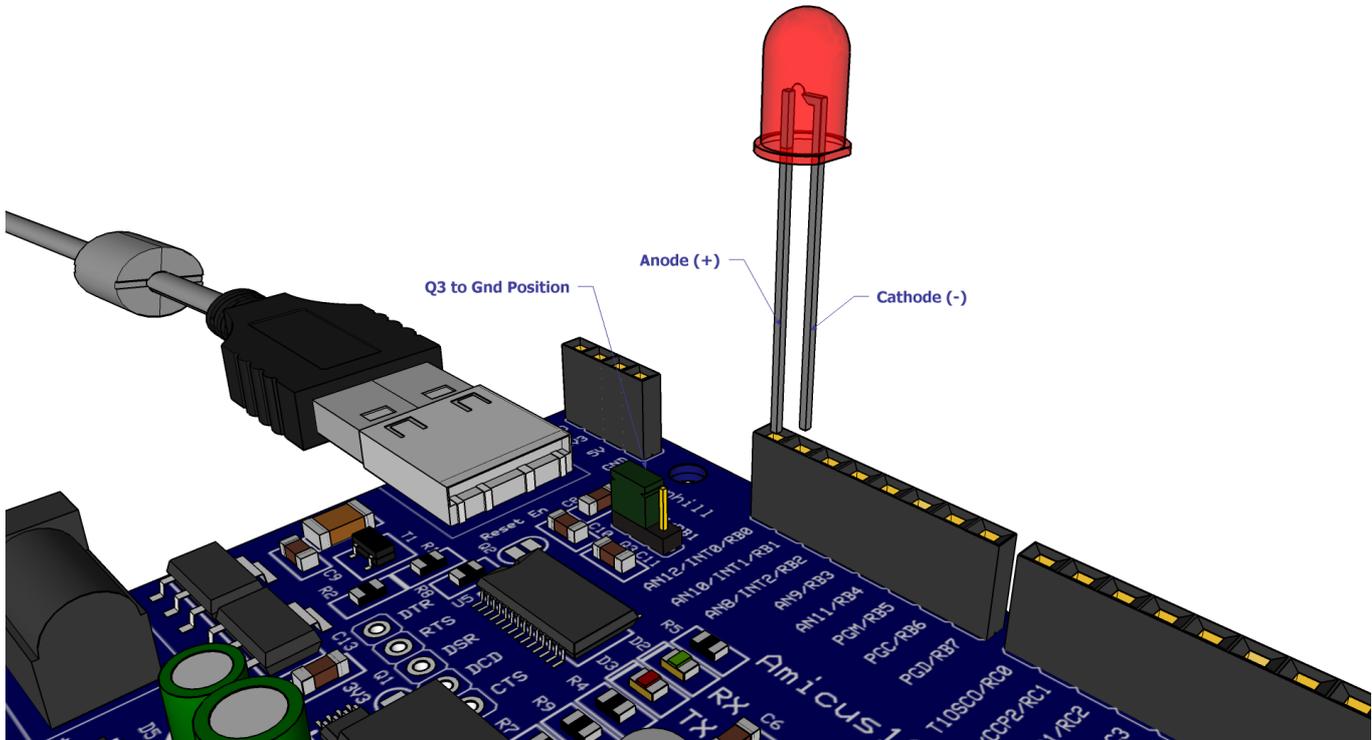
Connect the USB cable to the Amicus18 board, and make sure its red Power LED is illuminated. Press the *Compile and Program* button on the toolbar, or press F10. The code will then be compiled, and the bootloader will open to place the compiled code into the Amicus18's microcontroller. The LED will then start flashing.

The above layout works as expected, however, some rules have been broken in so much as the LED does not have a current limiting resistor in series with it. This means that the LED is seeing the full 3.3 Volts instead of it's working voltage of approx 2 Volts, and is pulling too much current from the microcontroller's IO pin. We can alleviate this situation by using the Companion Shield with a solderless breadboard.

The correct method for connecting an LED is shown overleaf.
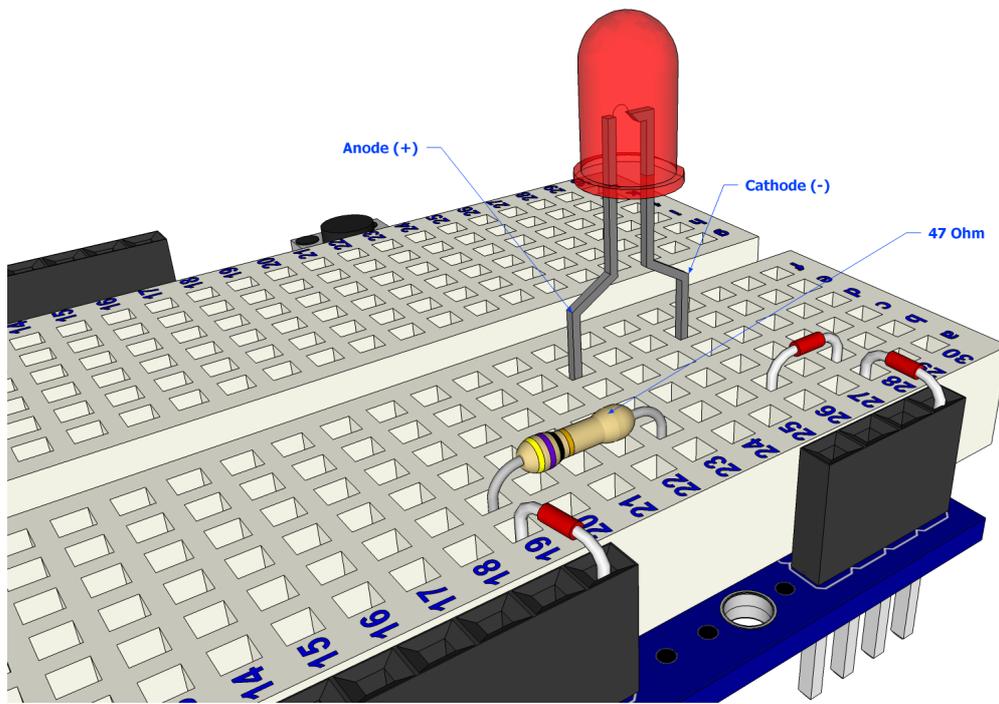
The circuit for the layout above is shown below:
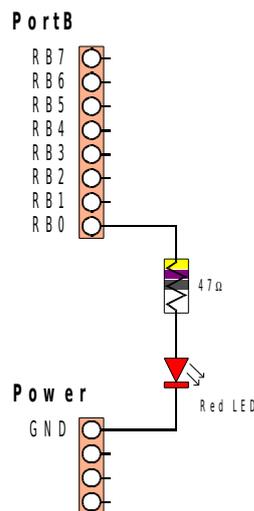
The same program may be used with the layout above, but this time the LED is protected from over voltage and over current.

```
' FLASH AN LED CONNECTED TO RB0
' MAKE SURE THE AMICUS18 BOARD'S JUMPER Q3 IS SET TO THE GND POSITION
  WHILE 1 = 1            ' CREATE AN INFINITE LOOP
    HIGH PORTB.0         ' BRING THE LED PIN HIGH (ILLUMINATE THE LED)
    PAUSE 500            ' WAIT 500MS (HALF A SECOND)
    LOW PORTB.0          ' PULL THE LED PIN LOW (EXTINGUISH THE LED)
    PAUSE 500            ' WAIT 500MS (HALF A SECOND)
  WEND                   ' CLOSE THE LOOP
```

Remember that you do not need to type in the comments. i.e. the *blue* text following the *'* character.

Once the program is typed into the IDE, press the toolbar's *Compile and Program* button to compile the code and place it into the Amicus18's microcontroller. As long as no typing errors have been made, the LED will then begin to flash. If any errors are found the offending line will be highlighted and an error message will be displayed on the bottom of the IDE.

How to choose the resistor value

A resistor is a device designed to cause resistance to an electric current and therefore cause a drop in voltage across its terminals. If you imagine a resistor to be like a water pipe that is a lot thinner than the pipe connected to it. As the water (the electric current) comes into the resistor, the pipe gets thinner and the current coming out of the other end is therefore reduced. We use resistors to decrease voltage or current to other devices. The value of resistance is known as an Ohm and its symbol is a Greek Omega symbol Ω.

In this case Digital Pin RB0 is outputting 3.3 volts DC at 25mA (milliamps), and our LED requires a voltage of 2v and a current of 20mA. We therefore need to put in a resistor that will reduce the 3.3 volts to 2.2 volts, and the current from 25mA to 20mA if we want to display the LED at its maximum brightness. If we want the LED to be dimmer we could use a higher value of resistance.

To calculate what resistor we need to do this we use what is called "Ohms law" which is $I = V/R$ where I is current, V is voltage and R is resistance. Therefore to work out the resistance we arrange the formula to be $R = V/I$ which is $R = 1.1/0.02$ which is 55 Ohms. V is 1.1 because we need the Voltage Drop, which is the supply voltage (3.3 volts) minus the Forward Voltage (2.2 volts) of the LED (found in the LED datasheet) which is 1.1 volts. We therefore need to find a 55Ω resistor. However, 55Ω resistors are not easily found, so we'll find a one close to it, 47 Ohms will do.

A resistor is too small to put writing onto that could be readable by most people so instead resistors use a colour code. Around the resistor you will typically find 4 coloured bands and by using the colour code in the chart on the next page you can find out the value of a resistor or what colour codes a particular resistance will be.

Colour	1st Band	2nd Band	3rd Band (multiplier)	4th Band (tolerance)
Black	0	0	x100	
Brown	1	1	x101	±1%
Red	2	2	x102	±2%
Orange	3	3	x103	
Yellow	4	4	x104	
Green	5	5	x105	±0.5%
Blue	6	6	x106	±0.25%
Violet	7	7	x107	±0.1%
Grey	8	8	x108	±0.05%
White	9	9	x109	
Gold			x10-1	±5%
Silver			x10-2	±10%
None				±20%

We need a 47Ω resistor, so if we look at the colour table we see that we need 4 in the first band, which is Yellow, followed by a 7 in the next band which is Violet and we then need to multiply this by 100 which is Black in the 3rd band. The final band is irrelevant for our purposes as this is the tolerance. Our resistor has a gold band and therefore has a tolerance of ±5% which means the actual value of the resistor can vary between 46.5Ω and 47.5Ω. We therefore need a resistor with a Yellow, Violet, Black, Gold colour band combination which looks like this:
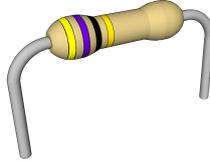
If we needed a 1K (or 1 kilo-ohm) resistor we would need a Brown, Black, Red combination (1, 0, +2 zeros). If we needed a 570K resistor the colours would be Green, Violet and Yellow.

In the same way, if you found a resistor and wanted to know what value it is you would do the same in reverse. So if you found this resistor and wanted to find out what value it was so you could store it away in your nicely labelled resistor storage box, we could look at the table to see it has a value of 220Ω.

The LED

The final component is an LED, which stands for Light Emitting Diode. A Diode is a device that permits current to flow in only one direction. So, it is just like a valve in a water system, but in this case it's letting electrical current to go in one direction, but if the current tried to reverse and go back in the opposite direction the diode would stop it from doing so. Diodes can be useful to prevent accidental connection of a Power supply in a circuit, and damaging the components.

An LED is the same thing, but it also emits light. LEDs come in all kinds of different colours and brightness's and can also emit light in the ultraviolet and infrared part of the spectrum (like in the LEDs within a TV remote control).

If you look carefully at the LED you will notice two things. One is that the legs are of different lengths and also that on one side of the LED, instead of it being cylindrical, it is flattened. These are indicators to show you which leg is the Anode (Positive) and which is the Cathode (Negative). The longer leg gets connected to the Positive Supply (3.3 volts) and the leg with the flattened side goes to Ground (Gnd).

If you connect the LED the wrong way, it will not damage it, but it is essential that you always place a resistor in series with the LED to ensure that the correct current gets to the LED. You can permanently damage the LED if you fail to do this.

As well as single colour LEDs you can also obtain bi-colour and tricolour LEDs. These will have several legs coming out of them with one of them being common (i.e. common anode or common cathode).

Epoxy lens/case
Wire bond
Reflective cavity
Semiconductor die
Anvil } Leadframe
Post }
Flat spot

Anode   Cathode
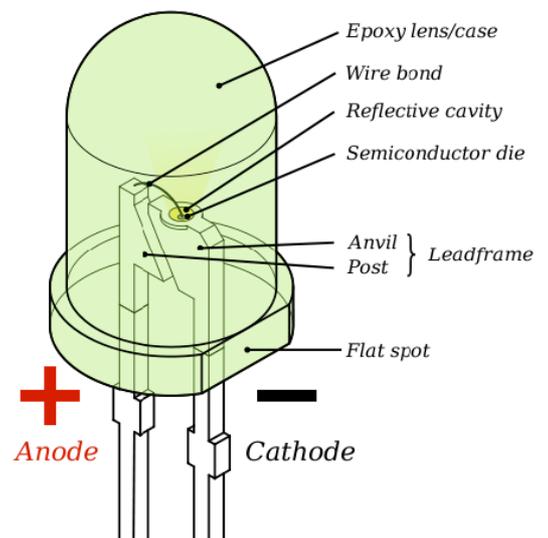
2 LED Flasher

Adding a second LED is simple, and the code for driving them is not too difficult either:

06-10-2009
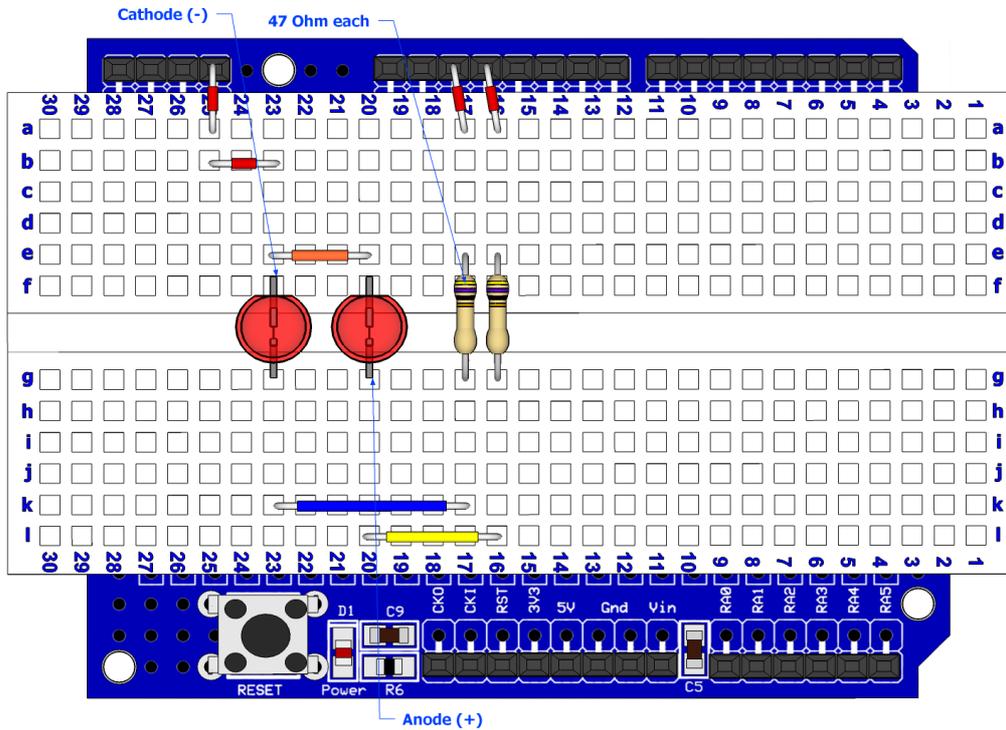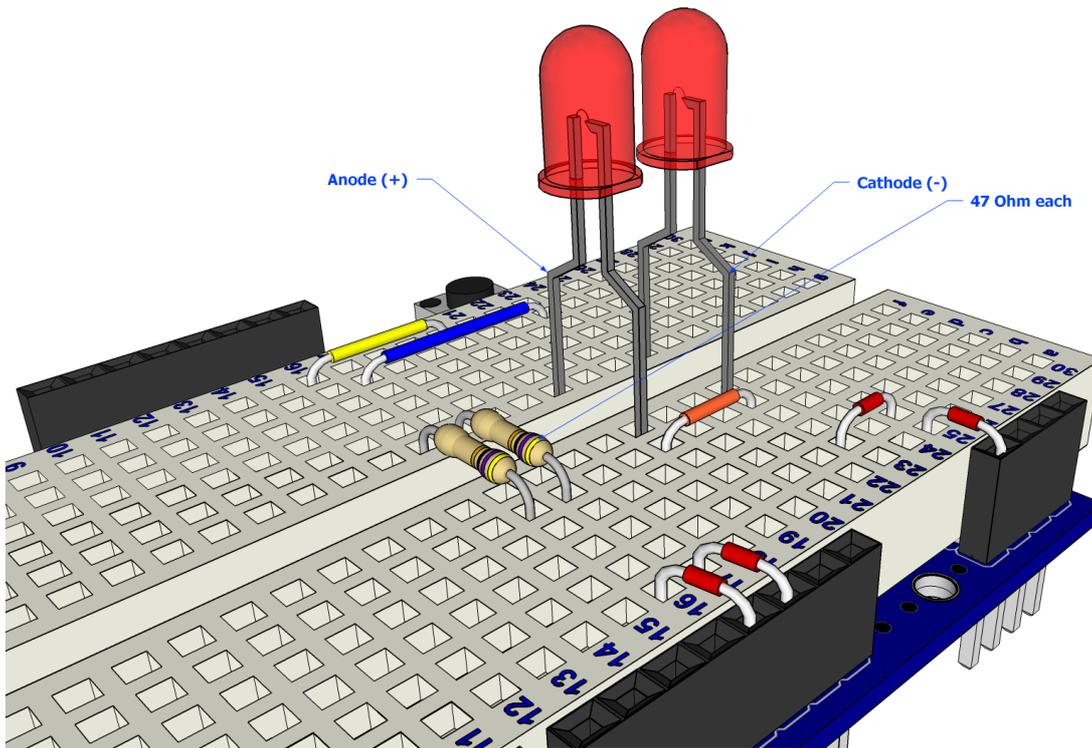
Amicus18 Companion Shield

The circuit for the two LED flasher layout is shown below:
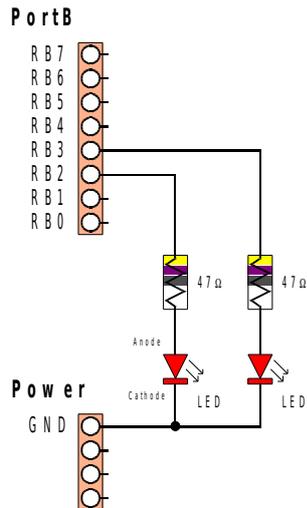
The code for driving the LEDs is shown below:

```
' PIC18F25K20 ON THE AMICUS BOARD
' Flash 2 LEDs connected to RB2 and RB3

    DEFINE OSC 64            'THE 16MHz IS PLLed TO 64 MHz
    DEFINE LOADER_USED 1     'USING THE AMICUS USB BOOT-LOADER

    PORTB = 0               'MAKES PORTB LOW
    TRISB=%00000000         'MAKES PORTB PINS OUTPUTS
    LED1 VAR PORTB.2        'ASSIGNS VAR TO PIN
    LED2 VAR PORTB.3        'ASSIGNS VAR TO PIN

    WHILE 1 = 1             ' Create an infinite loop
     HIGH LED1              ' Illuminate LED1
     PAUSE 500              ' Wait for half a second
     LOW LED1               ' Extinguish LED1
     HIGH LED2              ' Illuminate LED2
     PAUSE 500              ' Wait for half a second
     LOW LED2               ' Extinguish LED2
    WEND                    ' Do it forever
```

13

4 LED Sequencer

Adding, and using, extra LEDs is also very simple, as illustrated below:

Cathode (-)

47 Ohm each

Cathode (-)

47 Ohm each
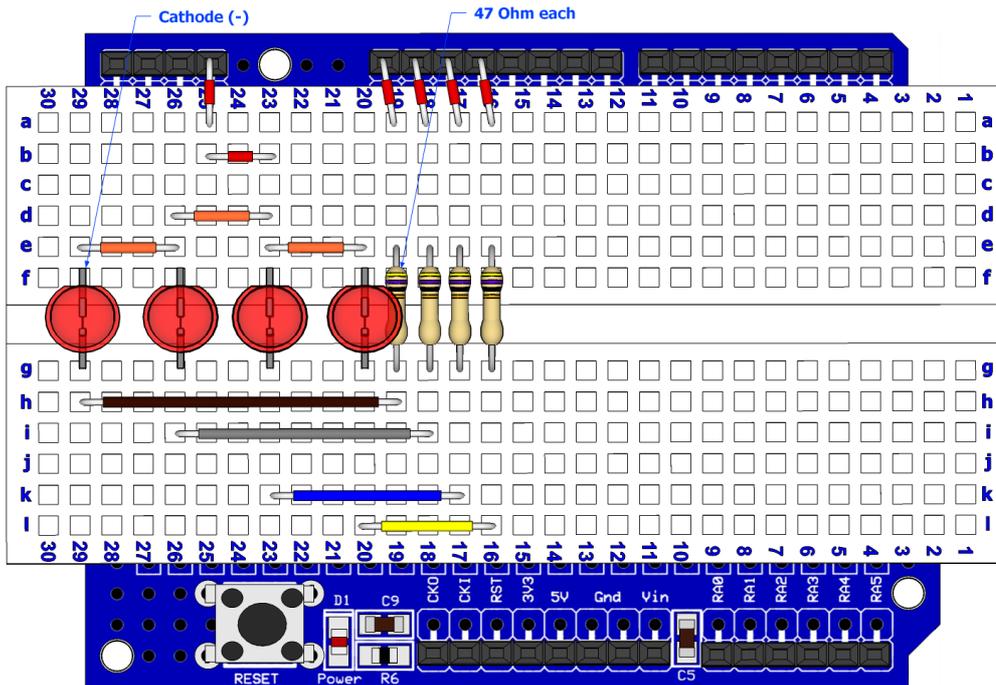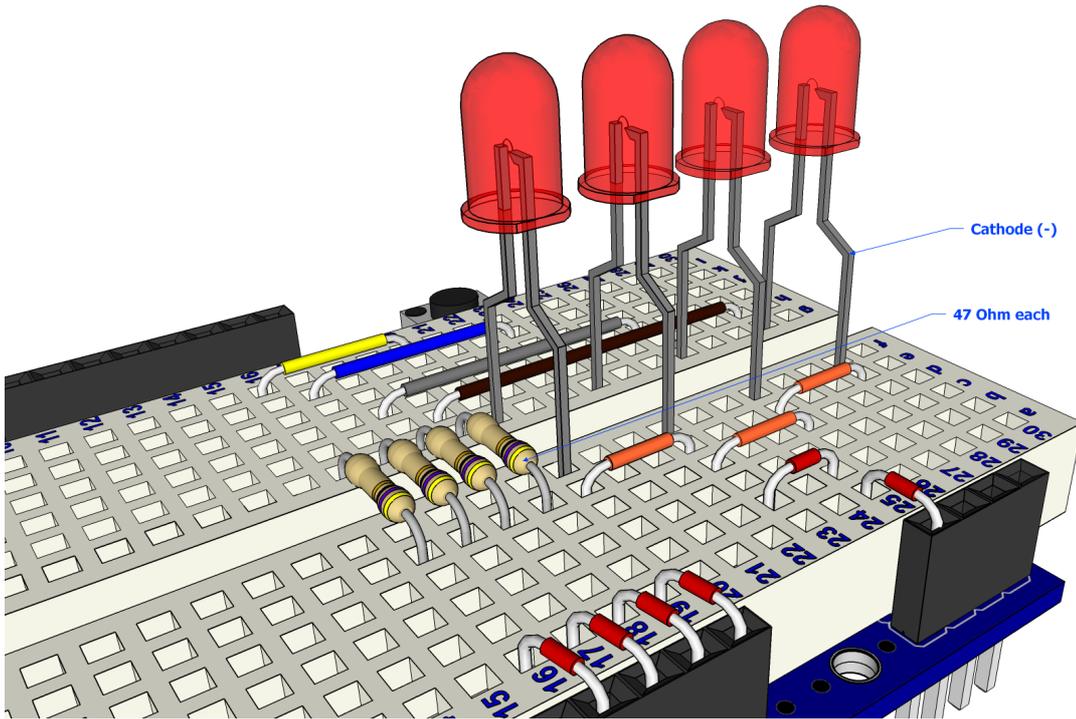
Amicus18 Companion Shield

The two extra LEDs are connected to RB0 and RB1 of PortB, as the circuit shows below:
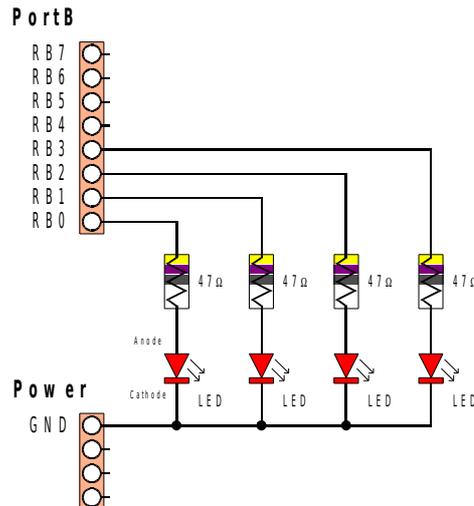
A suitable program for the 4 LED sequencer is shown below:

```
' PIC18F25K20 ON THE AMICUS BOARD
' ILLUMINATE 4 LEDS ATTACHED TO PORTB IN SEQUENCE
' MAKE SURE THE AMICUS18 BOARD'S JUMPER Q3 IS SET TO THE RB1 POSITION

    DEFINE OSC 64           'THE 16MHZ IS PLLED TO 64 MHZ
    DEFINE LOADER_USED 1    'USING THE AMICUS USB BOOT-LOADER
    PORTB = 0               'MAKES PORTB LOW
    TRISB=%00000000         'MAKES PORTB PINS OUTPUTS

    WHILE 1 = 1             ' CREATE AN INFINITE LOOP
     PORTB = %00000001      ' ILLUMINATE THE FIRST LED
     PAUSE 300              ' DELAY A PRE-DETERMINED AMOUNT OF TIME
     PORTB = %00000010      ' ILLUMINATE THE SECOND LED
     PAUSE 300              ' DELAY A PRE-DETERMINED AMOUNT OF TIME
     PORTB = %00000100      ' ILLUMINATE THE THIRD LED
     PAUSE 300              ' DELAY A PRE-DETERMINED AMOUNT OF TIME
     PORTB = %00001000      ' ILLUMINATE THE FOURTH LED
     PAUSE 300              ' DELAY A PRE-DETERMINED AMOUNT OF TIME
    WEND                    ' DO IT FOREVER
```

The above program will illuminate each LED in turn.

Amicus18 Companion Shield

A more advanced program to do the same thing is shown below:

```
' PIC18F25K20 ON THE AMICUS BOARD
' ILLUMINATE 4 LEDS ATTACHED TO PORTB IN SEQUENCE
' USING A MORE ADVANCED METHOD
' MAKE SURE THE AMICUS18 BOARD'S JUMPER Q3 IS SET TO THE RB1 POSITION

    DEFINE OSC 64          'THE 16MHZ IS PLLED TO 64 MHZ
    DEFINE LOADER_USED 1   'USING THE AMICUS USB BOOT-LOADER
    PORTB = 0              'MAKES PORTB LOW
    TRISB=%00000000        'MAKES PORTB PINS OUTPUTS
    BPORTSHADOW VAR BYTE   ' CREATE A VARIABLE TO HOLD THE STATE OF PORTB
    BLOOP VAR BYTE         ' CREATE A VARIABLE FOR THE BIT COUNTING LOOP

    WHILE 1 = 1                         ' CREATE AN INFINITE LOOP
        BPORTSHADOW = 1                 ' SET THE INITIAL STATE OF PORTB
        PORTB = BPORTSHADOW             ' TRANSFER THE SHADOW VARIABLE TO PORTB
        PAUSE 300                       ' WAIT A PRE-DETERMINED AMOUNT OF TIME
    FOR BLOOP = 0 TO 2                  ' CREATE A LOOP FROM 0 TO 2
        BPORTSHADOW = BPORTSHADOW << 1  ' SHIFT A BIT LEFT ONE POSITION
        PORTB = BPORTSHADOW             ' TRANSFER THE SHADOW VARIABLE TO PORTB
        PAUSE 300                       ' WAIT A PRE-DETERMINED AMOUNT OF TIME
    NEXT                                ' CLOSE THE LOOP
    WEND                                ' DO IT FOREVER
```

There are many variations of the programs that can be used with the four LED circuit. The program below sequences the LED's up then down the line.

```
' PIC18F25K20 ON THE AMICUS BOARD
' ILLUMINATE 4 LEDS ATTACHED TO PORTB IN SEQUENCE LEFT TO RIGHT
' USING A MORE ADVANCED METHOD
' MAKE SURE THE AMICUS18 BOARD'S JUMPER Q3 IS SET TO THE RB1 POSITION

    DEFINE OSC 64          'THE 16MHZ IS PLLED TO 64 MHZ
    DEFINE LOADER_USED 1   'USING THE AMICUS USB BOOT-LOADER
    PORTB = 0              'MAKES PORTB LOW
    TRISB=%00000000        'MAKES PORTB PINS OUTPUTS
    BPORTSHADOW VAR BYTE   ' CREATE A VARIABLE TO HOLD THE STATE OF PORTB
    BLOOP VAR BYTE         ' CREATE A VARIABLE FOR THE BIT COUNTING LOOP

    BPORTSHADOW = 1                     ' SET THE INITIAL STATE OF PORTB
    LOW PORTB                           ' MAKE PORTB OUTPUT LOW
    PORTB = BPORTSHADOW                 ' TRANSFER THE SHADOW VARIABLE TO PORTB
    PAUSE 300                           ' WAIT A PRE-DETERMINED AMOUNT OF TIME
    WHILE 1 = 1                         ' CREATE AN INFINITE LOOP
    FOR BLOOP = 0 TO 2                  ' CREATE A LOOP FROM 0 TO 2
        BPORTSHADOW = BPORTSHADOW << 1  ' SHIFT A BIT LEFT ONE POSITION
        PORTB = BPORTSHADOW             ' TRANSFER THE SHADOW VARIABLE TO PORTB
        PAUSE 300                       ' WAIT A PRE-DETERMINED AMOUNT OF TIME
    NEXT                                ' CLOSE THE LOOP
    FOR BLOOP = 2 TO 0 STEP -1          ' CREATE A LOOP FROM 2 TO 0
        BPORTSHADOW = BPORTSHADOW >> 1  ' SHIFT A BIT RIGHT ONE POSITION
        PORTB = BPORTSHADOW             ' TRANSFER THE SHADOW VARIABLE TO PORTB
        PAUSE 300                       ' WAIT A PRE-DETERMINED AMOUNT OF TIME
    NEXT                                ' CLOSE THE LOOP
    WEND                                ' DO IT FOREVER
```

8 LED Sequencer

A more sophisticated layout is shown below, in which eight LEDs are used. Notice how the use of different colour LEDs adds a new twist:
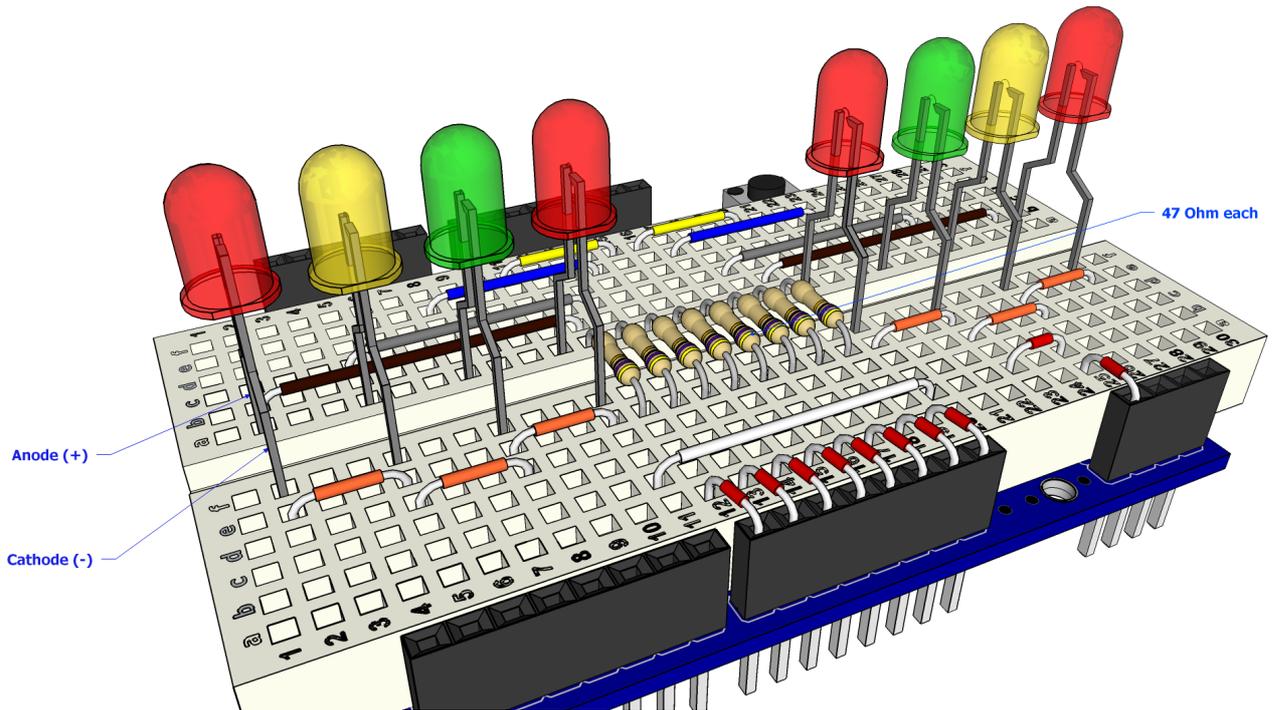

A top down view of the above layout is shown below for extra clarity:
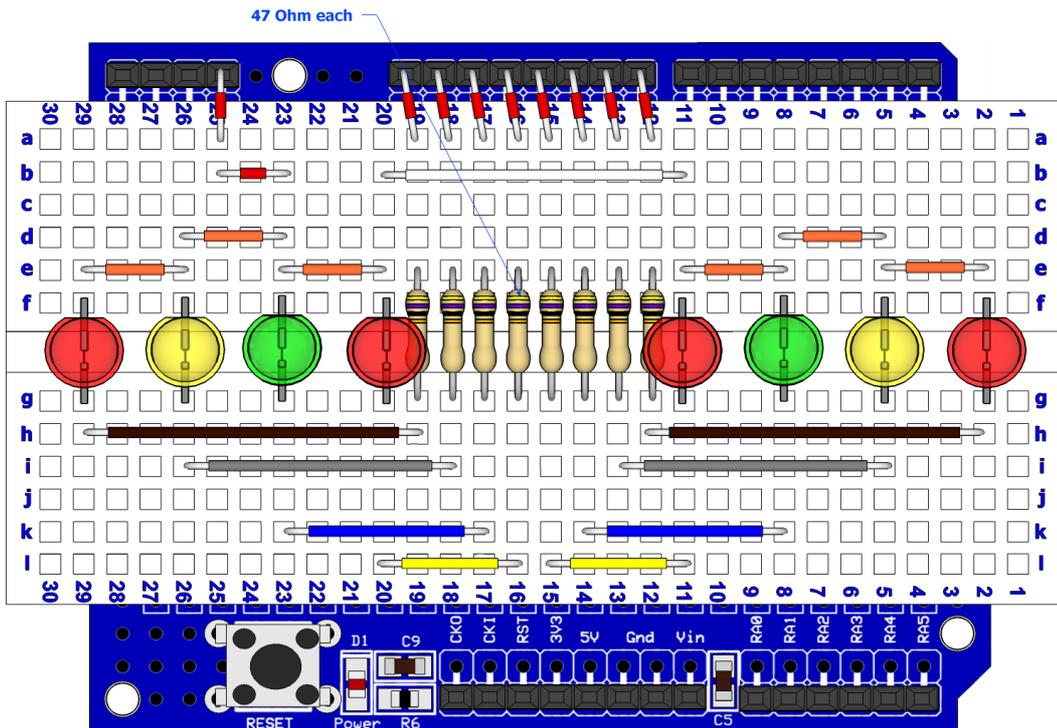
Note. Make sure the Amicus18's *Q3* jumper is set to the **RB1** position.

Amicus18 Companion Shield

The circuit for the eight LED layout is shown below:
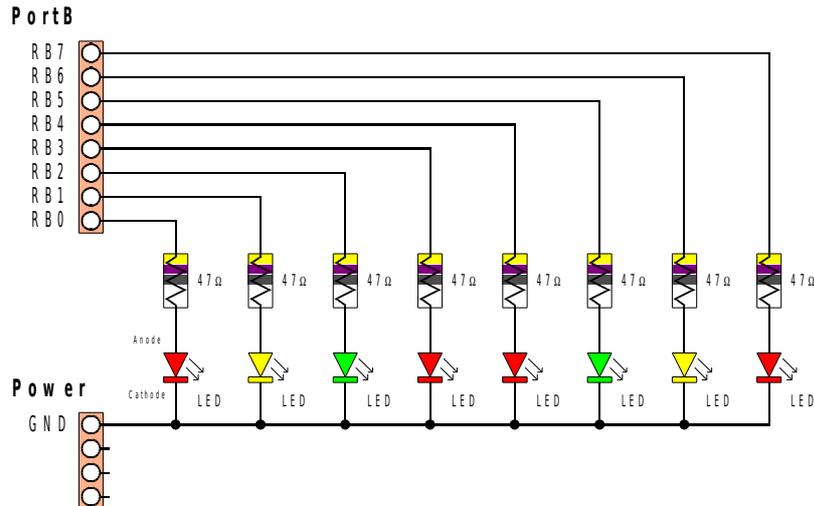
A suitable program for the 8 LED sequencer is shown below:

```
' PIC18F25K20 ON THE AMICUS BOARD
' ILLUMINATE 8 LEDS ATTACHED TO PORTB IN SEQUENCE
' MAKE SURE THE AMICUS18 BOARD'S JUMPER Q3 IS SET TO THE RB1 POSITION

    DEFINE OSC 64          'THE 16MHZ IS PLLED TO 64 MHZ
    DEFINE LOADER_USED 1   'USING THE AMICUS USB BOOT-LOADER
    PORTB = 0              'MAKES PORTB LOW
    TRISB=%00000000        'MAKES PORTB PINS OUTPUTS

    WHILE 1 = 1            ' CREATE AN INFINITE LOOP
      PORTB = %00000001    ' ILLUMINATE THE FIRST LED
      PAUSE 300            ' DELAY A PRE-DETERMINED AMOUNT OF TIME
      PORTB = %00000010    ' ILLUMINATE THE SECOND LED
      PAUSE 300            ' DELAY A PRE-DETERMINED AMOUNT OF TIME
      PORTB = %00000100    ' ILLUMINATE THE THIRD LED
      PAUSE 300            ' DELAY A PRE-DETERMINED AMOUNT OF TIME
      PORTB = %00001000    ' ILLUMINATE THE FOURTH LED
      PAUSE 300            ' DELAY A PRE-DETERMINED AMOUNT OF TIME
      PORTB = %00010000    ' ILLUMINATE THE FIFTH LED
      PAUSE 300            ' DELAY A PRE-DETERMINED AMOUNT OF TIME
      PORTB = %00100000    ' ILLUMINATE THE SIXTH LED
      PAUSE 300            ' DELAY A PRE-DETERMINED AMOUNT OF TIME
      PORTB = %01000000    ' ILLUMINATE THE SEVENTH LED
      PAUSE 300            ' DELAY A PRE-DETERMINED AMOUNT OF TIME
      PORTB = %10000000    ' ILLUMINATE THE EIGHTH LED
      PAUSE 300            ' DELAY A PRE-DETERMINED AMOUNT OF TIME
    WEND                   ' DO IT FOREVER
```

The above program will illuminate each LED in turn.

06-10-2009

A more advanced program to do the same thing is shown below:

```
' PIC18F25K20 ON THE AMICUS BOARD
' ILLUMINATE 8 LEDS ATTACHED TO PORTB IN SEQUENCE
' USING A MORE ADVANCED METHOD
' MAKE SURE THE AMICUS18 BOARD'S JUMPER Q3 IS SET TO THE RB1 POSITION

    DEFINE OSC 64            'THE 16MHZ IS PLLED TO 64 MHZ
    DEFINE LOADER_USED 1     'USING THE AMICUS USB BOOT-LOADER
    PORTB = 0                'MAKES PORTB LOW
    TRISB=%00000000          'MAKES PORTB PINS OUTPUTS
    BPORTSHADOW VAR BYTE     ' CREATE A VARIABLE TO HOLD THE STATE OF PORTB
    BLOOP VAR BYTE           ' CREATE A VARIABLE FOR THE BIT COUNTING LOOP

    WHILE 1 = 1                        ' CREATE AN INFINITE LOOP
        BPORTSHADOW = 1                ' SET THE INITIAL STATE OF PORTB
        PORTB = BPORTSHADOW            ' TRANSFER THE SHADOW VARIABLE TO PORTB
        PAUSE 300                      ' WAIT A PRE-DETERMINED AMOUNT OF TIME
    FOR BLOOP = 0 TO 6                 ' CREATE A LOOP FROM 0 TO 6
        BPORTSHADOW = BPORTSHADOW << 1 ' SHIFT A BIT LEFT ONE POSITION
        PORTB = BPORTSHADOW            ' TRANSFER THE SHADOW VARIABLE TO PORTB
        PAUSE 300                      ' WAIT A PRE-DETERMINED AMOUNT OF TIME
    NEXT                               ' CLOSE THE LOOP
    WEND                               ' DO IT FOREVER
```

There are many variations of the programs that can be used with the eight LED circuit. The program below sequences the LED's up then down the line.

```
' PIC18F25K20 ON THE AMICUS BOARD
' ILLUMINATE 4 LEDS ATTACHED TO PORTB IN SEQUENCE LEFT TO RIGHT
' USING A MORE ADVANCED METHOD
' MAKE SURE THE AMICUS18 BOARD'S JUMPER Q3 IS SET TO THE RB1 POSITION

    DEFINE OSC 64            'THE 16MHZ IS PLLED TO 64 MHZ
    DEFINE LOADER_USED 1     'USING THE AMICUS USB BOOT-LOADER
    PORTB = 0                'MAKES PORTB LOW
    TRISB=%00000000          'MAKES PORTB PINS OUTPUTS
    BPORTSHADOW VAR BYTE     ' CREATE A VARIABLE TO HOLD THE STATE OF PORTB
    BLOOP VAR BYTE           ' CREATE A VARIABLE FOR THE BIT COUNTING LOOP

    BPORTSHADOW = 1                    ' SET THE INITIAL STATE OF PORTB
    LOW PORTB                          ' MAKE PORTB OUTPUT LOW
    PORTB = BPORTSHADOW                ' TRANSFER THE SHADOW VARIABLE TO PORTB
    PAUSE 300                          ' WAIT A PRE-DETERMINED AMOUNT OF TIME
    WHILE 1 = 1                        ' CREATE AN INFINITE LOOP
    FOR BLOOP = 0 TO 6                 ' CREATE A LOOP FROM 0 TO 6
        BPORTSHADOW = BPORTSHADOW << 1 ' SHIFT A BIT LEFT ONE POSITION
        PORTB = BPORTSHADOW            ' TRANSFER THE SHADOW VARIABLE TO PORTB
        PAUSE 300                      ' WAIT A PRE-DETERMINED AMOUNT OF TIME
    NEXT                               ' CLOSE THE LOOP
    FOR BLOOP = 6 TO 0 STEP -1         ' CREATE A LOOP FROM 6 TO 0
        BPORTSHADOW = BPORTSHADOW >> 1 ' SHIFT A BIT RIGHT ONE POSITION
        PORTB = BPORTSHADOW            ' TRANSFER THE SHADOW VARIABLE TO PORTB
        PAUSE 300                      ' WAIT A PRE-DETERMINED AMOUNT OF TIME
    NEXT                               ' CLOSE THE LOOP
    WEND                               ' DO IT FOREVER
```

Amicus18 Companion Shield

Traffic Light Sequencer

Using an adaptation of the 8 multi-coloured LED layout, we can create the sequence for a UK traffic light. The layout is shown below, notice that the only difference is the removal of four LEDs and four resistors:


20
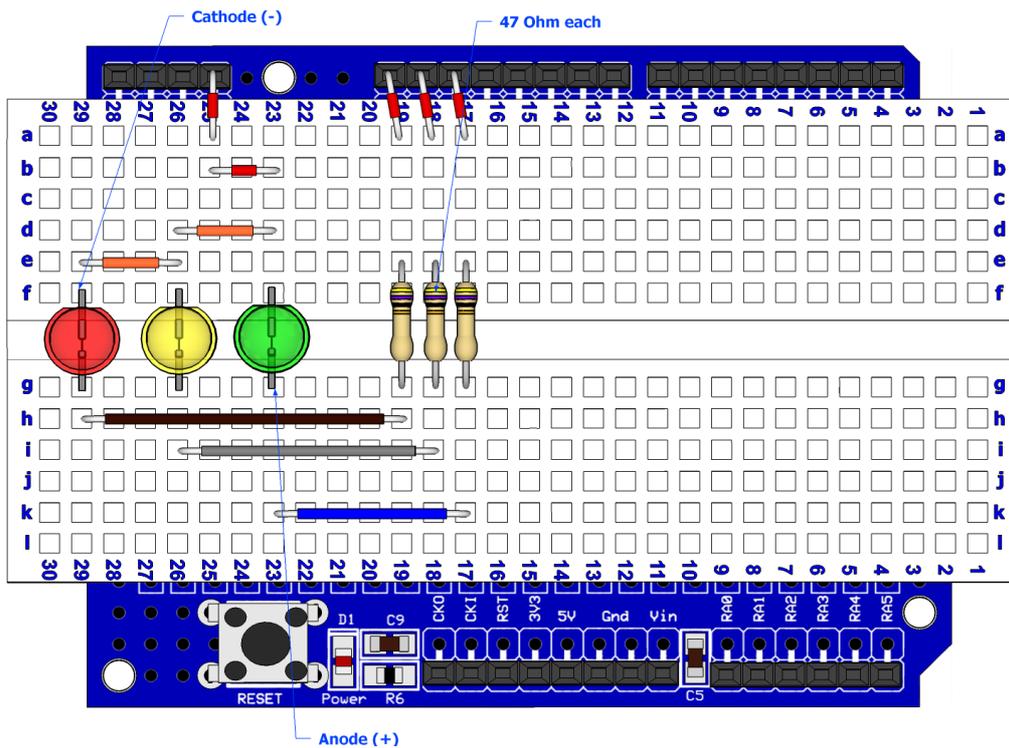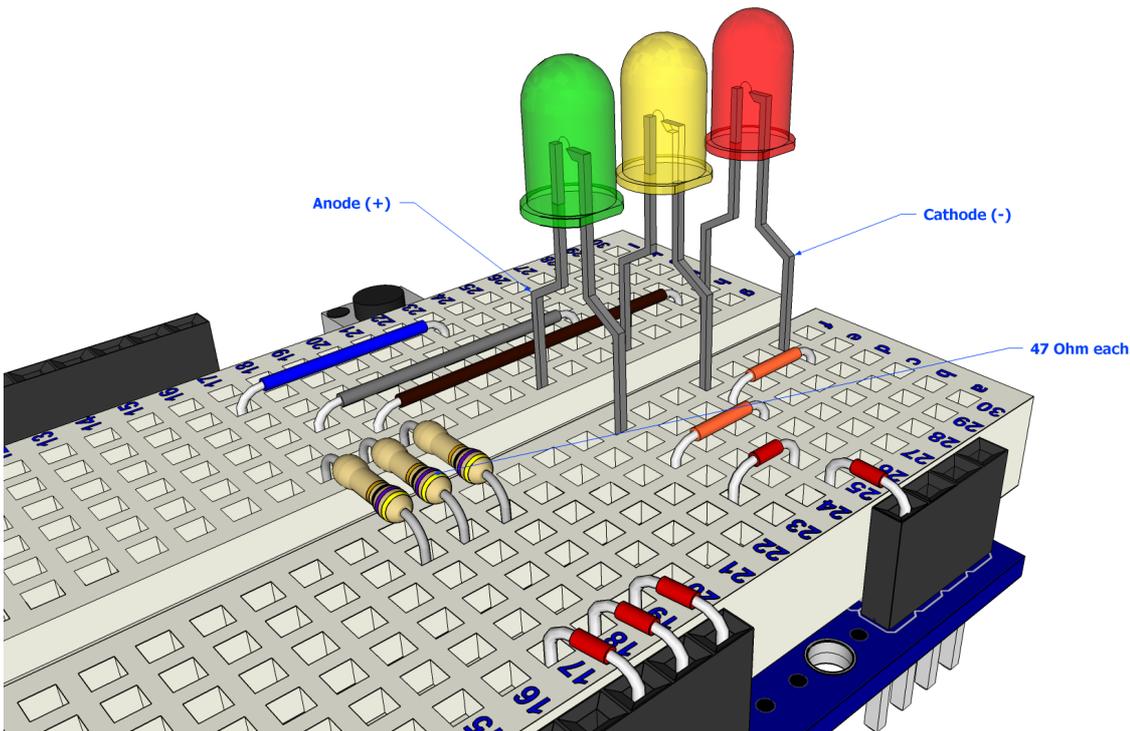
The circuit for the traffic light sequencer is shown below:
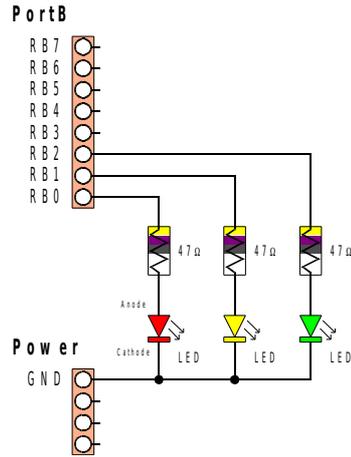
The sequence of traffic lights in the UK is shown below:
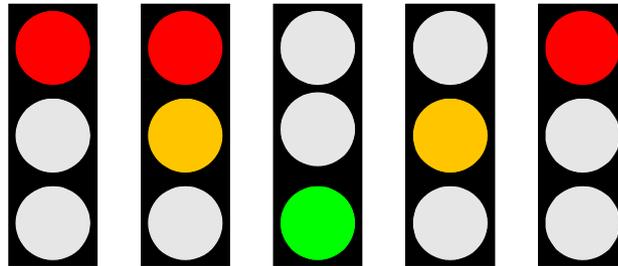
Amicus18 Companion Shield

The program below shows the steps required to reproduce the sequence of lights shown above:

```
' PIC18F25K20 ON THE AMICUS BOARD
' SIMULATE A SINGLE TRAFFIC LIGHT USING RED, YELLOW, AND GREEN LEDS

    DEFINE OSC 64          'THE 16MHZ IS PLLED TO 64 MHZ
    DEFINE LOADER_USED 1   'USING THE AMICUS USB BOOT-LOADER
    PORTB = 0              'MAKES PORTB LOW
    TRISB=%00000000        'MAKES PORTB PINS OUTPUTS

    RED   VAR PORTB.0           ' RED LED IS ATTACHED TO RB0
    AMBER VAR PORTB.1           ' AMBER LED IS ATTACHED TO RB1
    GREEN VAR PORTB.2           ' GREEN LED IS ATTACHED TO RB2
    REDINTERVAL CON 4000       ' TIME THAT THE RED LIGHT WILL STAY ON
' TIME THAT THE RED AND AMBER LIGHTS WILL STAY ON
    AMBERREDINTERVAL CON REDINTERVAL / 4
' TIME THAT THE AMBER LIGHT WILL STAY ON
    AMBERINTERVAL CON REDINTERVAL - AMBERREDINTERVAL
    GREENINTERVAL CON 6000     ' TIME THAT THE GREEN LIGHT WILL STAY ON

    WHILE 1 = 1                    ' CREATE AN INFINITE LOOP
        HIGH RED                   ' ILLUMINATE THE RED LED
        PAUSE REDINTERVAL          ' WAIT FOR THE APPROPRIATE LENGTH OF TIME
        HIGH AMBER                 ' ILLUMINATE THE AMBER LED
        PAUSE AMBERREDINTERVAL     ' WAIT FOR THE APPROPRIATE LENGTH OF TIME
        LOW RED                    ' EXTINGUISH THE RED LED
        PAUSE AMBERINTERVAL        ' WAIT FOR THE APPROPRIATE LENGTH OF TIME
        HIGH GREEN                 ' ILLUMINATE THE GREEN LED
        LOW AMBER                  ' EXTINGUISH THE AMBER LED
        PAUSE GREENINTERVAL        ' WAIT FOR THE APPROPRIATE LENGTH OF TIME
        LOW GREEN                  ' EXTINGUISH THE GREEN LED
        HIGH AMBER                 ' ILLUMINATE THE AMBER LED
        PAUSE AMBERINTERVAL        ' WAIT FOR THE APPROPRIATE LENGTH OF TIME
        LOW AMBER                  ' EXTINGUISH THE AMBER LED
    WEND                           ' DO IT FOREVER
```

Type in the program above, remembering that you do not need to type in the comments. Click on the toolbar *Compiler and Program* button or press **F10** to compile the code and load it into the Amicus18's microcontroller. The three LEDs will then start sequencing.

Sensing the Outside World

Interacting with the outside world is always desirable when using a microcontroller, whether it's choosing a drink in a vending machine or deciding which way a pacman will move. The easiest method of outside influence is through the use of a switch or button.

However, there are certain rules that must be observed when adding a switch to a microcontroller's pin. When the pin is configured as an input, it can be brought high to 3.3 Volts or pulled low to ground, however if neither of these states is performed, the pin is neither high or low and this is termed *floating*. Even if a switch was placed from the microcontroller's input pin to ground, when the switch is not being operated the input pin can be high or low (*floating*).

What's required is a pull-up resistor or a pull-down resistor in order to force a single state when not in use. A pull-up resistor is a weak resistance from the input pin to the 3.3 Volt line, while a pull-down resistor is a weak resistance from the input pin to ground.

Switch Input (Pulled-Up)

The layout below shows a pull-up resistance:

Amicus18 Companion Shield

The circuit for the pulled-up switch input is shown below:
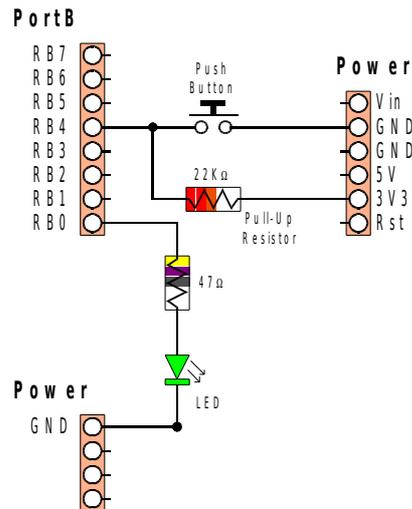
Open the Amicus IDE and type in the following program, or copy and paste from here:

```
' PIC18F25K20 ON THE AMICUS BOARD
' DEMOSTRATE A SWITCH INPUT USING A PULL-UP RESISTOR
' DISPLAY STATE OF THE INPUT PIN WHEN A PUSH-BUTTON IS OPERATED VIA SERIAL PORT.
' THE SERIAL PORT IS A VIRTUAL USB PORT CONNECTED TO PORTC.6.
' CHECK THE PORT NUMBER THE OPERATING SYSTEM ASSIGNS.

    DEFINE OSC 64           'THE 16MHz IS PLLed TO 64 MHz
    DEFINE LOADER_USED 1    'USING THE AMICUS USB BOOT-LOADER
    DEFINE DEBUG_REG PORTC  'SET DEBUG PIN PORT
    DEFINE DEBUG_BIT 6      'SET DEBUG PIN BIT
    DEFINE DEBUG_BAUD 9600  'SET DEBUG BAUD RATE
    DEFINE DEBUG_MODE 0     'SET DEBUG MODE: 0 = TRUE, 1 = INVERTED

    SWITCH  VAR PORTB.4      ' ASSIGNS VAR TO PIN (RB4)
    INPUT   SWITCH           ' SETS PIN AS INPUT

    WHILE 1 = 1                             ' CREATE AN INFINITE LOOP
       DEBUG "BUTTON = ", BIN1 SWITCH,13    ' DISPLAY THE INPUT STATE
       PAUSE 500                            ' DELAY FOR HALF A SECOND
    WEND                                    ' DO IT FOREVER
```

Click the toolbar icon *Compile and Program* or press **F10** to build the code and place it into the Amicus18's microcontroller.

Open the Serial Terminal by clicking on the toolbar, and open a connection to the Amicus18. Use the default baud of 9600. The serial terminal's window should show the text "Button = 1". This is displaying the state of the pin where the button is attached. Press the button and the test will change to "Button = 0":

06-10-2009
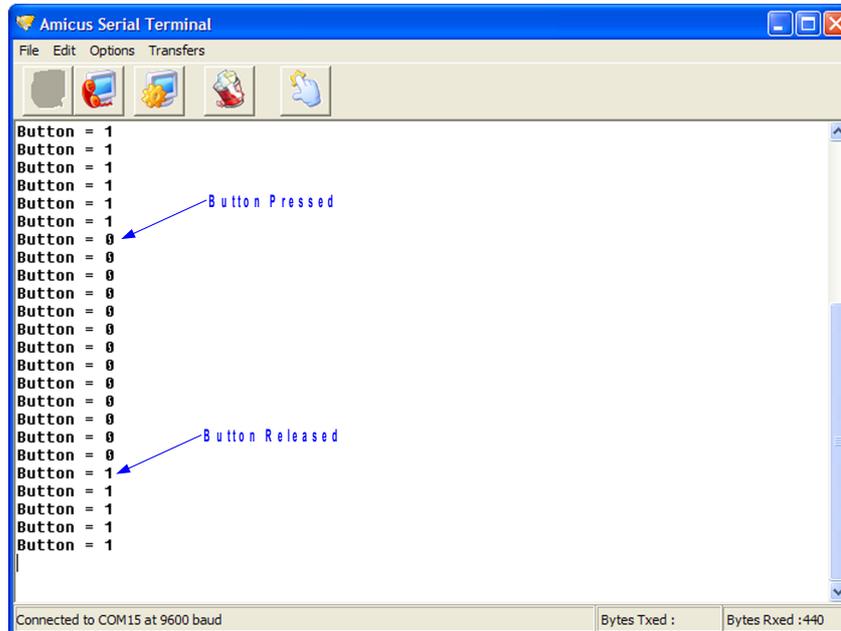
Notice how the state of the pin is 0 when the button is pressed. This is because the weak pull-up resistor (22KΩ) holds the pin to 3.3 Volts when it's not being operated, and the button pulls the pin to ground when it's operated.

Now that we know that the pin's state is 0 when the button is operated, decisions can be made upon it.

The program below will flash the LED 10 times when the button is pressed:

```
' PIC18F25K20 ON THE AMICUS BOARD
' DEMOSTRATE A SWITCH INPUT USING A PULL-UP RESISTOR
' FLASH AN LED BASED UPON A BUTTON PRESS

    DEFINE OSC 64           'THE 16MHz IS PLLed TO 64 MHz
    DEFINE LOADER_USED 1    'USING THE AMICUS USB BOOT-LOADER

    SWITCH  VAR PORTB.4     ' ASSIGNS VAR TO PIN (SWITCH = RB4)
    LED     VAR PORTB.0     ' ASSIGNS VAR TO PIN (LED = RB0)
    FLASH   VAR BYTE        ' HOLDS THE AMOUNT OF FLASHES

    GOTO MAIN               ' JUMP TO SUBROUTINE
'------------------------
' SUBROUTINE TO FLASH THE LED
    FLASH_LED:
        FOR FLASH = 0 TO 9          ' CREATE A TEN COUNT LOOP
            HIGH LED                ' ILLUMINATE THE LED
            PAUSE 100               ' DELAY FOR 100 MILLISECONDS
            LOW LED                 ' EXTINGUISH THE LED
            PAUSE 100               ' DELAY FOR 100 MILLISECOND
        NEXT                        ' CLOSE THE LOOP
        RETURN                      ' EXIT THE SUBROUTINE
' RETURN SENDS THE PROGRAM BACK TO THE LINE AFTER GOSUB
'------------------------
    MAIN:
        INPUT SWITCH                ' SETS PIN AS INPUT
        WHILE 1 = 1                 ' CREATE AN INFINITE LOOP
            IF SWITCH = 0 THEN      'IS THE BUTTON PRESSED ?
                GOSUB FLASH_LED     'IF YES, FLASH THE LED
            ENDIF
        WEND                        ' DO IT FOREVER
```

Switch Input (Pulled-Down)

The layout below shows a pull-down resistance:

Cathode (-)

47 Ohm

22K Ohm

47 Ohm

22K Ohm

Cathode (-)

06-10-2009
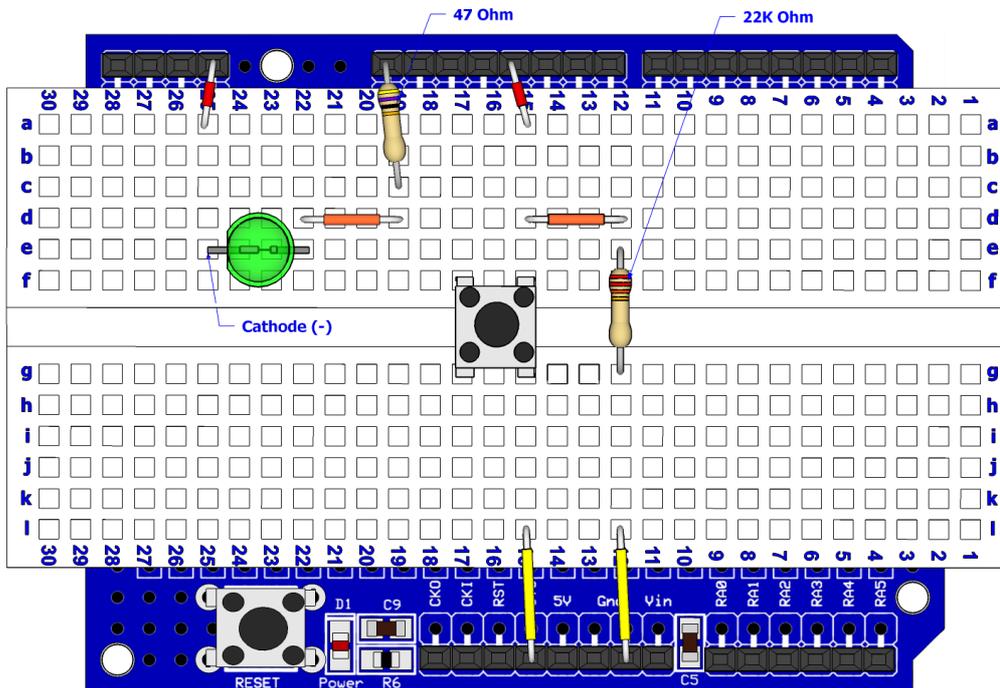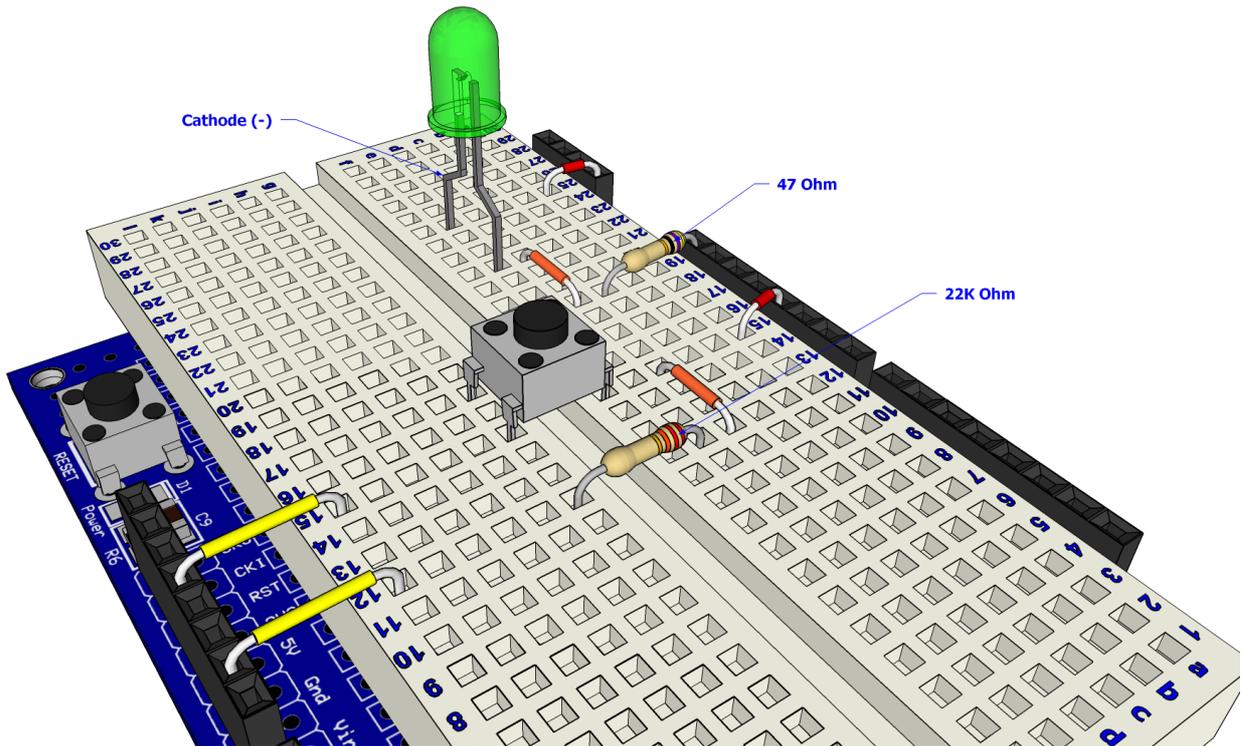
Amicus18 Companion Shield

The circuit for the above layout is shown below:
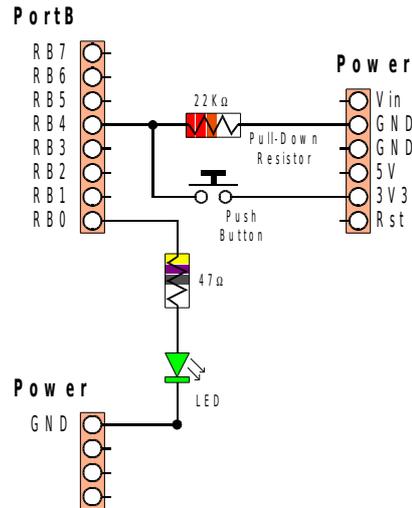
Open the Amicus IDE and type in the following program, or copy and paste from here:

```
' PIC18F25K20 ON THE AMICUS BOARD
' DEMOSTRATE A SWITCH INPUT USING A PULL-DOWN RESISTOR
' DISPLAY STATE OF THE INPUT PIN WHEN A PUSH-BUTTON IS OPERATED VIA SERIAL PORT.
' THE SERIAL PORT IS A VIRTUAL USB PORT CONNECTED TO PORTC.6.
' CHECK THE PORT NUMBER THE OPERATING SYSTEM ASSIGNS.

    DEFINE OSC 64            'THE 16MHz IS PLLed TO 64 MHz
    DEFINE LOADER_USED 1     'USING THE AMICUS USB BOOT-LOADER
    DEFINE DEBUG_REG PORTC   'SET DEBUG PIN PORT
    DEFINE DEBUG_BIT 6       'SET DEBUG PIN BIT
    DEFINE DEBUG_BAUD 9600   'SET DEBUG BAUD RATE
    DEFINE DEBUG_MODE 0      'SET DEBUG MODE: 0 = TRUE, 1 = INVERTED

    SWITCH  VAR PORTB.4      ' ASSIGNS VAR TO PIN (RB4)
    INPUT   SWITCH           ' SETS PIN AS INPUT

    WHILE 1 = 1                              ' CREATE AN INFINITE LOOP
        DEBUG "BUTTON = ", BIN1 SWITCH,13    ' DISPLAY THE INPUT STATE
        PAUSE 500                            ' DELAY FOR HALF A SECOND
    WEND                                     ' DO IT FOREVER
```

Click the toolbar icon *Compile and Program* or press **F10** to build the code and place it into the Amicus18's microcontroller.

Open the Serial Terminal by clicking on the toolbar, and open a connection to the Amicus18. Use the default baud of 9600. The serial terminal's window should show the text "Button = 0". This is displaying the state of the pin where the button is attached. Press the button and the test will change to "Button = 1":

06-10-2009
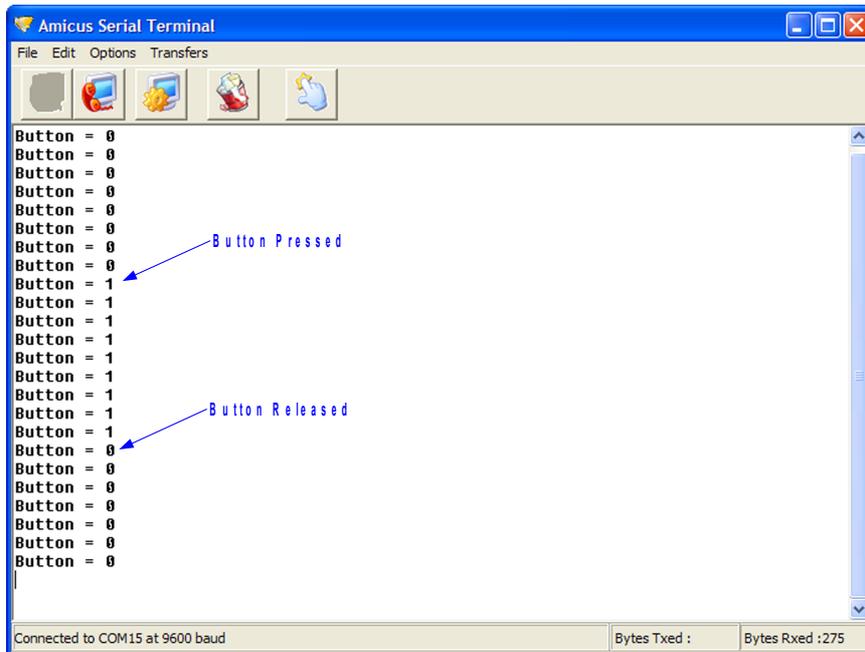
Notice how the state of the pin is 1 when the button is pressed. This is because the weak pull-up resistor (22KΩ) holds the pin to ground when it's not being operated, and the button pulls the pin to 3.3 Volts when it's operated. This is the exact opposite of using a pull-up resistor.

Now that we know that the pin's state is 1 (high) when the button is operated, decisions can be made upon it.

The program below will flash the LED 10 times when the button is pressed:

```
' PIC18F25K20 ON THE AMICUS BOARD
' DEMOSTRATE A SWITCH INPUT USING A PULL-DOWN RESISTOR
' FLASH AN LED BASED UPON A BUTTON PRESS

    DEFINE OSC 64          'THE 16MHz IS PLLed TO 64 MHz
    DEFINE LOADER_USED 1   'USING THE AMICUS USB BOOT-LOADER

    SWITCH   VAR PORTB.4     ' ASSIGNS VAR TO PIN (SWITCH = RB4)
    LED      VAR PORTB.0     ' ASSIGNS VAR TO PIN (LED = RB0)
    FLASH    VAR BYTE        ' HOLDS THE AMOUNT OF FLASHES

    GOTO MAIN               ' JUMP TO SUBROUTINE
'------------------------
' SUBROUTINE TO FLASH THE LED
    FLASH_LED:
        FOR FLASH = 0 TO 9         ' CREATE A TEN COUNT LOOP
            HIGH LED               ' ILLUMINATE THE LED
            PAUSE 100              ' DELAY FOR 100 MILLISECONDS
            LOW LED                ' EXTINGUISH THE LED
            PAUSE 100              ' DELAY FOR 100 MILLISECOND
        NEXT                       ' CLOSE THE LOOP
        RETURN                     ' EXIT THE SUBROUTINE
' RETURN SENDS THE PROGRAM BACK TO THE LINE AFTER GOSUB
'------------------------
    MAIN:
        INPUT SWITCH           ' SETS PIN AS INPUT
        WHILE 1 = 1            ' CREATE AN INFINITE LOOP
            IF SWITCH = 1 THEN     'IS THE BUTTON PRESSED ?
                GOSUB FLASH_LED  'IF YES, FLASH THE LED
            ENDIF
        WEND                       ' DO IT FOREVER
```

06-10-2009

Switch Debounce

Mechanical switches are frequently encountered in embedded processor applications, and are inexpensive, simple, and reliable. However, such switches are also often very electrically noisy. This noise is known as switch bounce, whereby the connection between the switch contacts makes and breaks several, perhaps even hundreds, of times before settling to the final switch state. This can cause a single switch push to be detected as several distinct switch pushes by the fast microcontroller used in the Amicus18 board, especially with an edge-sensitive input. Think of advancing the TV channel, but instead of getting the next channel, the selection skips ahead two or three.

Classic solutions to switch bounce involved low pass filtering out of the fast switch bounce transitions with a resistor-capacitor circuit, or using re-settable logic shift registers. While effective, these methods add additional cost and increase circuit board complexity. Debouncing a switch in software eliminates these issues.

A simple way to debounce a switch is to sample the switch until the signal is stable. How long to sample requires some investigation of the switch characteristics, but usually 5ms is sufficiently long.

The following code demonstrates sampling the switch input every 1mS, waiting for 5 consecutive samples of the same value before determining that the switch was pressed. Note that the tactile switches used for the layouts don't bounce much, but it is good practice to debounce all system switches.

```
' PIC18F25K20 ON THE AMICUS BOARD
' DEBOUNCE A SWITCH INPUT USING A PULL-UP RESISTOR

    DEFINE OSC 64              ' THE 16MHz IS PLLed TO 64 MHz
    DEFINE LOADER_USED 1       ' USING THE AMICUS USB BOOT-LOADER

    SWITCH  VAR PORTB.4         ' PIN WHERE SWITCH IS CONNECTED
    LED     VAR PORTB.0         ' PIN WHERE THE LED IS CONNECTED
    SWITCH_COUNT    VAR BYTE    ' HOLDS THE SWITCH COUNTER AMOUNT
    DETECTS_IN_A_ROW    CON 5 ' THE AMOUNT OF COUNTS TO PERFORM

    MAIN:
    LOW LED                         ' EXTINGUISH THE LED
    INPUT SWITCH                    ' MAKE THE SWITCH AN INPUT
    WHILE 1 = 1                     ' CREATE AN INFINITE LOOP
       WHILE SWITCH <> 1: WEND   ' WAIT FOR SWITCH TO BE RELEASED (PULLED UP)
       SWITCH_COUNT = 5             ' START AT 5, FIRST LOW ROLLS OVER TO 0
       DO                           ' MONITOR SWITCH FOR 5 LOWS IN A ROW TO DEBOUNCE
           IF SWITCH == 0 THEN   ' BUTTON PRESSED ?
               SWITCH_COUNT = SWITCH_COUNT + 1     ' YES? INCREMENT THE COUNTER
           ELSE                      ' IF NOT
               SWITCH_COUNT = 0 ' RESET THE COUNTER
           ENDIF
           PAUSE 1                 ' DELAY 1 MILLISECOND
       LOOP UNTIL SWITCH_COUNT >= DETECTS_IN_A_ROW ' EXIT IF 5 LOWS ARE DETECTED
       TOGGLE LED                  ' TOGGLE LED ON/OFF
    WEND                            ' DO IT FOREVER
```

Amicus18 Companion Shield

The same layout as the pulled-up switch demonstration can be used:
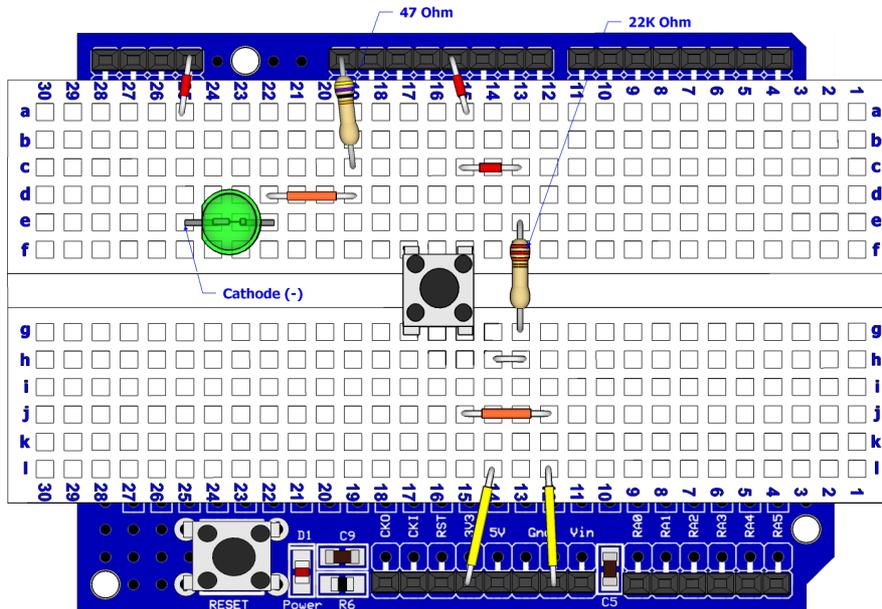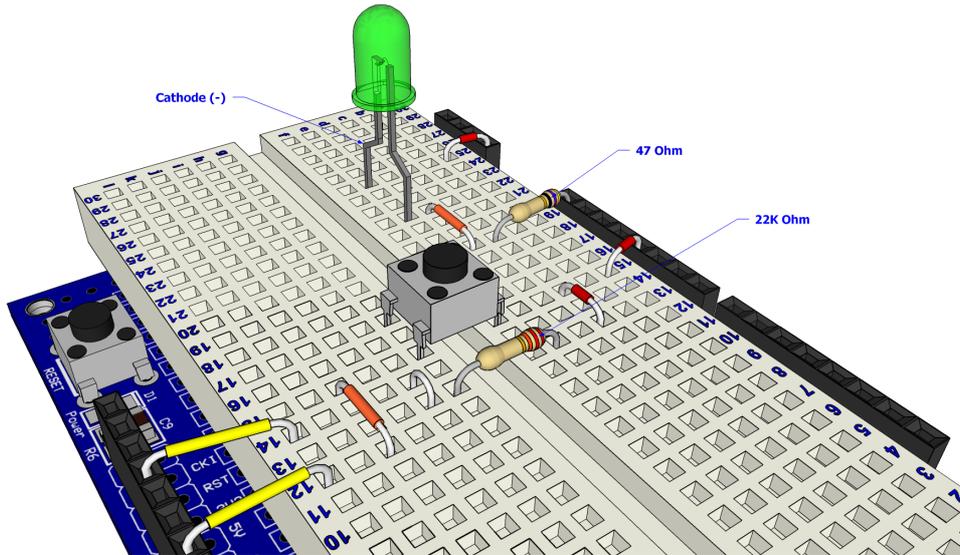
The
circuit for the debounced pulled-up switch input is shown below:

06-10-2009
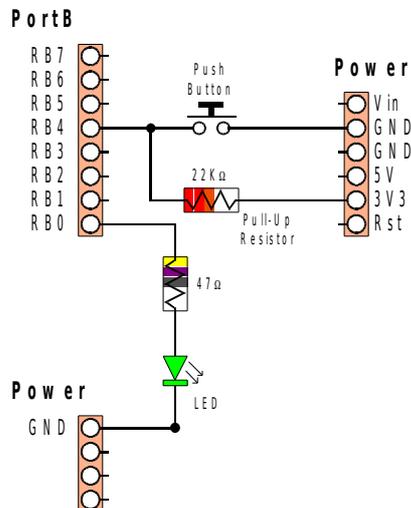
Amicus18 Companion Shield

In order to detect and debounce a switch that is pulled down to ground through a resistor, the following code can be used. It's essentially the same program as the pulled up version, but references to 0 now reference 1, and vice-versa:

```
' PIC18F25K20 ON THE AMICUS BOARD
' DEBOUNCE A SWITCH INPUT USING A PULL-DOWN RESISTOR

    DEFINE OSC 64                ' THE 16MHz IS PLLed TO 64 MHz
    DEFINE LOADER_USED 1         ' USING THE AMICUS USB BOOT-LOADER

    SWITCH  VAR PORTB.4          ' PIN WHERE SWITCH IS CONNECTED
    LED     VAR PORTB.0          ' PIN WHERE THE LED IS CONNECTED
    SWITCH_COUNT     VAR BYTE    ' HOLDS THE SWITCH COUNTER AMOUNT
    DETECTS_IN_A_ROW     CON 5   ' THE AMOUNT OF COUNTS TO PERFORM

    MAIN:
    LOW LED                      ' EXTINGUISH THE LED
    INPUT SWITCH                 ' MAKE THE SWITCH AN INPUT
    WHILE 1 = 1                  ' CREATE AN INFINITE LOOP
        WHILE SWITCH <> 0: WEND  ' WAIT FOR SWITCH TO BE RELEASED (PULLED DOWN)
        SWITCH_COUNT = 5         ' START AT 5, FIRST LOW ROLLS OVER TO 0
        DO                       ' MONITOR SWITCH FOR 5 LOWS IN A ROW TO DEBOUNCE
            IF SWITCH == 1 THEN  ' BUTTON PRESSED ?
                SWITCH_COUNT = SWITCH_COUNT + 1     ' YES? INCREMENT THE COUNTER
            ELSE                 ' IF NOT
                SWITCH_COUNT = 0 ' RESET THE COUNTER
            ENDIF
            PAUSE 1              ' DELAY 1 MILLISECOND
        LOOP UNTIL SWITCH_COUNT >= DETECTS_IN_A_ROW ' EXIT IF 5 LOWS ARE DETECTED
        TOGGLE LED               ' TOGGLE LED ON/OFF
    WEND                         ' DO IT FOREVER
```

Analogue Meets Digital

Not everything in the microcontroller world is made up of ons or offs, sometimes the input required is of an analogue nature i.e. a voltage. This is where an Analogue to Digital Converter (ADC) comes into it's own. An ADC samples the incoming voltage and converts it to a binary representation. The Amicus18 has nine ADC inputs, each capable of producing a 10-bit sample (0 to 1023). The ADC can measure resistance, current, sound, in fact anything that has a voltage.

To illustrate the use of the ADC peripheral, use the layout below:
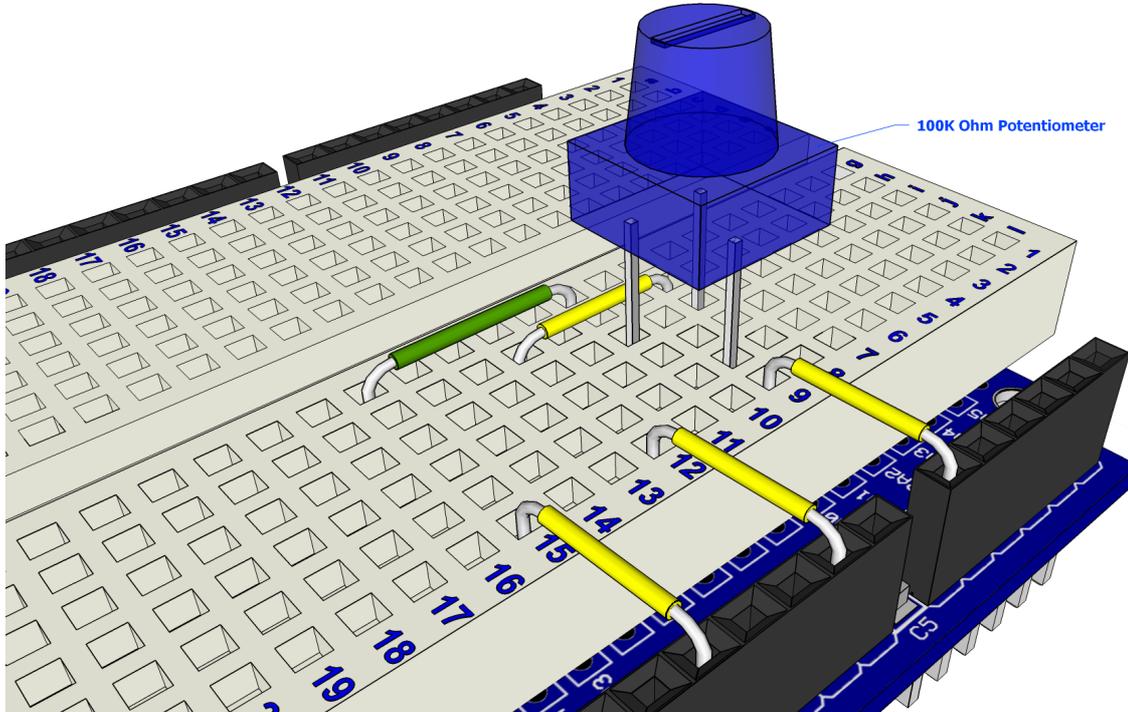
The circuit for the above layout is shown below:
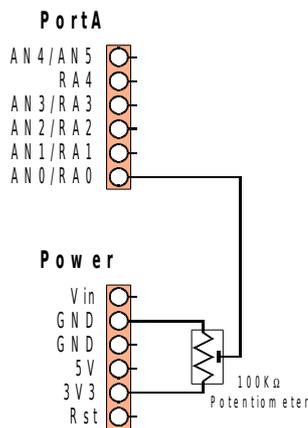
The program for the ADC demonstration is shown below:

```
' PIC18F25K20 ON THE AMICUS BOARD
' DEMOSTRATE AN ADC (ANALOG TO DIGITAL) INPUT.
' DISLPAY THE STATE OF AN0 (CHANNEL 0 OF THE ADC)
' ON THE SERIAL TERMINAL ( VIA USB CONNECTION).
    DEFINE OSC 64          'THE 16MHz IS PLLed TO 64 MHz
    DEFINE LOADER_USED 1   'USING THE AMICUS USB BOOT-LOADER
    DEFINE DEBUG_REG PORTC 'SET DEBUG PIN PORT
    DEFINE DEBUG_BIT 6     'SET DEBUG PIN BIT
    DEFINE DEBUG_BAUD 9600 'SET DEBUG BAUD RATE
    DEFINE DEBUG_MODE 0    'SET DEBUG MODE: 0 = TRUE, 1 = INVERTED

    DEFINE ADC_BITS 10     ' SETS THE ADC TO 10 BIT RESOLUTION (1023 STEPS)
    TRISA.0 = 1            ' MAKES PORTA BIT 0 AN INPUT
    ANSEL.0 = 1            ' MAKES PORTA BIT 0 ANALOG
    ADCON1 = 0             ' SET THE ADC TO VSS AND VDD AS THE REFERENCE VOLTAGE
    ADCON2.7 = 1           ' SETS THE ADC TO RIGHT JUSTIFIED FOR 10 BIT RESOLUTION
    ADC_INPUT   VAR WORD   ' VARIABLE TO HOLD THE ADC RESULT

    WHILE 1 = 1                        ' CREATE AN INFINITE LOOP
        ADCIN 0, ADC_INPUT            ' READ THE ADC AND PLACE VALUE INTO VARIABLE
        DEBUG "ADC = ", DEC ADC_INPUT,13 ' SEND VALUE TO THE TERMINAL
        PAUSE 500                      ' DELAY 50 MILLISECONDS
    WEND                               ' DO IT FOREVER
```

Once the program is compiled and loaded into the Amicus18 board by clicking on the tool-bar *Compile and Program* or pressing **F10**, open the serial terminal and connect to the Amicus18 board's com port:
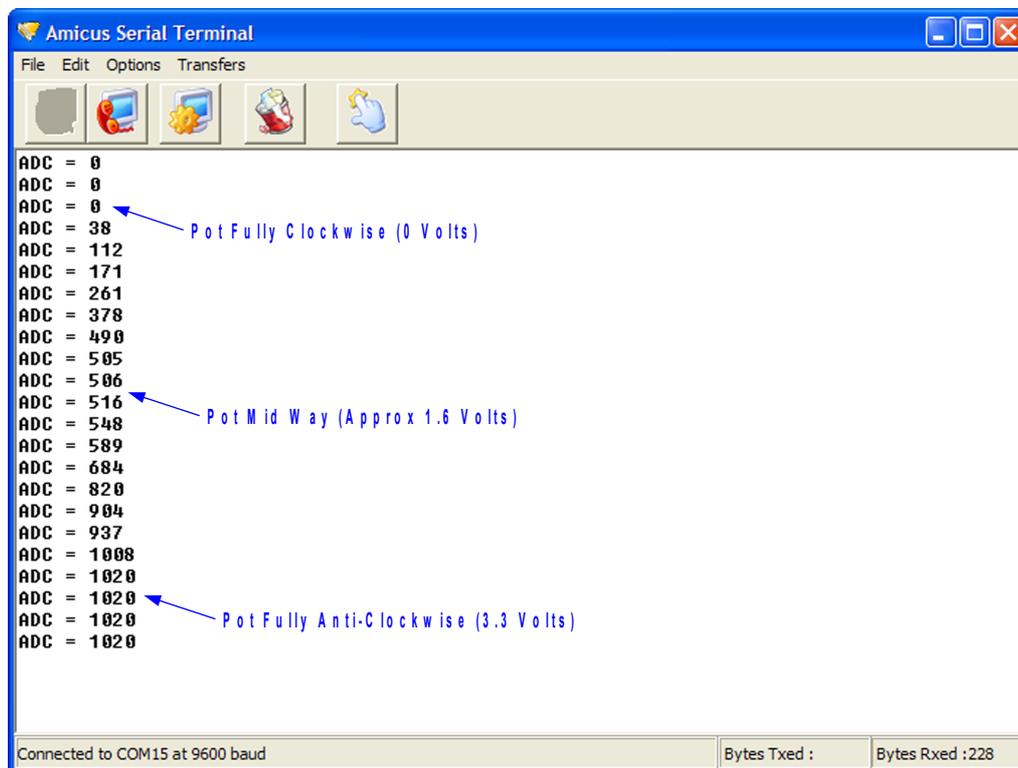
Turning the potentiometer anti-clockwise will increase the voltage to the ADC, therefore increasing the ADC's value. Turning the potentiometer clockwise will decrease the voltage to the ADC, and decrease the ADC's value, as can be seen from the screenshot above.

Don't worry too much is the ADC value isn't exactly 1023 for 3.3 Volts, as it's only a tiny fraction of the actual value, and this will make very little difference, if any, to most pro-

grams. This can be caused by many things, wrong *Tad* being used, wrong *Fosc*, losses in the wiring etc...

Light Level Switch (Cockroach Mode)

We can use the ADC for a more practical example now that we know it works. We'll use an LDR (Light Dependant Resistor) as the input to the ADC, and turn on an LED when the light level drops beyond a certain level.

An LDR, as it's name suggests, alters it's resistance depending on the amount of light falling upon it. It's one of the oldest methods of light detection, and one of the simplest of all light level detectors to use, and one of least expensive.

And LDR layout is shown below:

06-10-2009
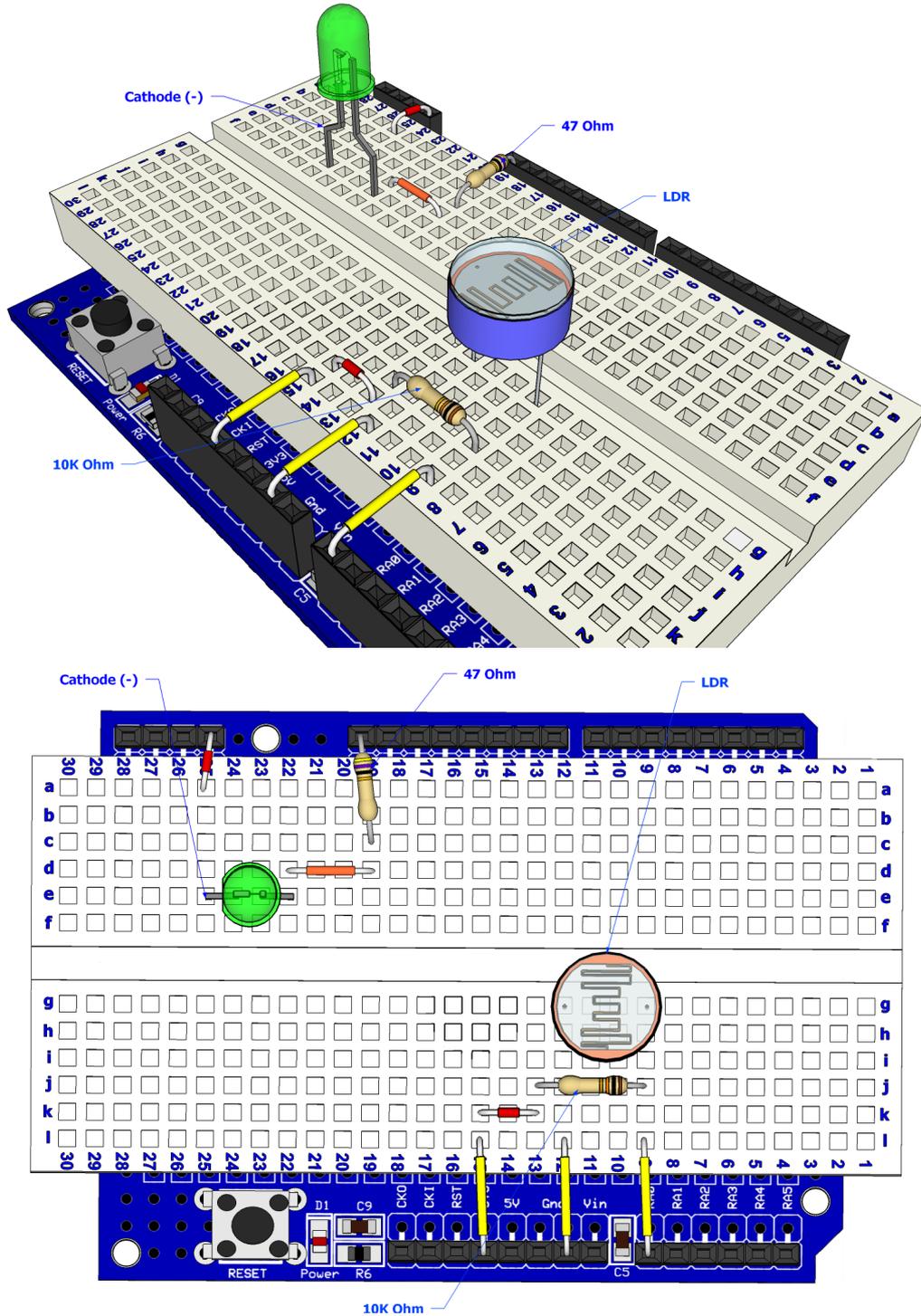
Don't worry if the LDR you use doesn't look like the one used in the layout as LDRs come in all shapes and sizes, but they all perform the same task. However, their light level resistance may vary. But again, this doesn't actually matter, as we'll be detecting changes in light level, not the level itself .

The circuit for the LDR layout is shown below:
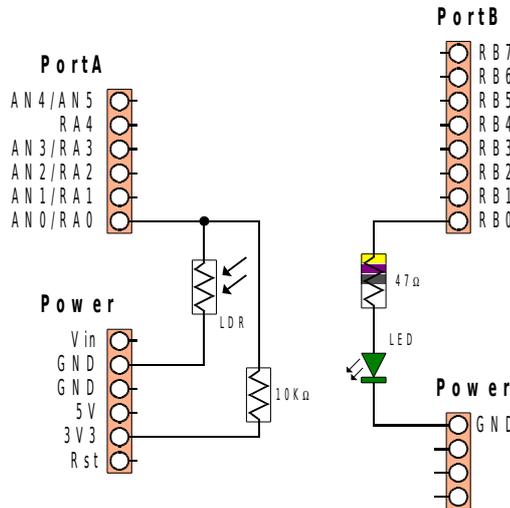
The program for the Light Level Detector is shown below. The code will activate the LED when the LDR sees a certain level of darkness, just like a cockroach:

```
' PIC18F25K20 ON THE AMICUS BOARD
' ILLUMINATE AN LED WHEN AN LDR CONNECTED TO AN0 SEES DARKNESS
' ALTERING THE VALUE WITHIN THE IF-THEN CONDITION WILL SET
' THE LIGHT LEVEL THRESHOLD.  ANY VALUE FROM 0 TO 1023 IS
' VALID, HOWEVER LARGER VALUES INDICATE DARKNESS.
    DEFINE OSC 64          'THE 16MHz IS PLLed TO 64 MHz
    DEFINE LOADER_USED 1   'USING THE AMICUS USB BOOT-LOADER

    DEFINE ADC_BITS 10     ' SETS THE ADC TO 10 BIT RESOLUTION (1023 STEPS)
    TRISA.0 = 1            ' MAKES PORTA BIT 0 AN INPUT
    ANSEL.0 = 1            ' MAKES PORTA BIT 0 ANALOG
    ADCON1 = 0            ' SET THE ADC TO VSS AND VDD AS THE REFERENCE VOLTAGE
    ADCON2.7 = 1         ' SETS THE ADC TO RIGHT JUSTIFIED FOR 10 BIT RESOLUTION
LDR_VALUE   VAR WORD     ' PIN WHERE THE GREEN LED IS CONNECTED
LED VAR PORTB.0          ' PIN WHERE THE RED LED IS CONNECTED

WHILE 1 = 1                   ' CREATE AN INFINITE LOOP
    ADCIN 0, LDR_VALUE        ' READ THE ADC AND PLACE VALUE INTO VARIABLE
    IF LDR_VALUE > 400 THEN   ' IS THE ADC VALUE ABOVE 400.(GETTING DARKER)
        HIGH LED              ' ILLUMINATE THE LED
    ELSE                      ' IF NOT..
        LOW LED               ' EXTINGUISH THE GREEN LED
    ENDIF
    PAUSEUS 30                ' ALLOW THE ADC TO RECOVER
WEND                          ' DO IT FOREVER
```

In order to change the level of darkness that the LDR will react too is simply a matter of changing the value within the line "**If** LDR_Value > *400* **Then**". A larger value will illuminate the LED at darker levels. The best way to calibrate the program is to examine the values produced by your particular LDR in light an dark situations. The program below will display the LDR values on the serial terminal:

```
' PIC18F25K20 ON THE AMICUS BOARD
' DISPLAY THE VALUE PRODUCED FROM AN LDR
    DEFINE OSC 64           'THE 16MHz IS PLLed TO 64 MHz
    DEFINE LOADER_USED 1    'USING THE AMICUS USB BOOT-LOADER
    DEFINE DEBUG_REG PORTC  'SET DEBUG PIN PORT
    DEFINE DEBUG_BIT 6      'SET DEBUG PIN BIT
    DEFINE DEBUG_BAUD 9600  'SET DEBUG BAUD RATE
    DEFINE DEBUG_MODE 0     'SET DEBUG MODE: 0 = TRUE, 1 = INVERTED

    DEFINE ADC_BITS 10      ' SETS THE ADC TO 10 BIT RESOLUTION (1023 STEPS)
    TRISA.0 = 1             ' MAKES PORTA BIT 0 AN INPUT
    ANSEL.0 = 1             ' MAKES PORTA BIT 0 ANALOG
    ADCON1 = 0              ' SET THE ADC TO VSS AND VDD AS THE REFERENCE VOLTAGE
    ADCON2.7 = 1            ' SETS THE ADC TO RIGHT JUSTIFIED FOR 10 BIT RESOLUTION
    LDR_VALUE   VAR WORD    ' VARIABLE TO HOLD THE ADC RESULT

    WHILE 1 = 1                      ' CREATE AN INFINITE LOOP
        ADCIN 0, LDR_VALUE           ' READ THE ADC AND PLACE VALUE INTO VARIABLE
        DEBUG "LDR VALUE = ", DEC LDR_VALUE,13 ' SEND VALUE TO THE TERMINAL
        PAUSE 500                    ' DELAY 50 MILLISECONDS
    WEND                             ' DO IT FOREVER
```

Once the code is compiled and loaded into the Amicus18, open the serial teminal:
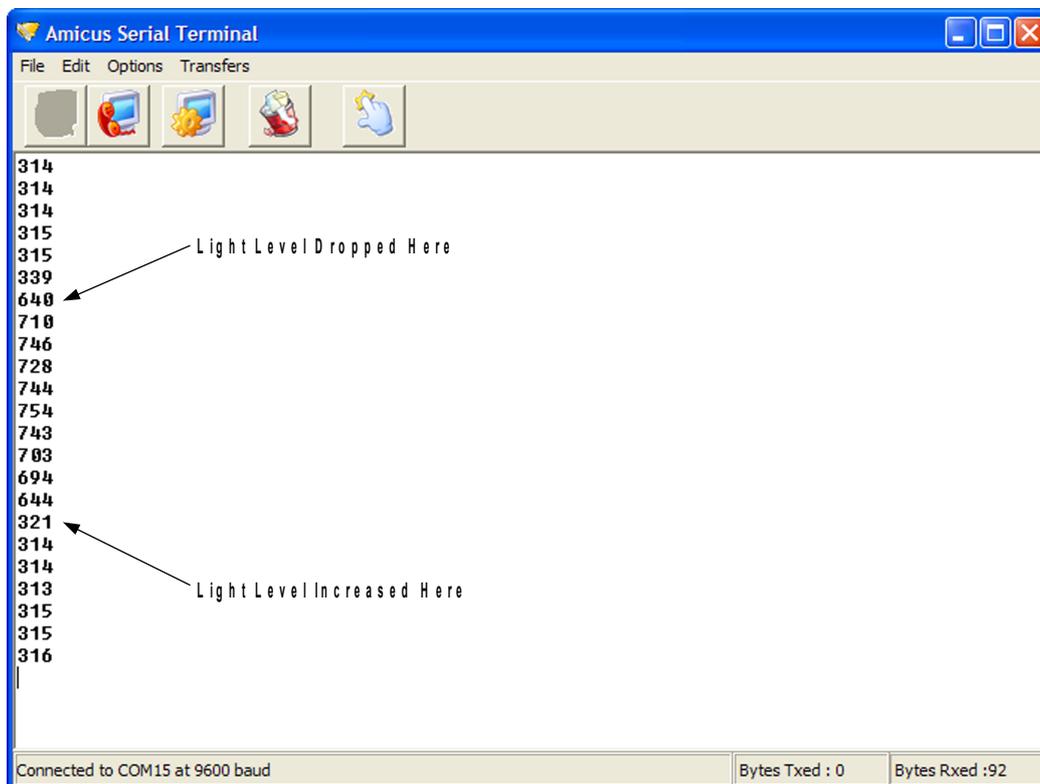
As can be seen from the above screenshot, ambient light levels give an approximate value of 315, so anything above this value will indicate a light level decrease. However, we don't want to make it too sensitive, so a value of 400 is ideal.

Light Level Switch (Moth Mode)

The same circuit and layout is used for the opposite reaction to light levels. The code below will illuminate the LED when light levels increase, just like a moth to a flame.

The code is essentially the same as cockroach mode, except the LED illuminates when the ADC value is less that 400, instead of greater than 400.

```
' PIC18F25K20 ON THE AMICUS BOARD
' ILLUMINATE AN LED WHEN AN LDR CONNECTED TO AN0 SEES LIOGHT
' ALTERING THE VALUE WITHIN THE IF-THEN CONDITION WILL SET
' THE LIGHT LEVEL THRESHOLD.  ANY VALUE FROM 0 TO 1023 IS
' VALID, HOWEVER LARGER VALUES INDICATE DARKNESS.
    DEFINE OSC 64          'THE 16MHz IS PLLed TO 64 MHz
    DEFINE LOADER_USED 1   'USING THE AMICUS USB BOOT-LOADER

    DEFINE ADC_BITS 10     ' SETS THE ADC TO 10 BIT RESOLUTION (1023 STEPS)
    TRISA.0 = 1            ' MAKES PORTA BIT 0 AN INPUT
    ANSEL.0 = 1            ' MAKES PORTA BIT 0 ANALOG
    ADCON1 = 0             ' SET THE ADC TO VSS AND VDD AS THE REFERENCE VOLTAGE
    ADCON2.7 = 1           ' SETS THE ADC TO RIGHT JUSTIFIED FOR 10 BIT RESOLUTION
    LDR_VALUE   VAR WORD   ' VARIABLE TO HOLD THE ADC RESULT
    LED VAR PORTB.0        ' PIN WHERE THE LED IS CONNECTED

    WHILE 1 = 1                 ' CREATE AN INFINITE LOOP
        ADCIN 0, LDR_VALUE      ' READ THE ADC AND PLACE VALUE INTO VARIABLE
        IF LDR_VALUE <= 400 THEN ' IS THE ADC VALUE BELOW 400.(GETTING LIGHTER)
            HIGH LED            ' ILLUMINATE THE LED
        ELSE                    ' IF NOT..
            LOW LED             ' EXTINGUISH THE RED LED
        ENDIF
        PAUSEUS 30              ' ALLOW THE ADC TO RECOVER
    WEND                        ' DO IT FOREVER
```

Temperature Sensor

One of the simplest, and least expensive, temperature sensors is a thermistor. This is a special type of resistor that alters it's resistance based upon it's temperature. There are generally two types of thermistor; an NTC type (Negative Temperature Coefficient), whose resistance drops as the temparature increases, and a PTC type (Positive Temparature Coefficient), whose resistance increases as the temparature increases. For this demonstration, we'll use an NTC thermistor.

Just like there fixed resistance cousins, thermistors come in different packages and resistance-per-temperature values. These range anywhere from a few hundred Ohms to tens of thousands of Ohms.

The device used in this demonstration is a bead thermistor with a resistance of 10KΩ at a temperature of 25° centigrade, but any thermistor will do with a few program code changes.

A thermistor layout is shown below:

Cathode (-)

47 Ohm

Thermistor (NTC)

10K Ohm

47 Ohm

Thermistor (NTC)

10K Ohm

06-10-2009
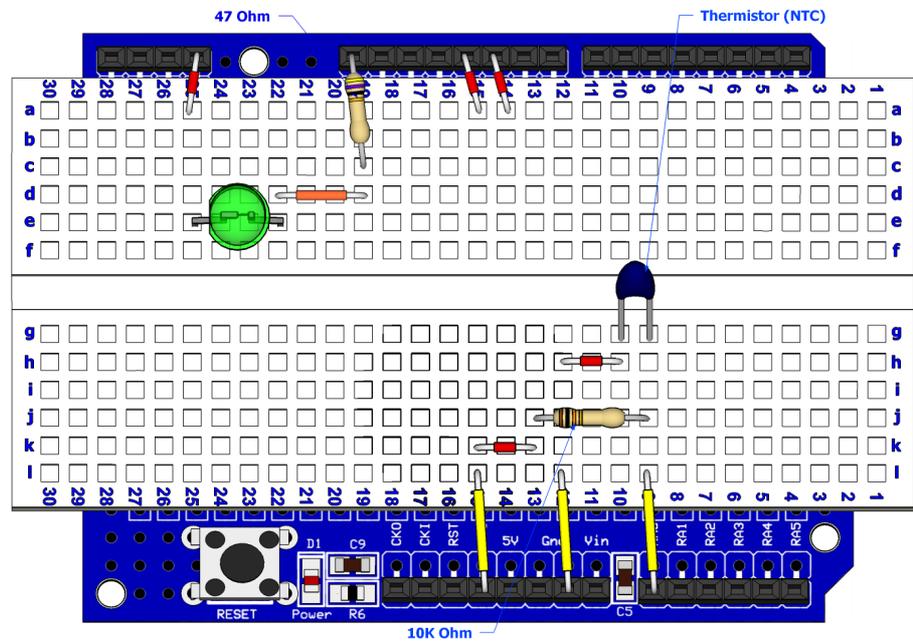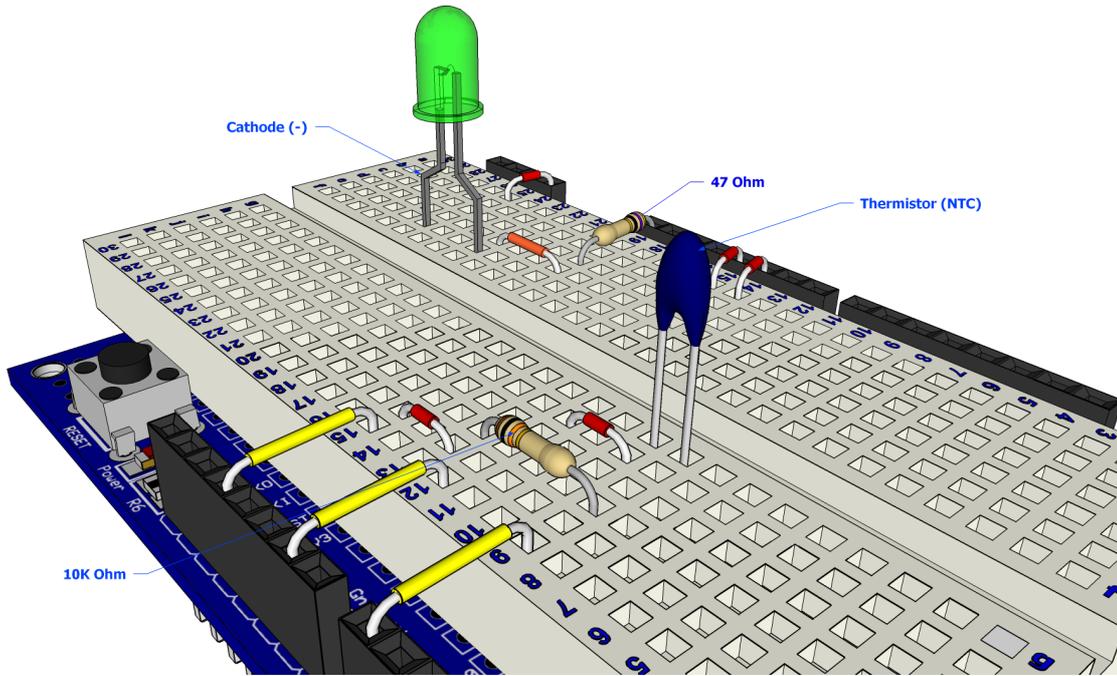
# Amicus18 Companion Shield

The circuit for the temperature layout is shown below:
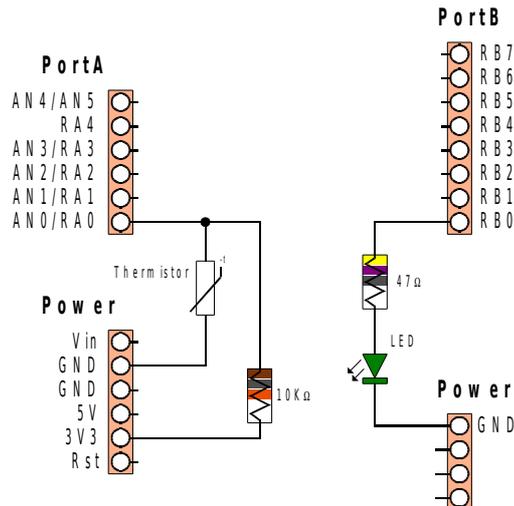
A program to display the the values produced from the thermistor on the serial terminal is shown below:

```
' PIC18F25K20 ON THE AMICUS BOARD
' DISLPAY THE VALUE OF AN NTC THERMISTOR ON THE SERIAL TERMINAL.
' THE THERMISTOR IS CONNECTED TO AN0(CHANNEL 0 OF THE ADC).
  DEFINE OSC 64            'THE 16MHz IS PLLed TO 64 MHz
  DEFINE LOADER_USED 1     'USING THE AMICUS USB BOOT-LOADER
  DEFINE DEBUG_REG PORTC   'SET DEBUG PIN PORT
  DEFINE DEBUG_BIT 6       'SET DEBUG PIN BIT
  DEFINE DEBUG_BAUD 9600   'SET DEBUG BAUD RATE
  DEFINE DEBUG_MODE 0      'SET DEBUG MODE: 0 = TRUE, 1 = INVERTED

  DEFINE ADC_BITS 10       ' SETS THE ADC TO 10 BIT RESOLUTION (1023 STEPS)
  TRISA.0 = 1              ' MAKES PORTA BIT 0 AN INPUT
  ANSEL.0 = 1              ' MAKES PORTA BIT 0 ANALOG
  ADCON1 = 0              ' SET THE ADC TO VSS AND VDD AS THE REFERENCE VOLTAGE
  ADCON2.7 = 1            ' SETS THE ADC TO RIGHT JUSTIFIED FOR 10 BIT RESOLUTION
  THERMISTOR_IN   VAR WORD    ' VARIABLE TO HOLD THE ADC RESULT

WHILE 1 = 1
    ADCIN 0, THERMISTOR_IN
    DEBUG "THERMISTOR = ", DEC THERMISTOR_IN,13
    PAUSE 500
WEND
```

Once the program has been loaded into the Amicus18 board, open the serial terminal and connect to the Amicus18's com port:
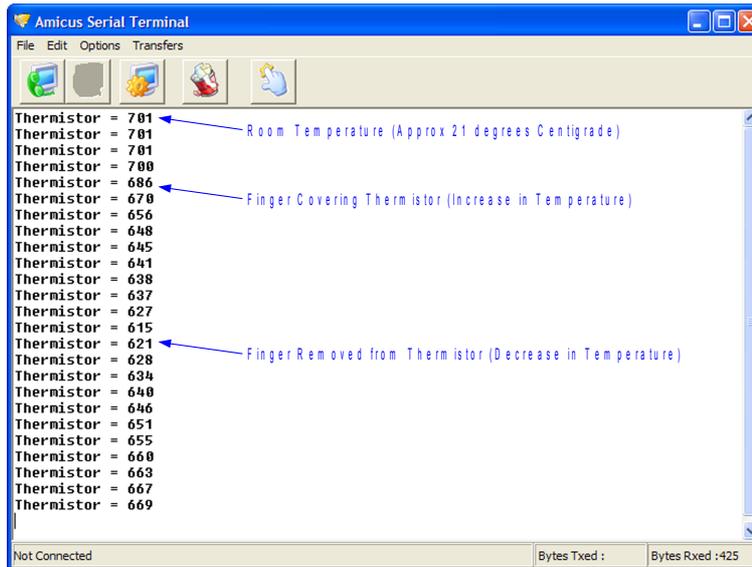
The display shows the decrease in voltage with the increase in temperature when a finger covers the thermistor, and is then removed. As can be seen, a thermistor is quite sensitive.

Thermostat (increase in temperature)

We can use the information we have to trigger an external device, in this case an LED, when the thermistor reaches a pre-determined value. We know that room temperature give an ADC value of approx 701, and any value lower than this is an increase in temperature, and a lower value is a decrease in temperature, so even without knowing the actual temperature we can write some code:

```
' PIC18F25K20 ON THE AMICUS BOARD
' ILLUMINATE AN LED WHEN THE TEMPERATURE INCREASES.
' DISLPAY THE VALUE OF AN NTC THERMISTOR ON THE SERIAL TERMINAL.
' THE THERMISTOR IS CONNECTED TO AN0(CHANNEL 0 OF THE ADC).
    DEFINE OSC 64            'THE 16MHz IS PLLed TO 64 MHz
    DEFINE LOADER_USED 1     'USING THE AMICUS USB BOOT-LOADER
    DEFINE DEBUG_REG PORTC   'SET DEBUG PIN PORT
    DEFINE DEBUG_BIT 6       'SET DEBUG PIN BIT
    DEFINE DEBUG_BAUD 9600   'SET DEBUG BAUD RATE
    DEFINE DEBUG_MODE 0      'SET DEBUG MODE: 0 = TRUE, 1 = INVERTED

    DEFINE ADC_BITS 10       ' SETS THE ADC TO 10 BIT RESOLUTION (1023 STEPS)
    TRISA.0 = 1              ' MAKES PORTA BIT 0 AN INPUT
    ANSEL.0 = 1              ' MAKES PORTA BIT 0 ANALOG
    ADCON1 = 0              ' SET THE ADC TO VSS AND VDD AS THE REFERENCE VOLTAGE
    ADCON2.7 = 1            ' SETS THE ADC TO RIGHT JUSTIFIED FOR 10 BIT RESOLUTION
    THERMISTOR_IN  VAR WORD   ' VARIABLE TO HOLD THE ADC RESULT
    LED VAR PORTB.0          ' PIN WHERE THE LED IS CONNECTED

WHILE 1 = 1                           ' CREATE AN INFINITE LOOP
    ADCIN 0, THERMISTOR_IN            ' READ THE ADC AND PLACE VALUE INTO VARIABLE
    DEBUG "THERMISTOR = ", DEC THERMISTOR_IN,13 ' SEND VALUE TO THE TERMINAL
    PAUSE 50                          ' DELAY 50 MILLISECONDS
    IF THERMISTOR_IN < 600 THEN       ' INCREASE IN TEMPERATURE ?
        HIGH LED                      ' ILLUMINATE THE LED
    ELSE                              ' IF NOT...
        LOW LED
    ENDIF                             ' EXTINGUISH THE LED
WEND                                  ' DO IT FOREVER
```

Once the program is compiled and loaded into the Amicus18 board using the toolbar *Compile and Program* or pressing *F10*, placing a finger over the thermistor, thus increasing the temperature, will illuminate the LED. To adjust the threshold of the temperature trigger, alter the value within the code line: "**If** ThermistorIn < 600 **Then**". A lower value will illuminate the LED at higher temparatures.

Thermostat (decrease in temperature)

In order to illuminate the LED at lower temperatures, use the program below:

Once the program is compiled and loaded into the Amicus18 board using the toolbar *Compile and Program* or pressing **F10**, blowing over the thermistor, thus decreasing the temperature, will illuminate the LED. To adjust the threshold of the temperature trigger, alter the value within the code line: "**If** ThermistorIn >= 750 **Then**". A higher value will illuminate the LED at lower temparatures.

```
' PIC18F25K20 ON THE AMICUS BOARD
' ILLUMINATE AN LED WHEN THE TEMPERATURE DECREASES.
' DISLPAY THE VALUE OF AN NTC THERMISTOR ON THE SERIAL TERMINAL.
' THE THERMISTOR IS CONNECTED TO AN0(CHANNEL 0 OF THE ADC).
    DEFINE OSC 64              'THE 16MHz IS PLLed TO 64 MHz
    DEFINE LOADER_USED 1       'USING THE AMICUS USB BOOT-LOADER
    DEFINE DEBUG_REG PORTC     'SET DEBUG PIN PORT
    DEFINE DEBUG_BIT 6         'SET DEBUG PIN BIT
    DEFINE DEBUG_BAUD 9600     'SET DEBUG BAUD RATE
    DEFINE DEBUG_MODE 0        'SET DEBUG MODE: 0 = TRUE, 1 = INVERTED

     DEFINE ADC_BITS 10     ' SETS THE ADC TO 10 BIT RESOLUTION (1023 STEPS)
    TRISA.0 = 1               ' MAKES PORTA BIT 0 AN INPUT
    ANSEL.0 = 1               ' MAKES PORTA BIT 0 ANALOG
    ADCON1 = 0                ' SET THE ADC TO VSS AND VDD AS THE REFERENCE VOLTAGE
    ADCON2.7 = 1              ' SETS THE ADC TO RIGHT JUSTIFIED FOR 10 BIT RESOLUTION
    THERMISTOR_IN   VAR WORD     ' VARIABLE TO HOLD THE ADC RESULT
    LED VAR PORTB.0           ' PIN WHERE THE LED IS CONNECTED

    WHILE 1 = 1                         ' CREATE AN INFINITE LOOP
        ADCIN 0, THERMISTOR_IN      ' READ THE ADC AND PLACE VALUE INTO VARIABLE
        DEBUG "THERMISTOR = ", DEC THERMISTOR_IN,13 ' SEND VALUE TO THE TERMINAL
        PAUSE 50                     ' DELAY 50 MILLISECONDS
        IF THERMISTOR_IN >= 700 THEN ' DECREASE IN TEMPERATURE ?
            HIGH LED                 ' ILLUMINATE THE LED
        ELSE                         ' IF NOT...
            LOW LED                  ' EXTINGUISH THE LED
        ENDIF
    WEND                             ' DO IT FOREVER
```

Amicus18 Companion Shield

Thermostat (increase and decrease of temperature)

The layout and code below allows the demonstration of high, normal, and low temperature changes. Both LEDs will be extinguished when the temperature is normal, the Red LED will illuminate when the temperature rises above a pre-determined value, and the Green LED will illuminate when the temperature decreases beyond a pre-determined level:
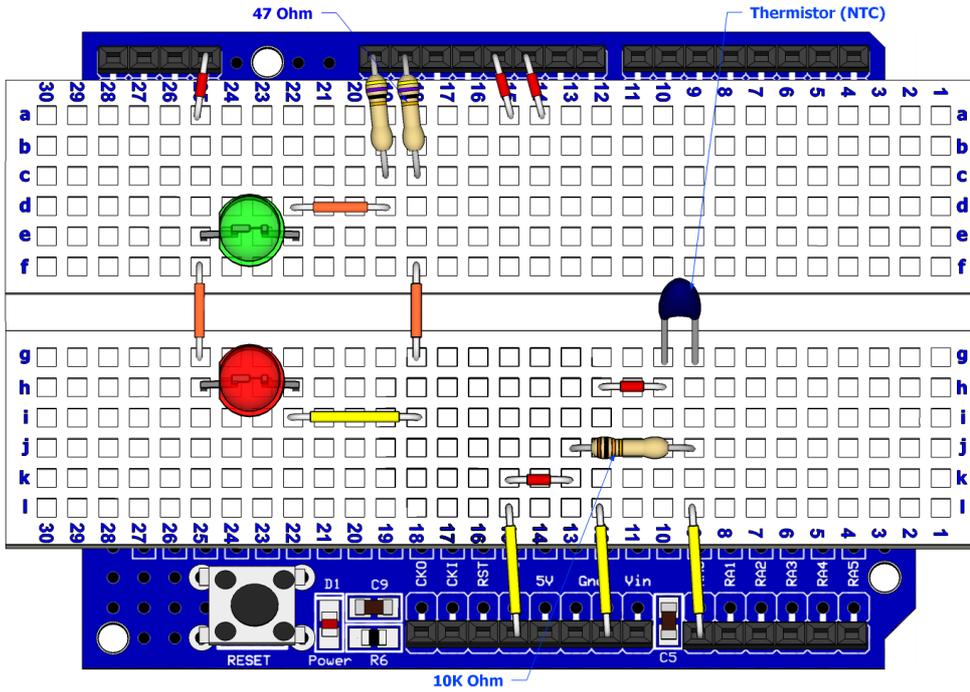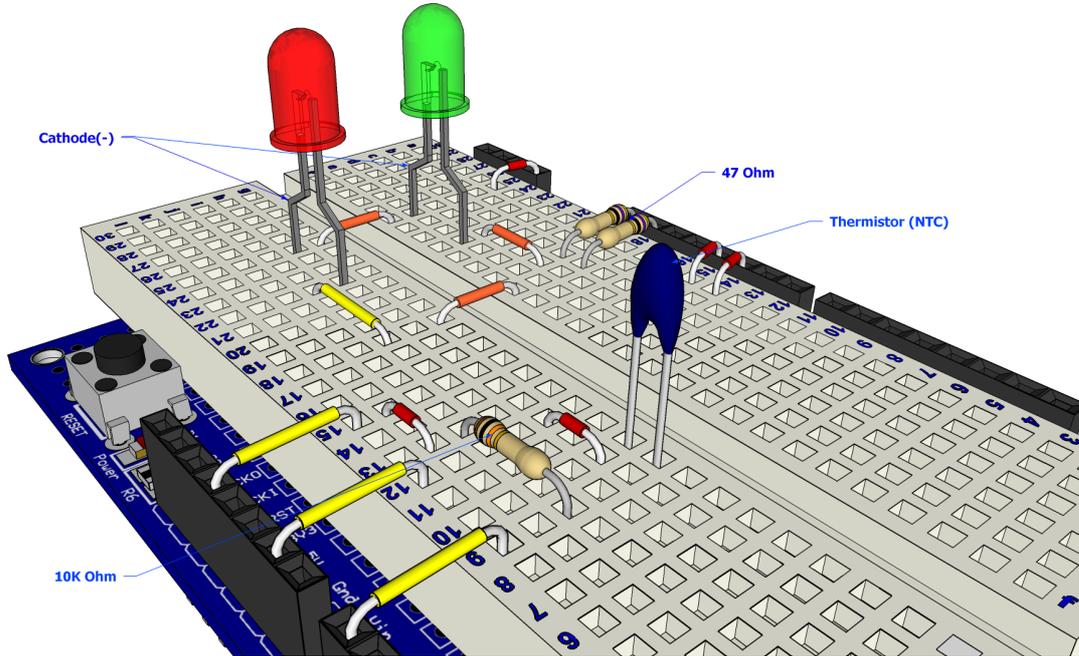
Amicus18 Companion Shield

The circuit for the thermistat layout is shown below:
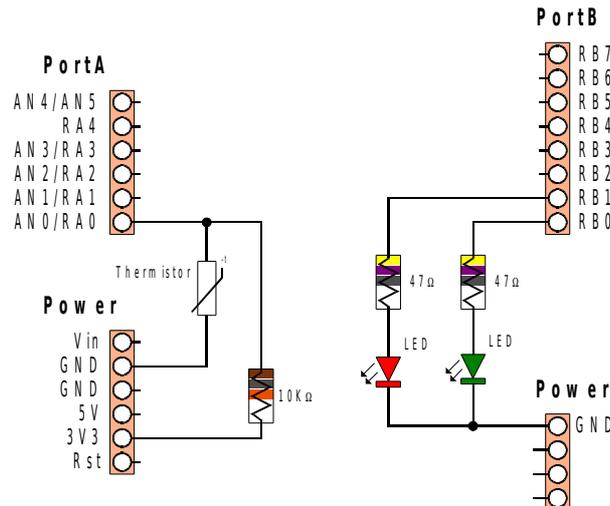
The code for the two LED thermostat is shown below:

```
' PIC18F25K20 ON THE AMICUS BOARD
' ILLUMINATE A RED LED WHEN THE TEMPERATURE INCREASES.
' ILLUMINATE A GREEN LED WHEN THE TEMPERATURE DECREASES.
' DISLPAY THE VALUE OF AN NTC THERMISTOR ON THE SERIAL TERMINAL.
' THE THERMISTOR IS CONNECTED TO AN0(CHANNEL 0 OF THE ADC).

   DEFINE OSC 64           'THE 16MHz IS PLLed TO 64 MHz
   DEFINE LOADER_USED 1    'USING THE AMICUS USB BOOT-LOADER

   DEFINE ADC_BITS 10      ' SETS THE ADC TO 10 BIT RESOLUTION (1023 STEPS)
   TRISA.0 = 1             ' MAKES PORTA BIT 0 AN INPUT
   ANSEL.0 = 1            ' MAKES PORTA BIT 0 ANALOG
   ADCON1 = 0                ' SET THE ADC TO VSS AND VDD AS THE REFERENCE VOLTAGE
   ADCON2.7 = 1             ' SETS THE ADC TO RIGHT JUSTIFIED FOR 10 BIT RESOLUTION
   THERMISTOR_IN  VAR WORD  ' VARIABLE TO HOLD THE ADC RESULT
   GREEN_LED VAR PORTB.0    ' PIN WHERE THE GREEN LED IS CONNECTED
   RED_LED  VAR PORTB.1     ' PIN WHERE THE RED LED IS CONNECTED

   WHILE 1 = 1                        ' CREATE AN INFINITE LOOP
       ADCIN 0, THERMISTOR_IN         ' READ THE ADC AND PLACE VALUE INTO VARIABLE
       DEBUG "THERMISTOR = ", DEC THERMISTOR_IN,13 ' SEND VALUE TO THE TERMINAL
       PAUSE 50                       ' DELAY 50 MILLISECONDS
       IF THERMISTOR_IN < 600 THEN    ' INCREASE IN TEMPERATURE ?
           HIGH RED_LED               ' YES ? ILLUMINATE THE RED LED
           LOW GREEN_LED              ' EXTINGUISH THE GREEN LED
       ELSEIF THERMISTOR_IN > 750 THEN ' DECREASE IN TEMPERATURE ?
           LOW RED_LED                ' YES ? EXTINGUISH THE RED LED
           HIGH GREEN_LED             ' ILLUMINATE THE GREEN LED
       ELSE                           ' IF NOT..
           LOW RED_LED                ' EXTINGUISH THE RED LED
           LOW GREEN_LED              ' EXTINGUISH THE GREEN LED
       ENDIF
   WEND                               ' DO IT FOREVER
```

Digital Meets Analogue

Sometimes the microcontroller needs to interface back to the real world with an analogue result. This is termed Digital to Analogue Conversion, or DAC. This can be performed several ways; by using a dedicated DAC peripheral device, by using a digital resistor device, or by using Pulse Width Modulation (PWM). PWM is the method that is built into the Amicus18's microcontroller, and requires no specialised devices to be used, so we'll discuss this method here.

Pulse Width Modulation (PWM)

Pulse Width Modulation fakes a voltage by producing a series of pulses at regular intervals, and varying the width of the pulses. The resulting average voltage is the result of the pulse widths. The Amicus18's microcontroller can produce a high voltage of 3.3 Volts and low of 0 Volts.

In the illustration below, the pin is pulsed high for the same length of time as it is pulsed low. The time the pin is high (called the pulsewidth) is about half the total time it takes to go from low to high to low again. This ratio is called the duty cycle. When the duty cycle is 50%, the average voltage is about half the total voltage. i.e. 1.6 Volts.
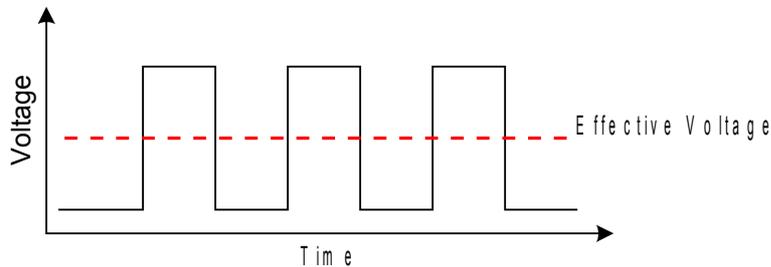
If the duty cycle is made less than 50% by pulsing on for a shorter amount of time, a lower effective voltage is produced:
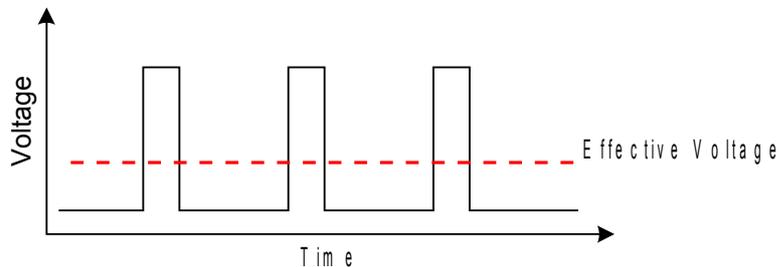
If the duty cycle is made greater than 50% by pulsing on for a longer amount of time, a higher effective voltage is produced:
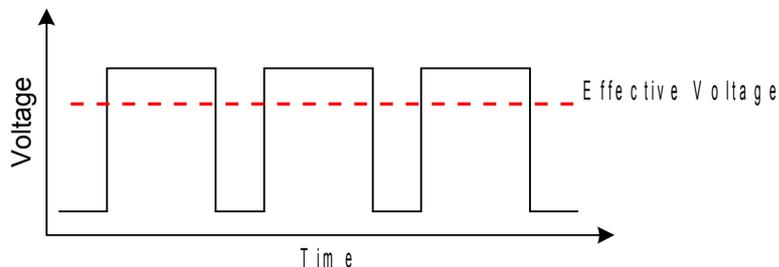
In order to create a constant voltage instead of a series of pulses, we need a simple RC low pass filter. As it's name suggests this consists of a Resistor and a Capacitor.

A filter is a circuit that allows voltage changes of only a certain frequency range to pass. For example, a low-pass filter would block frequencies above a certain range. This means that if the voltage is changing more than a certain number of times per second, these changes would not make it past the filter, and only an average voltage would be seen.

There are calculations for the values of the resistor and capacitor used, but we won't go into that here, but a search for *RC filter* on the internet will produce a huge amount of information. Here's two of them that are valid at the time of writing:

http://www.cvs1.uklinux.net/cgi-bin/calculators/time_const.cgi
http://www.sengpielaudio.com/calculator-period.htm

Channel 1 PWM

If we choose a value of 47 Ohms for our resistor so that we don't loose too much current, we need a capacitance value of 340.425nF (0.34uF). There is no common capacitor of that value so we'll choose a close value, for example 330nF (0.33uF).

The circuit for a suitable RC low pass filter is shown below:
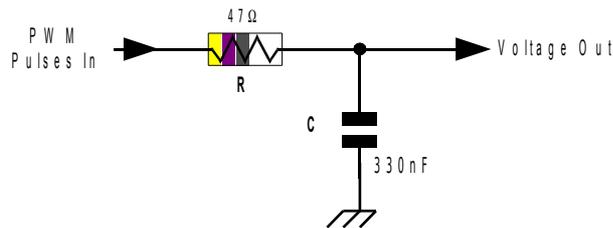
The Amicus18's microcontroller has two PWM peripherals; PWM1 from PortC pin RC2, and PWM2 from PortC pin RC1. Each pin can produce a differing duty cycle (average voltage), but each share the same frequency.

A demo layout for channel 1 of the PWM is shown below:

47
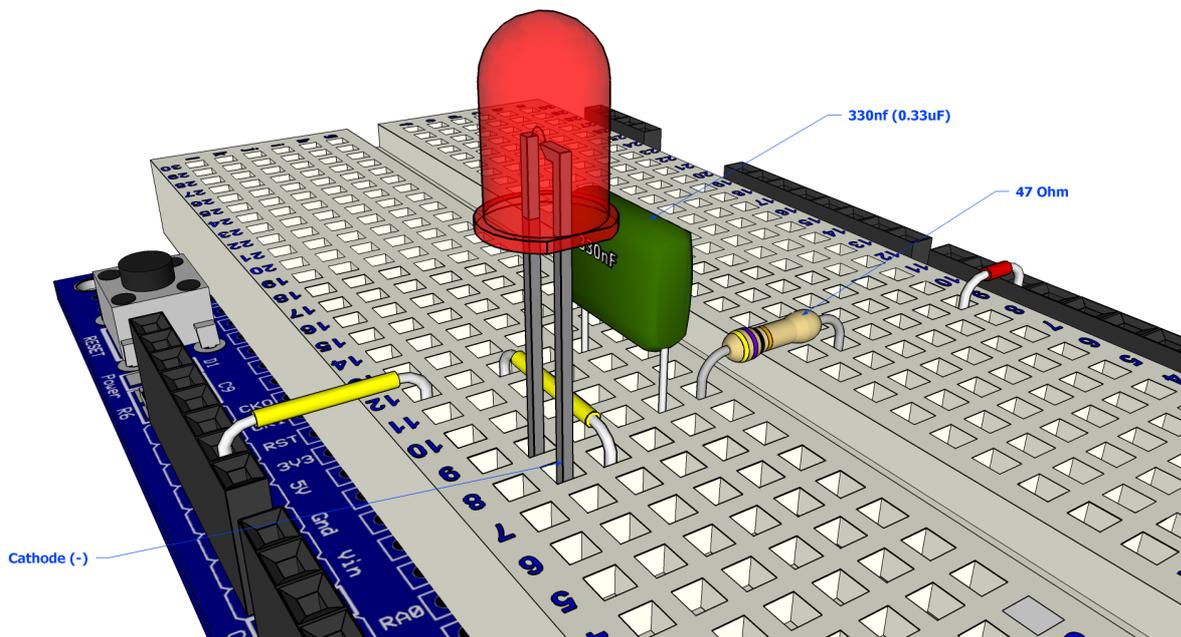
The circuit for the PWM1 layout is shown below:
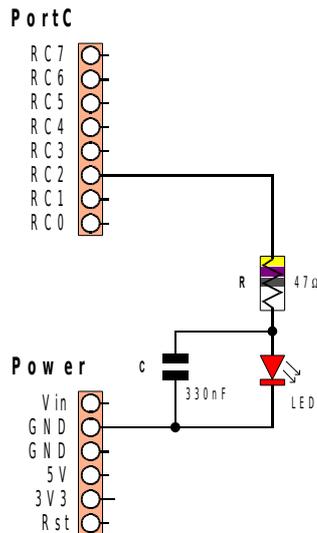
The PWM peripherals operate in the background, which means that once a PWM duty cycle is set, it does not block any other instructions from occurring.

Type in the following code and program it into the Amicus18 board by clicking on the tool-bar *Compile and Program*, or pressing **F10**:

```
' PIC18F25K20 ON THE AMICUS BOARD
' AMICUS18 HARDWARE PWM (PULSE WIDTH MODULATION) DEMO PROGRAM

    DEFINE OSC 64          'THE 16MHZ IS PLLED TO 64 MHZ
    DEFINE LOADER_USED 1  'USING THE AMICUS USB BOOT-LOADER

    DEFINE CCP1_REG PORTC   ' CHANNEL 1 PORT
    DEFINE CCP1_BIT 2       ' CHANNEL 1 BIT
    TRISC.2 = 0             ' SET PORTC.2 TO OUTPUT (CCP1)

    HPWM 1,127,2000          ' 50% DUTY CYCLE AT 2 KHZ
```

The LED will now be glowing, but not at full brightness. What's happening is that channel 1 of the PWM has been instructed to set the duty cycle to 50%, which is half the full range of 255, which is 127. Try different DUTY cycle values and see what it does to the LED's brightness.

Amicus18 Companion Shield

A more sophisticated program is shown below that will cycle the LED to full brightness then back to off repeatedly:

```
' PIC18F25K20 ON THE AMICUS BOARD
' AMICUS18 HARDWARE PWM (PULSE WIDTH MODULATION) DEMO PROGRAM
' AN LED ATTACHED TO BIT-2 OF PORTC (RC2) WILL INCREASE ILLUMINATION, THEN DIM

    DEFINE OSC 64            'THE 16MHZ IS PLLED TO 64 MHZ
    DEFINE LOADER_USED 1     'USING THE AMICUS USB BOOT-LOADER

    DEFINE CCP1_REG PORTC    ' CHANNEL 1 PORT
    DEFINE CCP1_BIT 2        ' CHANNEL 1 BIT
    TRISC.2 = 0              ' SET PORTC.2 TO OUTPUT (CCP1)
    DUTY    VAR BYTE         ' CREATE A VARIABLE FOR DUTY CYCLE

    WHILE 1 = 1             ' CREATE AN INFINITE LOOP
' INCREASE LED BRIGHTNES
    FOR DUTY = 0 TO 255     ' CYCLE THE FULL RANGE 0 TO 255
        HPWM 1,DUTY,2000    ' PWM ON CCP1 (BIT-2 OF PORTC)
        PAUSE 10            ' TIME TO VIEW
    NEXT

' DECREASE LED BRIGHTNESS
    FOR DUTY = 255 TO 0 STEP -1 ' CYCLE THE FULL RANGE 255 TO 0
        HPWM 1,DUTY,2000    ' PWM ON CCP1 (BIT-2 OF PORTC)
        PAUSE 10            ' TIME TO VIEW
    NEXT
    WEND                    ' DO IT FOREVER
```

49

Channel 2 PWM

As has been mentioned, the Amicus18 has two hardware PWM channels, each can work independently of each other when adjusting the duty cycle, but share a common operating frequency and resolution. This is because they both operate from the microcontroller's Timer 2 module.

Operating the second channel of the PWM peripheral uses exactly the same procedure as operating channel 1, but uses a different pin of PortC (RC1).

A demo layout for channel 2 of the PWM is shown below:

The circuit for the above layout is shown below:
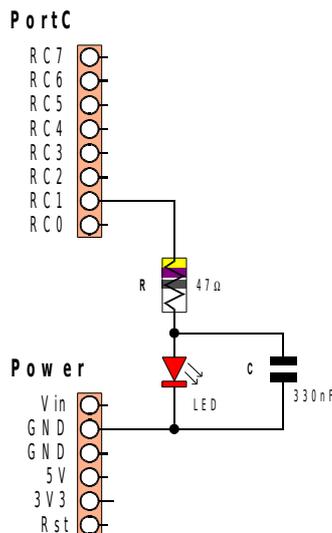
The PWM peripherals operate in the background, which means that once a PWM duty cycle is set, it does not block any other instructions from occurring.

Type in the following code and program it into the Amicus18 board by clicking on the tool-bar *Compile and Program*, or pressing **F10**:

```
' PIC18F25K20 ON THE AMICUS BOARD
' AMICUS18 HARDWARE PWM (PULSE WIDTH MODULATION) DEMO PROGRAM

    DEFINE OSC 64           'THE 16MHZ IS PLLED TO 64 MHZ
    DEFINE LOADER_USED 1    'USING THE AMICUS USB BOOT-LOADER

    DEFINE CCP2_REG PORTC   ' CHANNEL 2 PORT
    DEFINE CCP2_BIT 1       ' CHANNEL 2 BIT
    TRISC.1 = 0             ' SET PORTC.1 TO OUTPUT (CCP2)

    HPWM 2,127,2000         ' 50% DUTY CYCLE AT 2 KHZ
```

The LED will now be glowing, but not at full brightness. What's happening is that channel 2 of the PWM has been instructed to set the duty cycle to 50%, which is half the full range of 1023, which is 512. Try different values within the braces of the WriteAnalog2 command and see what it does to the LED's brightness.

A more sophisticated program is shown below that will cycle the LED to full brightness then back to off repeatedly:

```
' PIC18F25K20 ON THE AMICUS BOARD
' AMICUS18 HARDWARE PWM (PULSE WIDTH MODULATION) DEMO PROGRAM
' AN LED ATTACHED TO BIT-1 OF PORTC (RC1) WILL INCREASE ILLUMINATION, THEN DIM

    DEFINE OSC 64           'THE 16MHZ IS PLLED TO 64 MHZ
    DEFINE LOADER_USED 1    'USING THE AMICUS USB BOOT-LOADER

    DEFINE CCP2_REG PORTC   ' CHANNEL 2 PORT
    DEFINE CCP2_BIT 1       ' CHANNEL 2 BIT
    TRISC.1 = 0             ' SET PORTC.2 TO OUTPUT (CCP1)
    DUTY    VAR BYTE        ' CREATE A VARIABLE FOR DUTY CYCLE

    WHILE 1 = 1             ' CREATE AN INFINITE LOOP
' INCREASE LED BRIGHTNES
    FOR DUTY = 0 TO 255     ' CYCLE THE FULL RANGE 0 TO 255
        HPWM 2,DUTY,2000    ' PWM ON CCP2 (BIT-1 OF PORTC)
        PAUSE 10            ' TIME TO VIEW
    NEXT

' DECREASE LED BRIGHTNESS
    FOR DUTY = 255 TO 0 STEP -1 ' CYCLE THE FULL RANGE 255 TO 0
        HPWM 2,DUTY,2000    ' PWM ON CCP2 (BIT-1 OF PORTC)
        PAUSE 10            ' TIME TO VIEW
    NEXT
    WEND                    ' DO IT FOREVER
```

06-10-2009

Two channels of PWM simultaneously (Pulsing Light)

The layout below demonstrates both PWM channels operating simultaneously:

06-10-2009
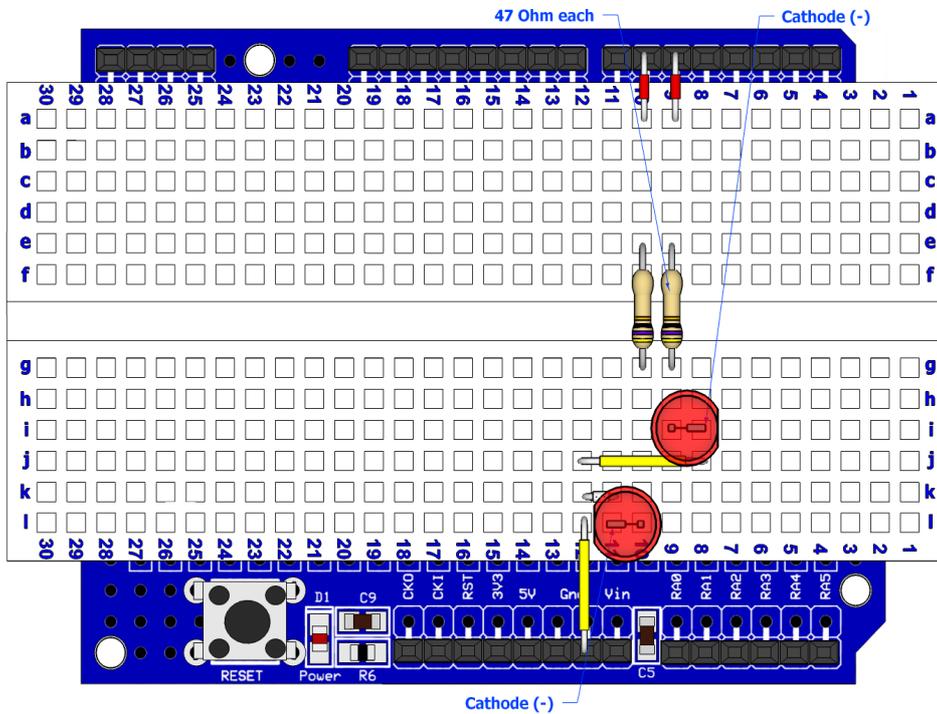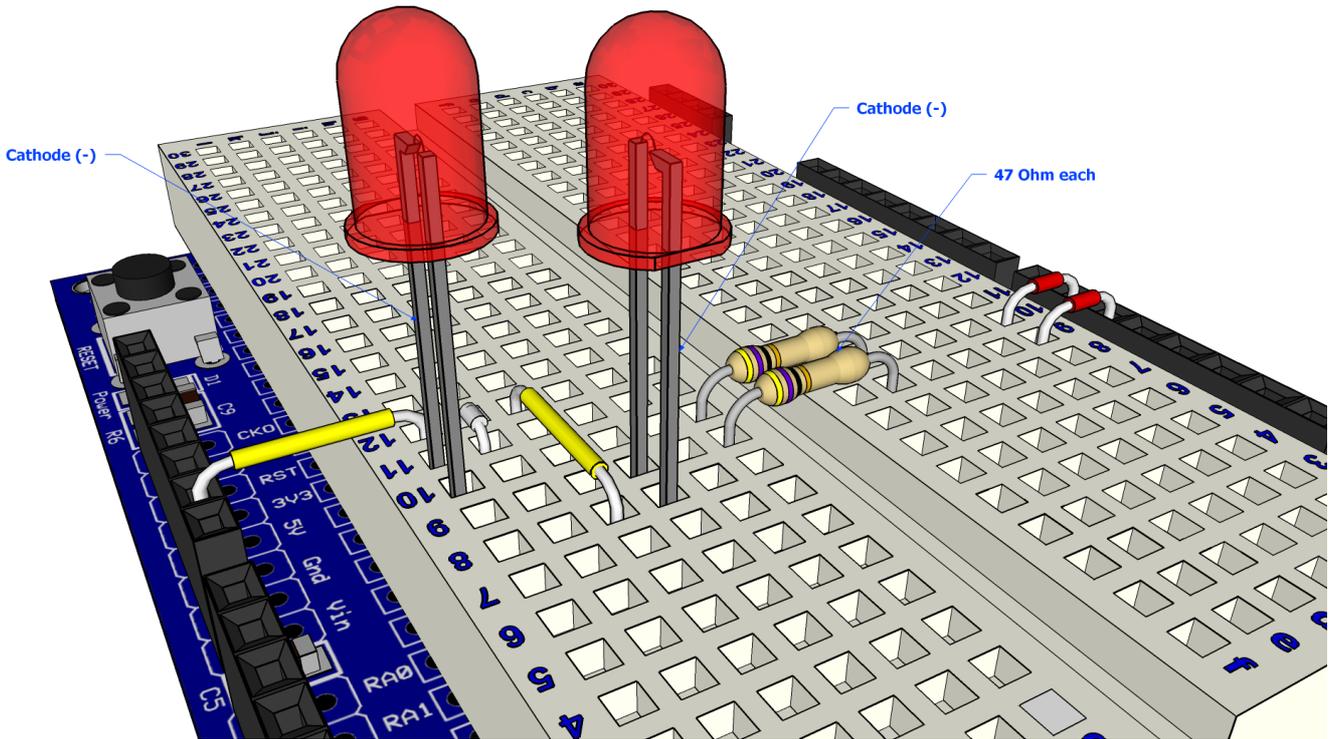
The circuit for the 2 PWMs layout is shown below:
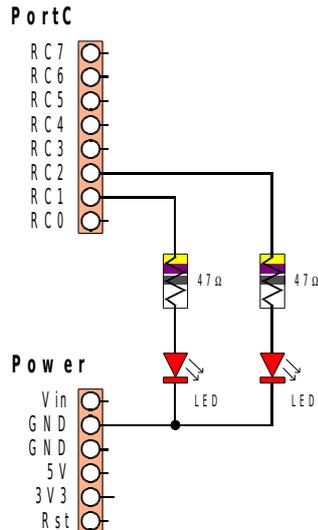
The capacitors normally associated with PWM output have been dispensed with because the operating frequency of the PWM channels is so high (62.5KHz) that no noticeable flicker from the pulses will be observed on the LEDs.

The code to produce the pulsing of the LEDs is shown below:

```
' PIC18F25K20 ON THE AMICUS BOARD
' AMICUS18 HARDWARE PWM (PULSE WIDTH MODULATION) DEMO PROGRAM
' PULSE BOTH LED, ONE INCREASES WHILE THE OTHER DECREASES IN BRIGHTNESS
    DEFINE OSC 64           'THE 16MHZ IS PLLED TO 64 MHZ
    DEFINE LOADER_USED 1    'USING THE AMICUS USB BOOT-LOADER

    DEFINE CCP1_REG PORTC   ' CHANNEL 1 PORT
    DEFINE CCP1_BIT 2       ' CHANNEL 1 BIT
    DEFINE CCP2_REG PORTC   ' CHANNEL 2 PORT
    DEFINE CCP2_BIT 1       ' CHANNEL 2 BIT
    TRISC.1 = 0             ' SET PORTC.2 TO OUTPUT (CCP1)
    TRISC.2 = 0             ' SET PORTC.2 TO OUTPUT (CCP1)
    DUTY    VAR BYTE        ' CREATE A VARIABLE FOR DUTY CYCLE

    WHILE 1 = 1            ' CREATE AN INFINITE LOOP
' INCREASE LED 1 WHILE DECREASING LED 2 IN BRIGHTNESS
    FOR DUTY = 0 TO 255            ' CYCLE THE FULL RANGE 0 TO 255
        HPWM 1,DUTY,2000          ' PWM ON CCP1 (BIT-2 OF PORTC)
        HPWM 2,(255 - DUTY),2000  ' PWM ON CCP2 (BIT-1 OF PORTC)
        PAUSE 10                  ' TIME TO VIEW
    NEXT

' DECREASE LED 1 WHILE INCREASING LED 2 IN BRIGHTNESS
    FOR DUTY = 255 TO 0 STEP -1   ' CYCLE THE FULL RANGE 255 TO 0
        HPWM 1,DUTY,2000          ' PWM ON CCP1 (BIT-2 OF PORTC)
        HPWM 2,(255 - DUTY),2000  ' PWM ON CCP2 (BIT-1 OF PORTC)
        PAUSE 10                  ' TIME TO VIEW
    NEXT
    WEND                          ' DO IT FOREVER
```

APPENDIX A

## Ho to set-up Micro Code Studio for use with the AMICUS18.

The AMICUS18 board comes pr-installed with a boot-loader. The easiest way to take advantage of the boot-loader is to install the AMICUS18 compiler. The compiler can be found at the myamicus web site.

Once the Amicus18 compiler is installed start Micro Code Studio.

From the tool-bar select "View", from the drop down click "Compile and Program Options". Then select the the "Programmer" tab. Select "Add New Programmer" and check the "Create a custom programmer entry".
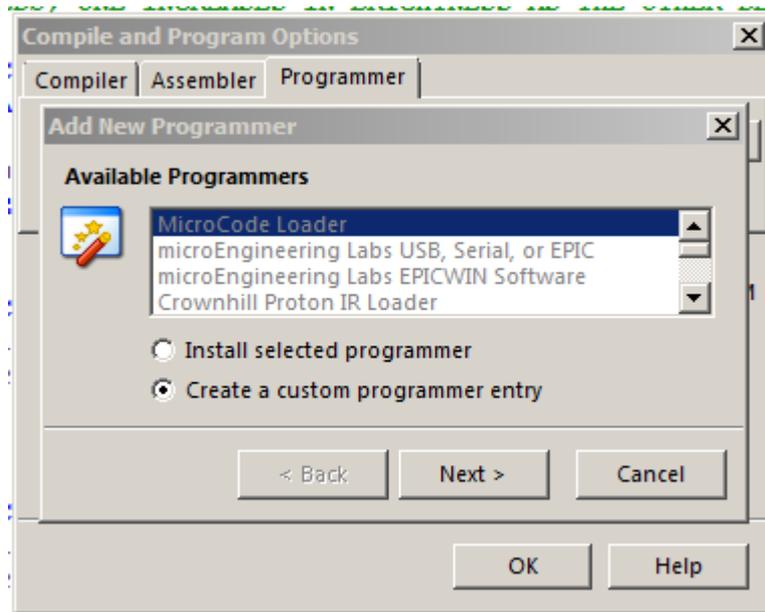
In the Display Name field enter "amicus18_loader", click "Next".
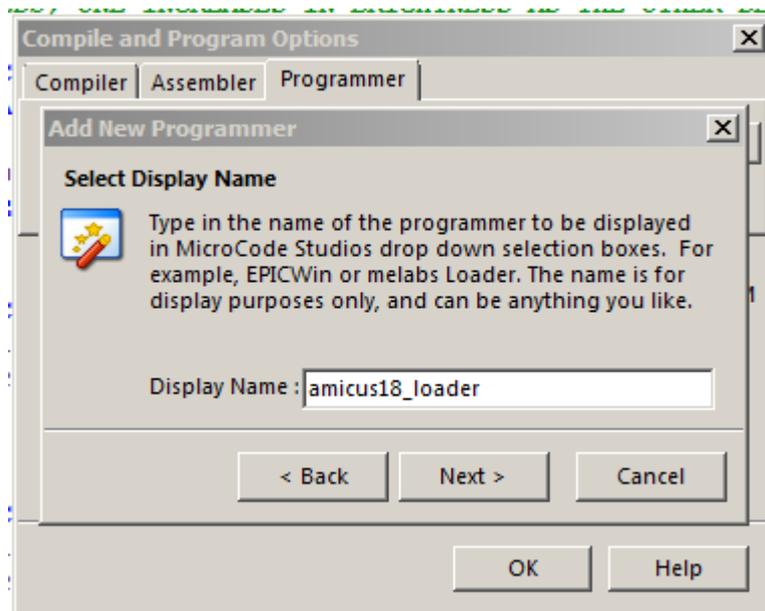
In the Programmer Filename field enter "amicus18_loader.exe". Click "Next".
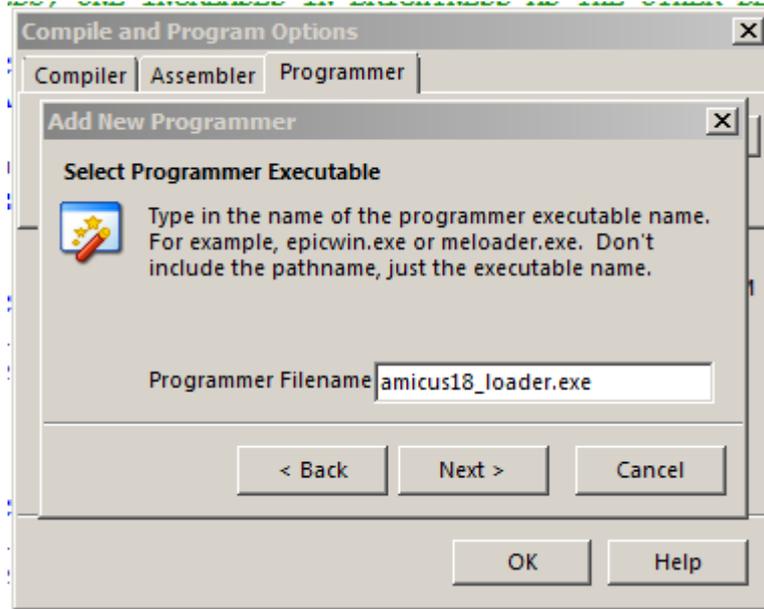
The next screen click "Find Manually". If you installed the Amicus18 compiler to the default location the path should be:
"C:\Program Files\AmicusIDE\Includes".
Click "Next".
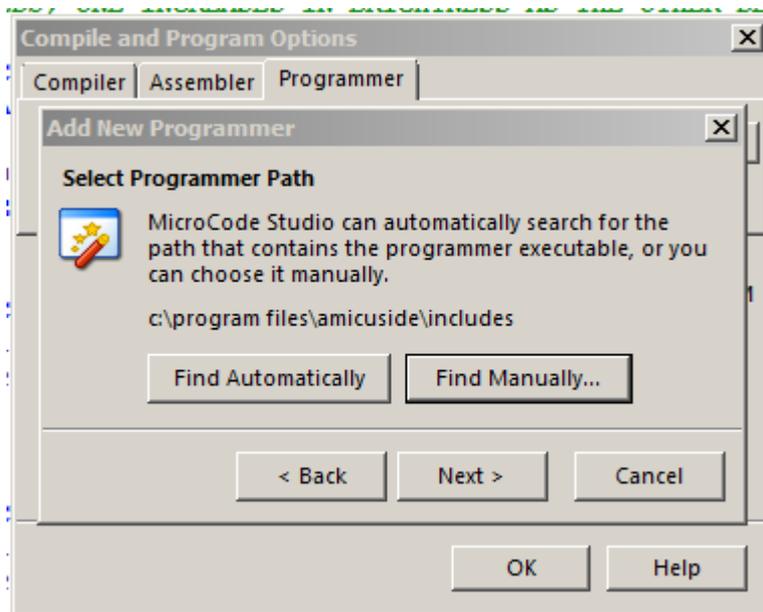
Finally the parameters are set. Enter "$long-hex-filename$ -iprogram". Then click "Finished"
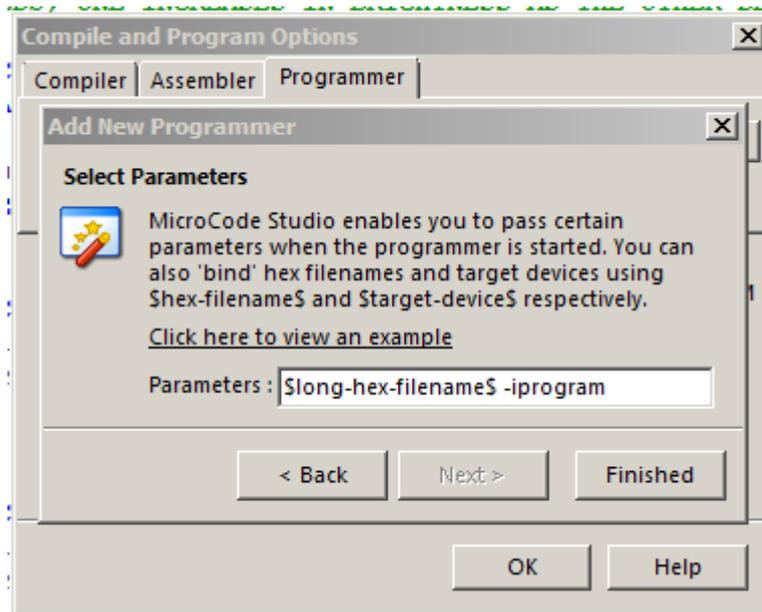
Now you can use the "F10" key from Micro Code Studio or click the "Compile Program" button to send your code to the Amicus18.