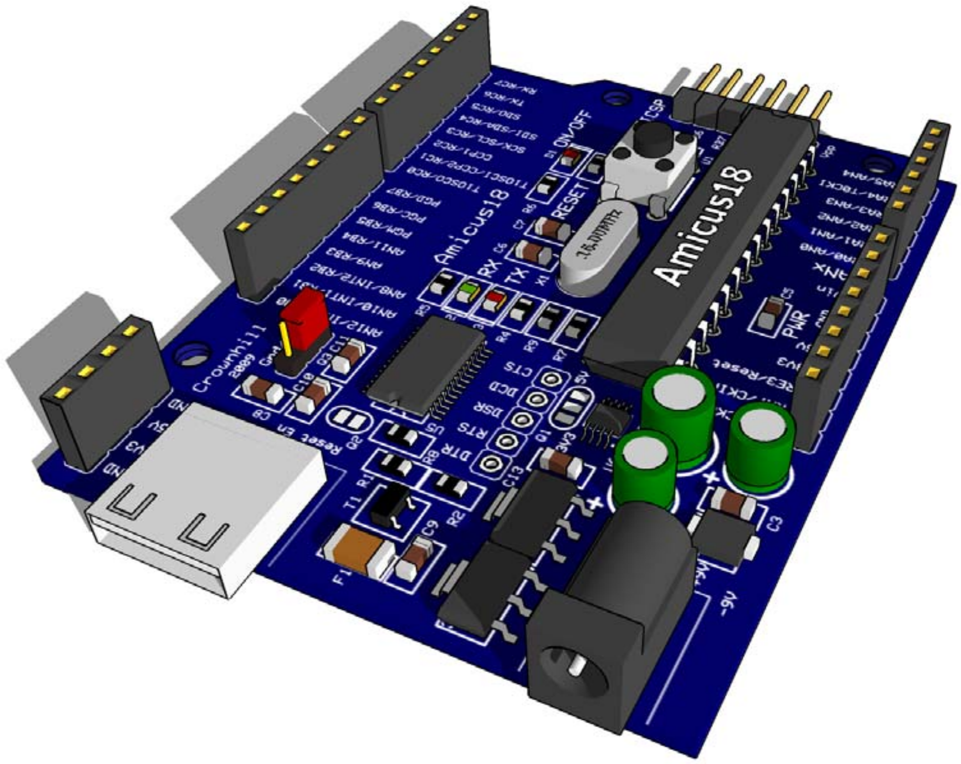
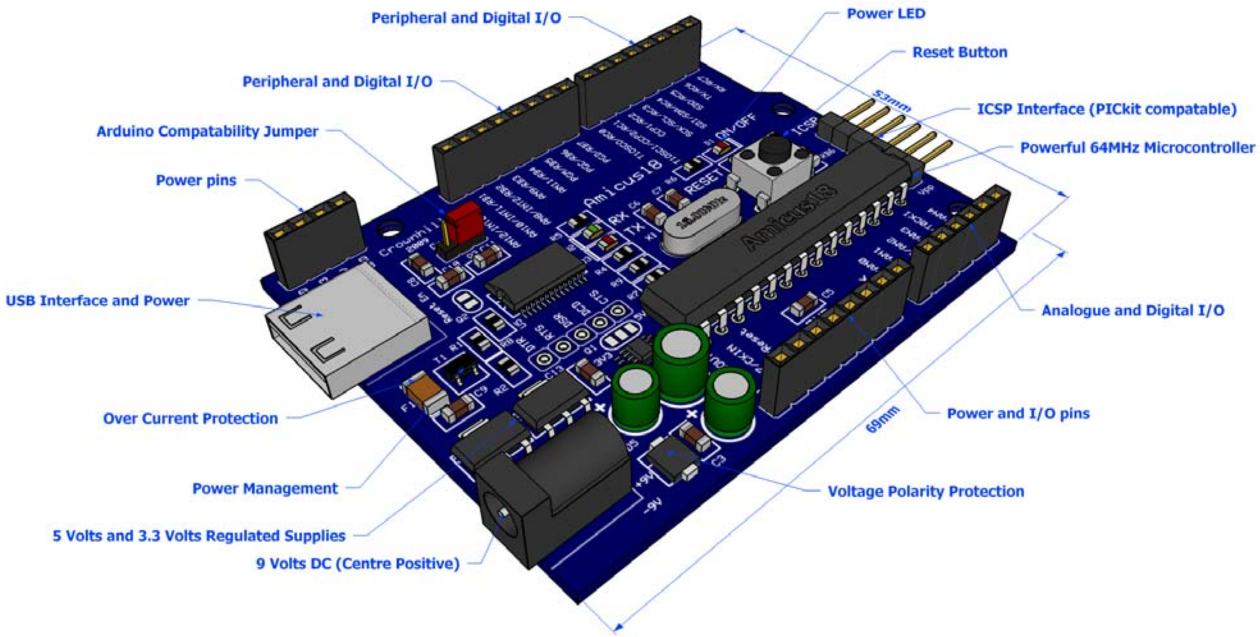
Using Amicus18 Hardware With Proton



Using Amicus18 Hardware with the Proton compiler

Amicus18 Hardware Overview

The Amicus18 hardware is based upon the world famous Arduino board, however, the Amicus18 board uses a Microchip PIC® microcontroller instead of an Atmel AVR type.



It has exactly the same dimensions as the Arduino, and all Arduino shields will physically fit on the Amicus18 board.

The microcontroller used on the Amicus18 is the Microchip PIC18F25K20, or the PIC18F25K22 which each have 32768 bytes of flash memory, 1536 bytes of RAM, and operate at 64MHz, which equates to 16 MIPS (Million Instructions per Second).

There are up to eleven 10-bit ADC (Analogue to Digital Converter) inputs, and two 10-bit PWM (Pulse Width Modulation) outputs, as well as comparators, USARTs (Universal Synchronous Asynchronous Receiver Transmitter), SPI (Serial Peripheral Interface), I^2C (Inter-Integrated Circuit), and up to six timers, each with various internal operations attached to them.

Each of the microcontroller's I/O lines are brought out for use with external devices such as LEDs, Servos, Potentiometers, LCDs etc...

Communication with the Amicus18 board is through a USB interface, which presents itself as a standard serial port on the PC. The microcontroller can be programmed directly through this port so there is no need for a dedicated device programmer, however, if the need arises, there is an ICSP (In Circuit Serial Programming) interface suitable for all programmers, but tailored for the Microchip PICkit2™ programmer.

Power can be supplied to the board either via the USB port, or an external 9 Volt DC source. When powered from the USB port, a maximum of 500mA (milliAmp) may be drawn, and the USB port is protected by a resetable fuse. When powered via a 9V source, a maximum of 800mA may be drawn.

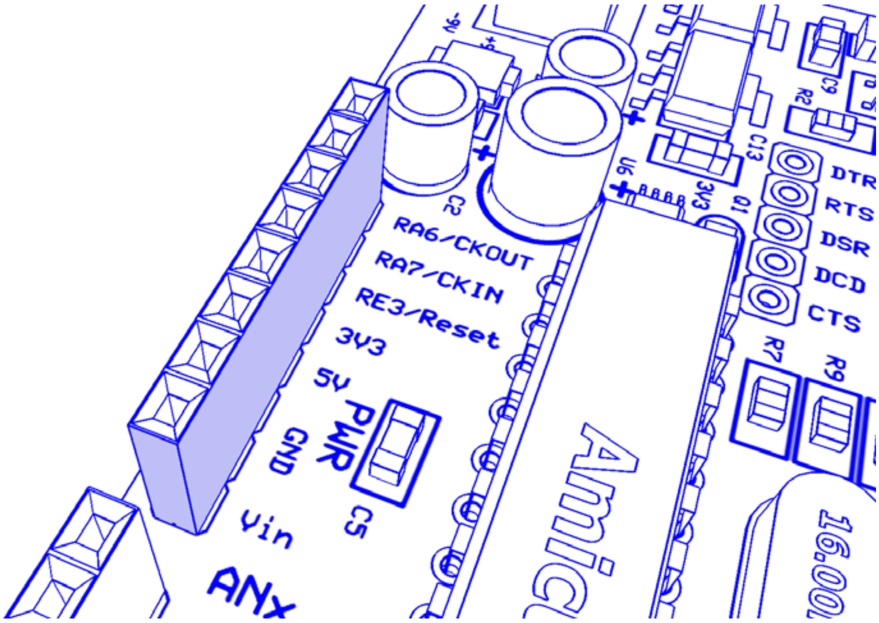
The PIC18F25K20 microcontroller is a 3.3 volts type, while the PIC18F25K22 will operate with both 3.3 volts and 5 volts.

The Amicus18 board is extremely easy to use, in fact, no previous microcontroller experience is required in order to get your first project up and running, as you'll find out later.

Amicus18 Sockets

As mentioned earlier, each of the microcontroller's I/O lines is brought to the outside world via 2.54mm (0.1") SIL sockets on the Amicus18 board. The operation of each block of pins is outlined below:

The 8-pin Power header socket:



- RA6 which is bit-6 of PortA. This pin defaults to the Clock Output Pin where the crystal is connected. It may be used as an I/O pin only when an internal oscillator setting is chosen.

- RA7 which is bit-7 of PortA. This pin defaults to the Clock Input Pin where the crystal is connected. It may be used as an I/O pin only when an internal oscillator setting is chosen.

- Microcontroller's reset line, which also acts as bit-3 of PortE (RE3), and is also the voltage input for a device programmer such as the PICkit2™ or the PICkit3™.

- 3.3 Volts output. 500mA when powered via USB, or 800mA when powered by an external 9 Volts source.

- 5 Volts output. 500mA when powered via USB, or 800mA when powered by an external 9 Volts source.

- Ground (0 Volts).

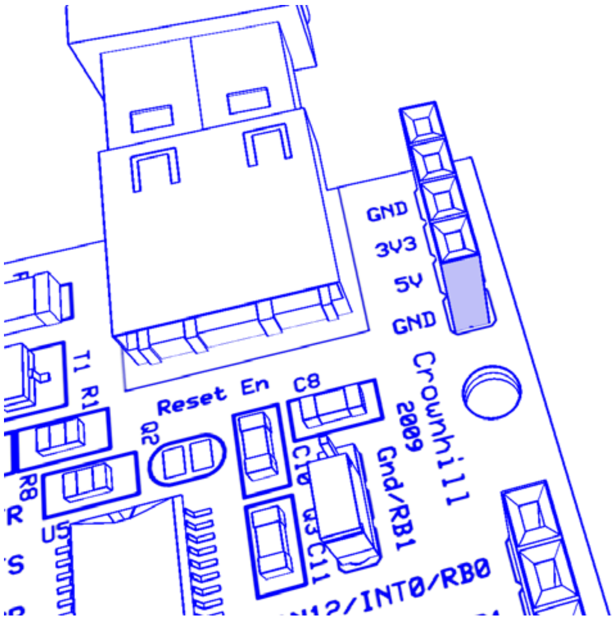
- DC 9 Volts input. This may be used to power the board.

The 4-pin Power header socket:

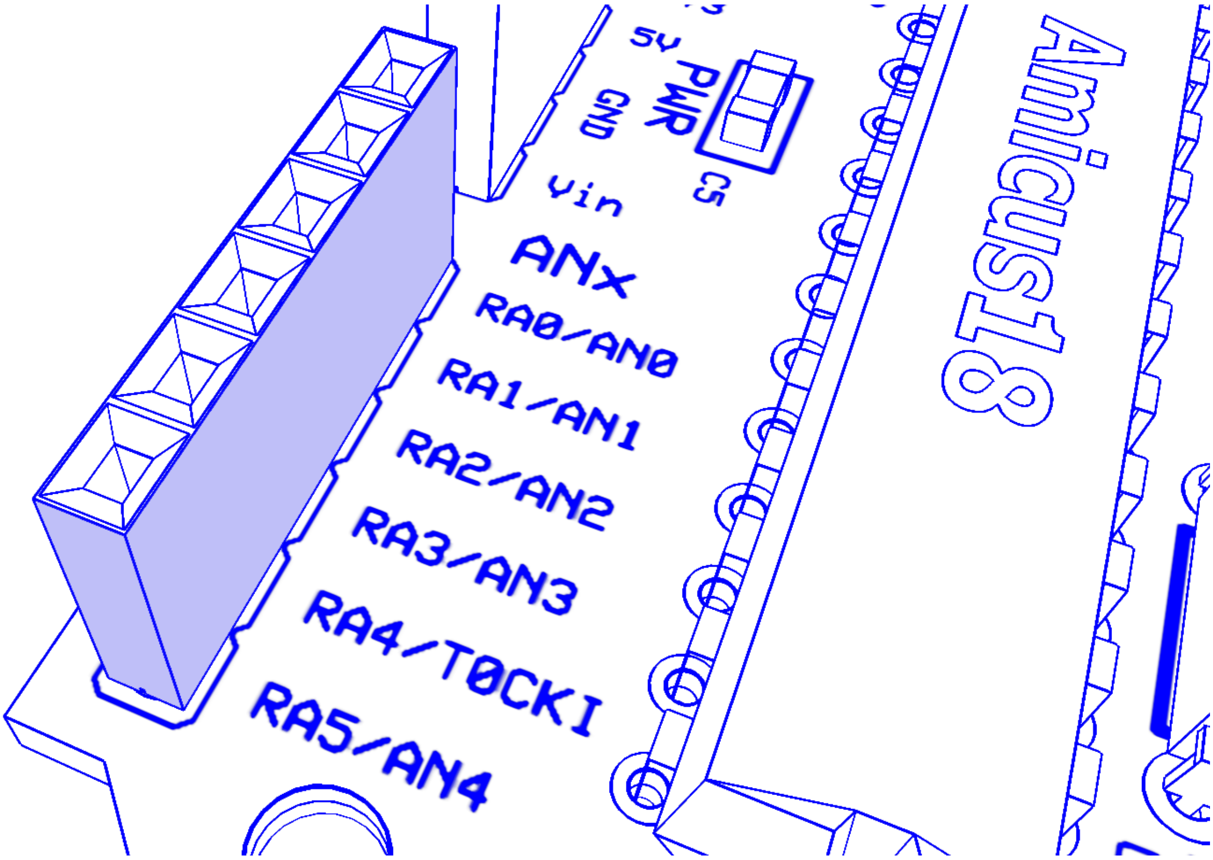
- Ground (0 Volts)

- 3.3 Volts output. 500mA when powered via USB, or 800mA when powered by an external 9 Volts source.

- 5 Volts output. 500mA when powered via USB, or 800mA when powered by an external 9 Volts source.



The PortA (Anx) socket:



- RA0 which is bit-0 of digital PortA. This pin can also be configured as Input 0 (AN0) of the 10-bit ADC (*Analogue to Digital Converter*). It can also be configured as the negative (-) input pin to either Comparator 1 or 2.

- RA1 which is bit-1 of digital PortA. This pin can also be configured as Input 1 (AN1) of the 10-bit ADC (*Analogue to Digital Converter*). It can also be configured as the negative (-) input pin to either Comparator 1 or 2.

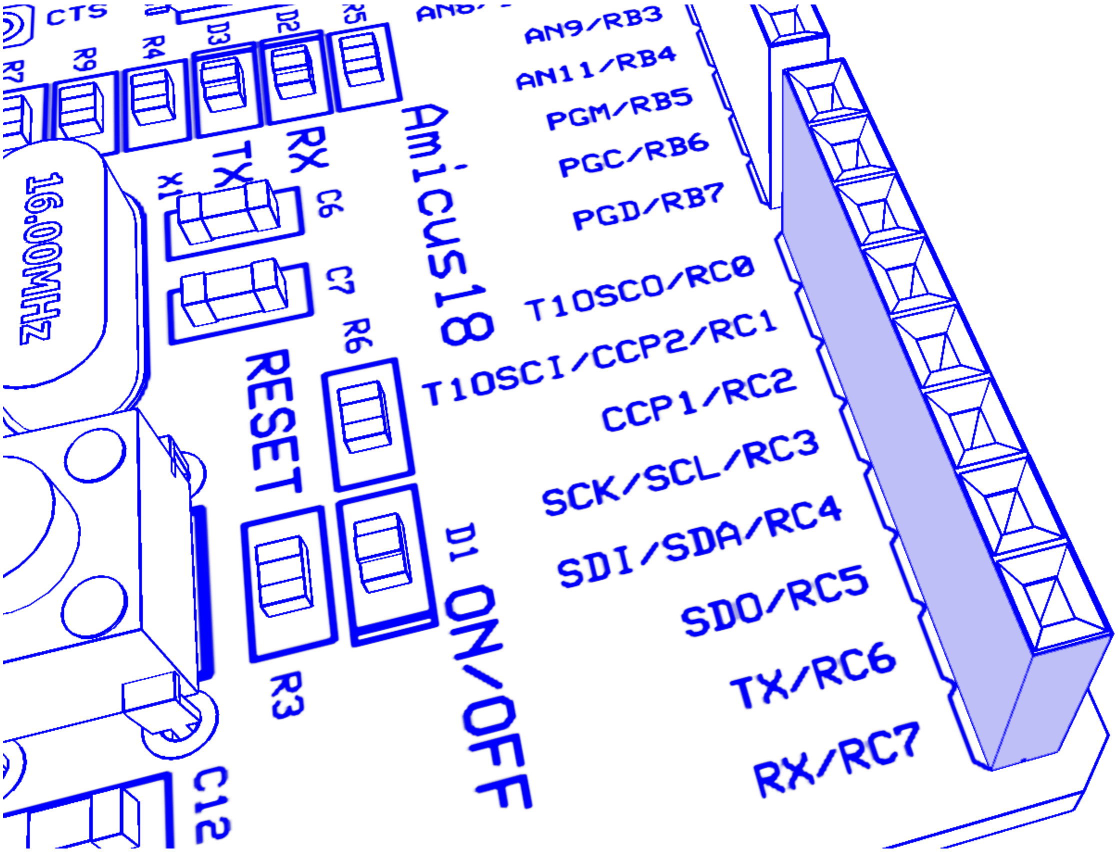
- RA2 which is bit-2 of digital PortA. This pin can also be configured as Input 2 (AN2) of the 10-bit ADC (*Analogue to Digital Converter*). It can also be configured as the positive (+) input pin to Comparator 2, or the output for the internal voltage reference.

- RA3 which is bit-3 of digital PortA. This pin can also be configured as Input 3 (AN3) of the 10-bit ADC (*Analogue to Digital Converter*). It can also be configured as the positive (+) input pin to Comparator 1.

- RA4 which is bit-4 of digital PortA. This pin can also be configured as the input trigger for Timer 0. It can also be configured as the output pin of Comparator 1.

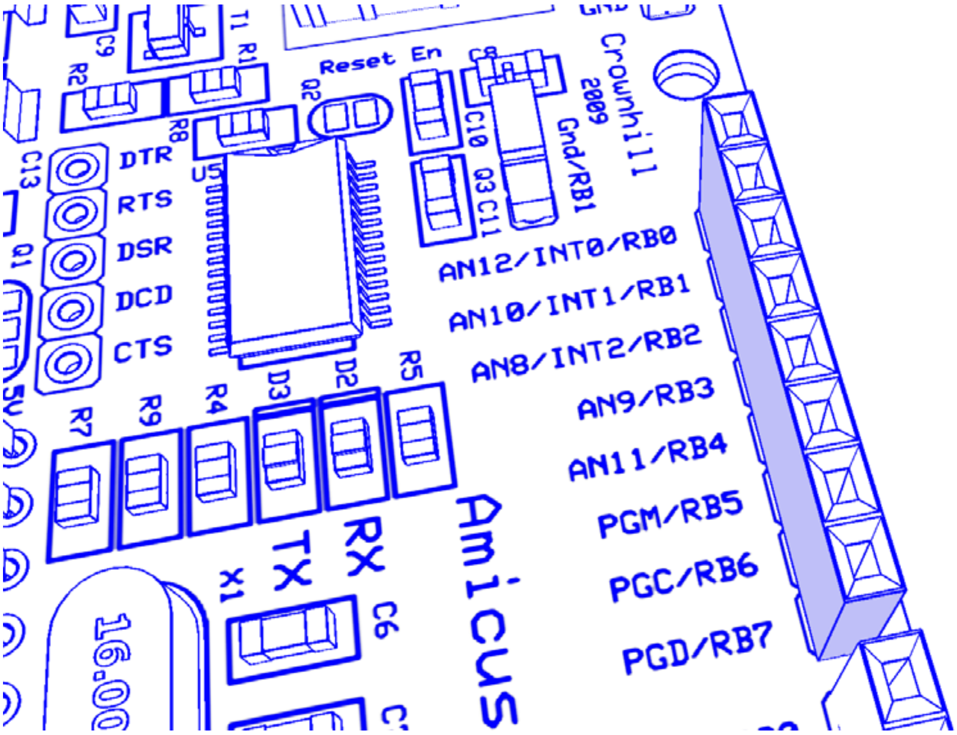
- RA5 which is bit-5 of digital PortA. This pin can also be configured as Input 4 (AN4) of the 10-bit ADC (*Analogue to Digital Converter*). It can also be configured as the output pin of Comparator 2.

The PortC socket:



- RC0 which is bit-0 of digital PortC. This pin can also be configured as the input for Timer 1.
- RC1 which is bit-1 of digital PortC. This pin can also be configured as the input for Timer 1, or a PWM (*Pulse Width Modulation*) output.
- RC2 which is bit-2 of digital PortC. This pin can also act as a PWM (*Pulse Width Modulation*) output.
- RC3 which is bit-3 of digital PortC. This pin can also be configured as the clock source for I^2C (*Inter-Integrated Circuit*) or SPI (*Serial Peripheral Interface*) communications.
- RC4 which is bit-4 of digital PortC. This pin can also be configured as the data source for I^2C (*Inter-Integrated Circuit*) or the data output for SPI (*Serial Peripheral Interface*) communications.
- RC5 which is bit-5 of digital PortC. This pin can also be configured as the data input for SPI (*Serial Peripheral Interface*) communications.
- RC6 which is bit-6 of digital PortC. This pin can also be configured as the USART (*Universal Synchronous Asynchronous Receiver Transmitter*) output for serial communications.
- RC7 which is bit-7 of digital PortC. This pin can also be configured as the USART (*Universal Synchronous Asynchronous Receiver Transmitter*) input for serial communications.

The PortB socket:



- RB0 which is bit-0 of digital PortB. This pin can also be configured as input 12 (AN12) of the 10-bit ADC, or an external interrupt trigger.

- RB1 which is bit-1 of digital PortB. This pin can also be configured as input 10 (AN10) of the 10-bit ADC, or an external interrupt trigger.

- RB2 which is bit-2 of digital PortB. This pin can also be configured as input 8 (AN8) of the 10-bit ADC, or an external interrupt trigger.

- RB3 which is bit-3 of digital PortB. This pin can also be configured as input 9 (AN9) of the10-bit ADC, or an alternative PWM (*Pulse Width Modulation*) output.

- RB4 which is bit-4 of digital PortB. This pin can also be configured as input 11 (AN11) of the 10-bit ADC, or an external interrupt trigger.

- RB5 which is bit-5 of digital PortB. This pin can also be configured as an external interrupt trigger.

- RB6 which is bit-6 of digital PortB. This pin can also be configured as an external interrupt trigger, and is also the clock line for a device programmer such as the PICkit2™ or the PICkit3™.

- RB7 which is bit-7 of digital PortB. This pin can also be configured as an external interrupt trigger, and is also the data line for a device programmer such as the PICkit2™ or the PICkit3™.

Each pin of the microcontroller is capable of sourcing or sinking 25mA, with a maximum of 100mA per port.

The microcontroller's architecture is very versatile, allowing several internal peripherals to share the same pin, thus maximising the flexibility, but keeping the size of the device small. Each internal peripheral can be enabled, disabled and configured very easily from within the free BASIC compiler environment.

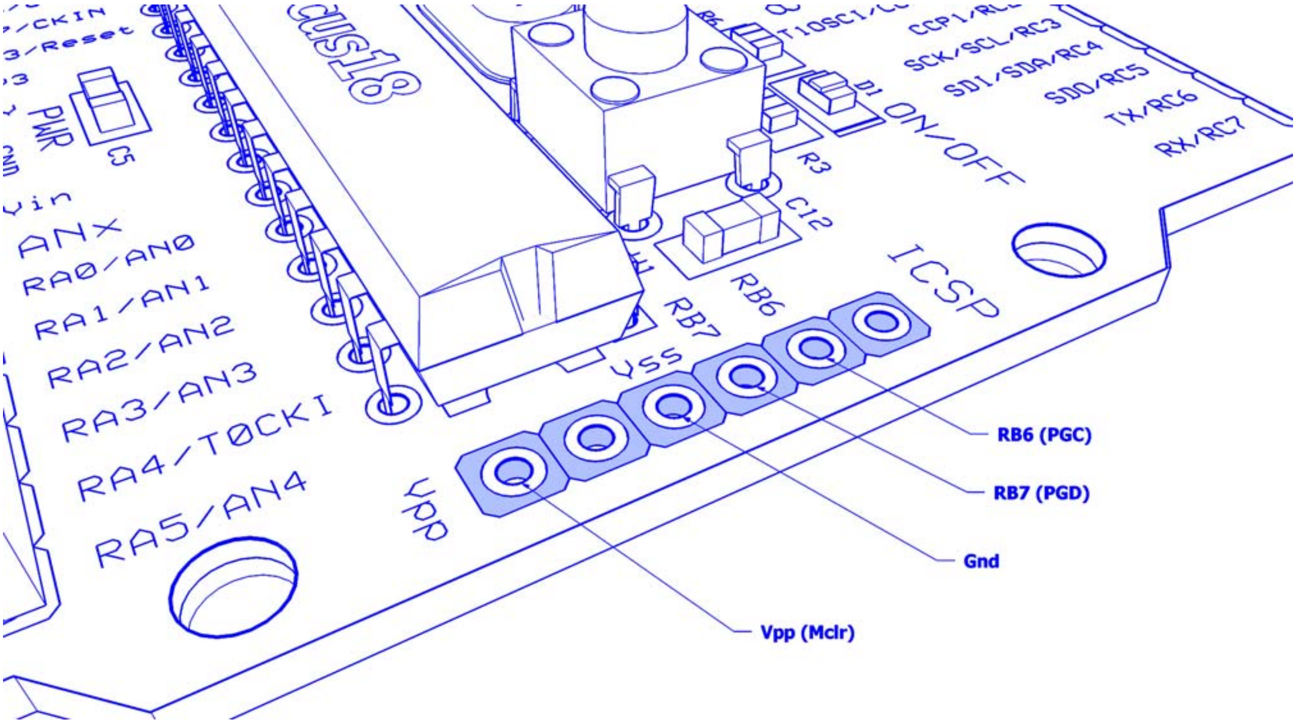
Although the microcontroller has a 3.3 Volts operating voltage, all I/O pins are 5 Volt tolerant.

Device Programming Header

The Amicus18 board has the ability to be programmed in circuit. This bypasses the built in bootloader, and indeed, will overwrite it.

The header has been designed for a PICkit2™ or PICkit3™ programmer to fit straight onto it, however, any other device programmer may be used with a suitable adapter. It must be remembered that the microcontroller is a 3.3 Volt PIC18F25K20 type, therefore if a programmer other than a PICkit2™ or a PICkit3™ is used, ensure that it supports this device, as a 5 Volt only programmer will damage the microcontroller.

The programming header's location is shown below:

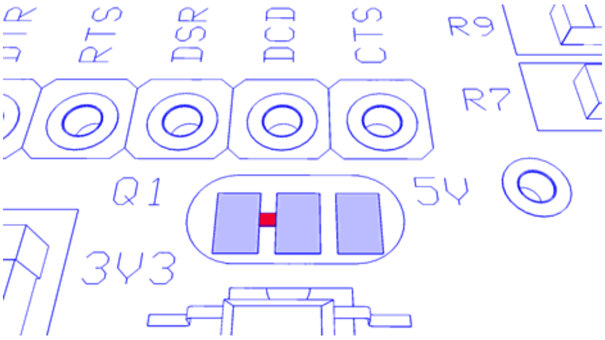


Jumper and Pad Settings

The Amicus18 board has a jumper and two pads that can alter it's characteristics.

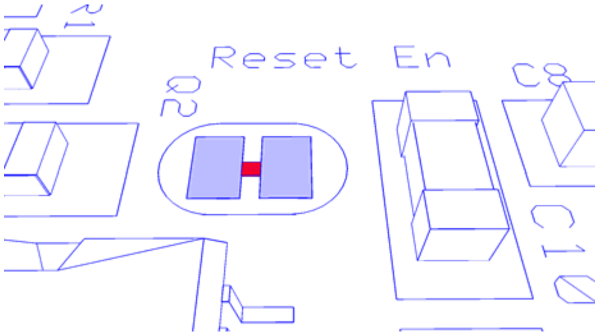
Pad Q1

This allows a 5 Volts type microcontroller to be used with the board instead of the supplied 3.3 Volt type.



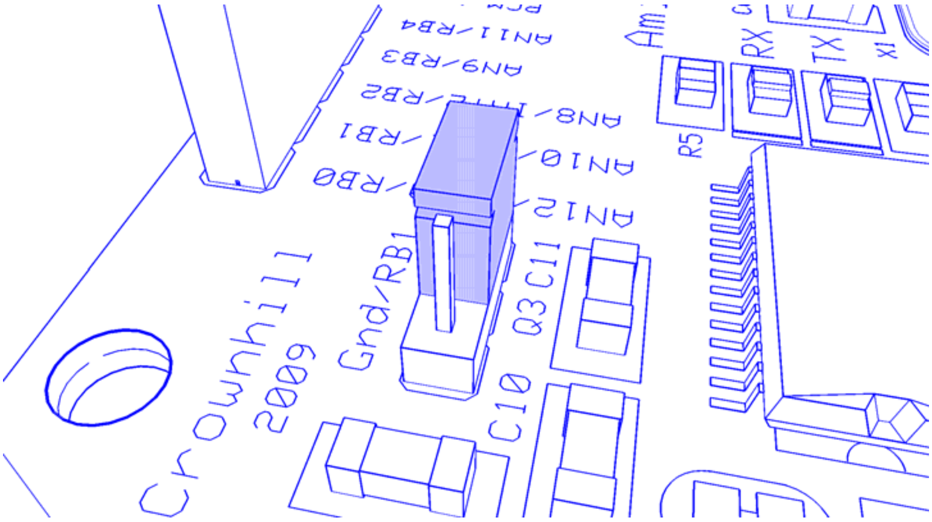
Pad Q2

This allows disconnection of the internal Reset for the microcontroller from the USB bootloader.



Jumper Q3

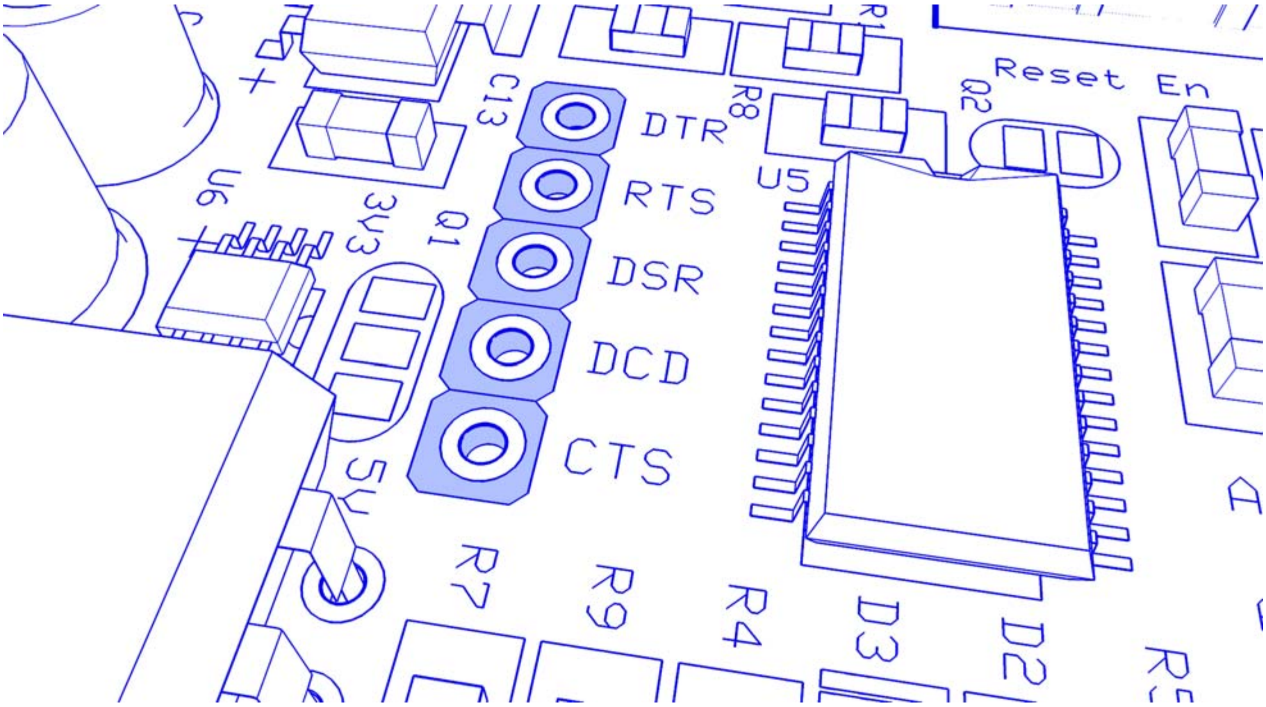
This allows maximum compatibility with existing Arduino shields. The PIC18F25K20 and PIC25K22 microcontrollers have more I/O lines than that of an Atmel, therefore, two of the pins on the PortB socket operate differently on the Amicus18. RB1 is a Ground pin on the Arduino board, but this would waste a valuable I/O pin if it were simply grounded. Instead, Jumper Q3 can be configured for RB1 or Ground.



 Version 1.0 06-10-2009

Serial Handshake Connections

The USB to serial device also emulates the handshaking lines of a conventional serial port. These are shown below:



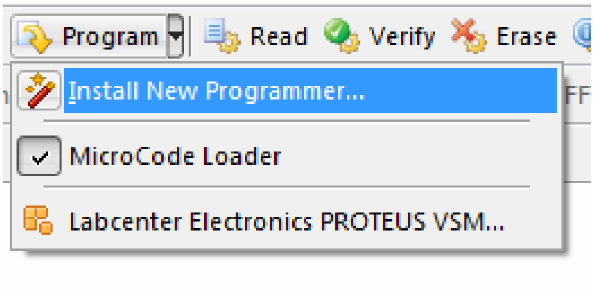
The Amicus18 board uses the DTR line in-order to reset the microcontroller, however, the other lines are available to use. The direction of each line is shown below:

- DTR This is an output from the PC to the Amicus18 board.
- RTS This is an output from the PC to the Amicus18 board.
- DSR This is an input to the PC from the Amicus18 board.
- DCD This is an input to the PC from the Amicus18 board.
- CTS This is an input to the PC from the Amicus18 board.

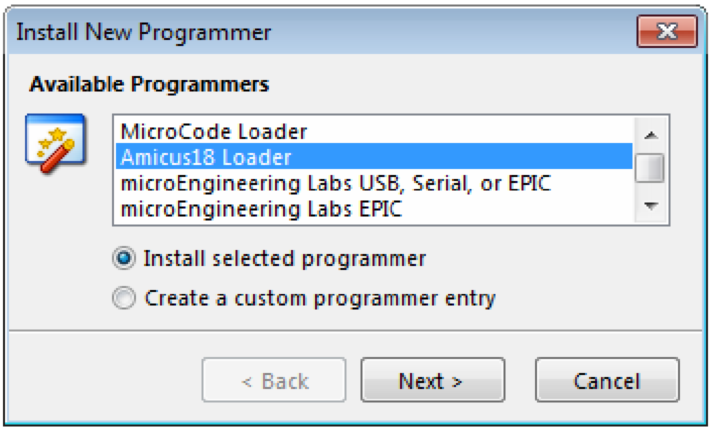
Using the Proton Compiler with the Amicus18 board

Configuring the Proton compiler to work with the Amicus18 board is simplicity itself, as all the applications required are installed along with the compiler.

The Amicus18 board's microcontroller has a built-in bootloader, so first we'll choose the correct bootloader from within the Proton IDE. On the toolbar, Click the small arrow on the Program button:

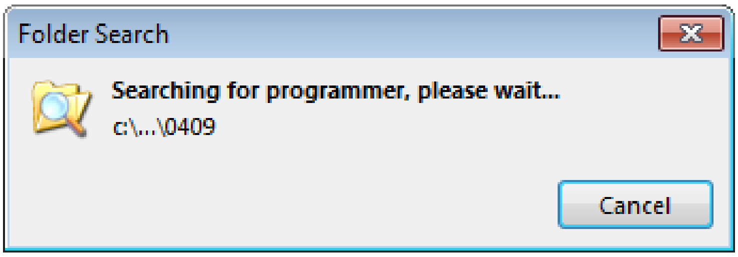


Choose the option "Install New Programmer" and a window will open:

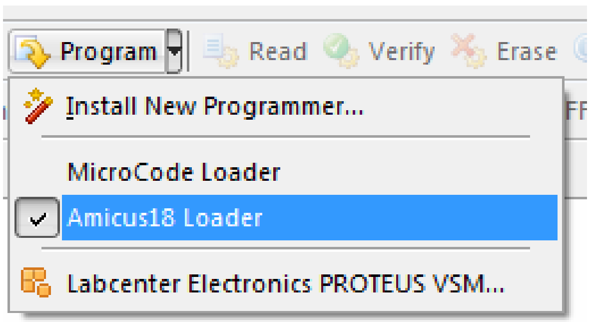


Choose the Amicus18 Loader option and click Next.

The bootloader's executable will then be searched for:



Once it has been found the window will disappear and the job is done. In order to verify that the Amicus18 bootloader has been allocated correctly, click the downward arrow on the program button again:



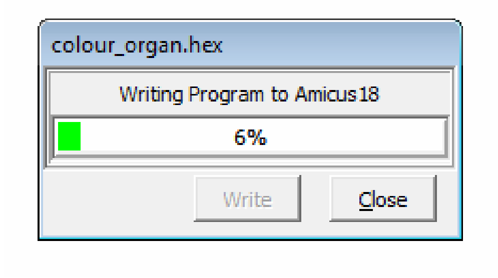
Writing your first Amicus18 program using the Proton compiler

Here's a very small sample of the Proton BASIC language:

```
' Flash an LED connected to RB0
  Include "Amicus18.inc"  ' Configure the compiler to use the Amicus18 board
  While 1 = 1             ' Create an infinite loop
    High PORTB.0          ' Bring the LED pin high (illuminate the LED)
    DelayMs 500           ' Wait 500ms (half a second)
    Low PORTB.0           ' Pull the LED pin low (Extinguish the LED)
    DelayMs 500           ' Wait 500ms (half a second)
  Wend                    ' Close the loop
```

As can be seen, the language is very simple to understand, but has a powerful command set, and produces true assembler code that talks to the microcontroller directly.

Click the toolbar button ***Compile and Program***, and watch as the compiler takes over automatically. The program will be compiled and if there are no syntax errors, the bootloader will be invoked, which will automatically locate the Amicus18 board connected to USB and program its microcontroller:

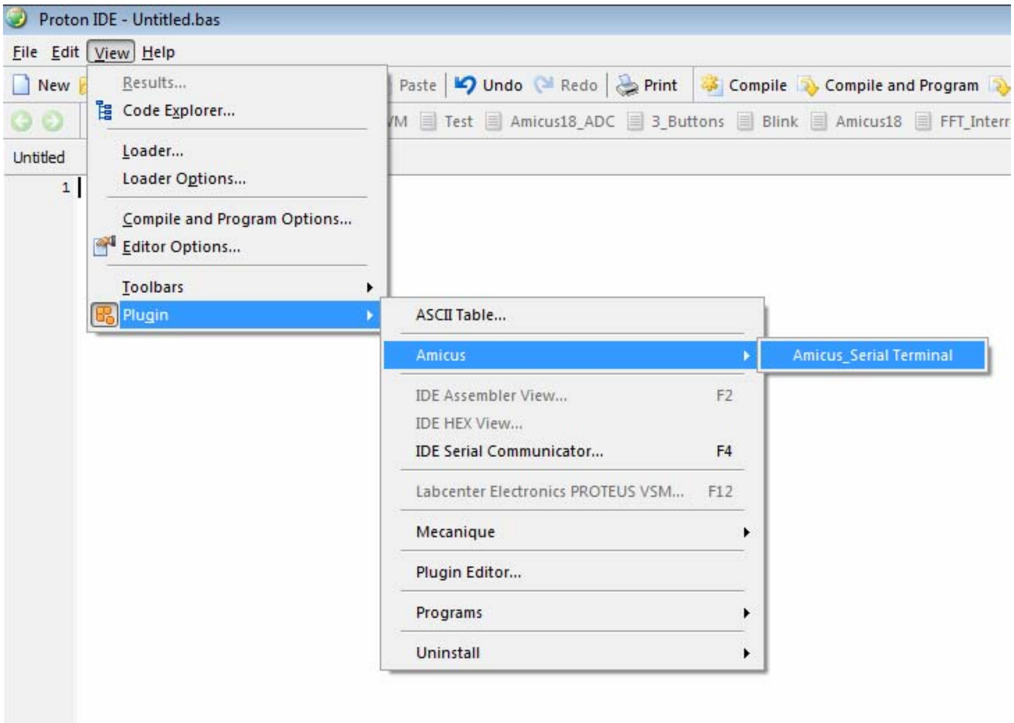


Here's a slightly more complex program:

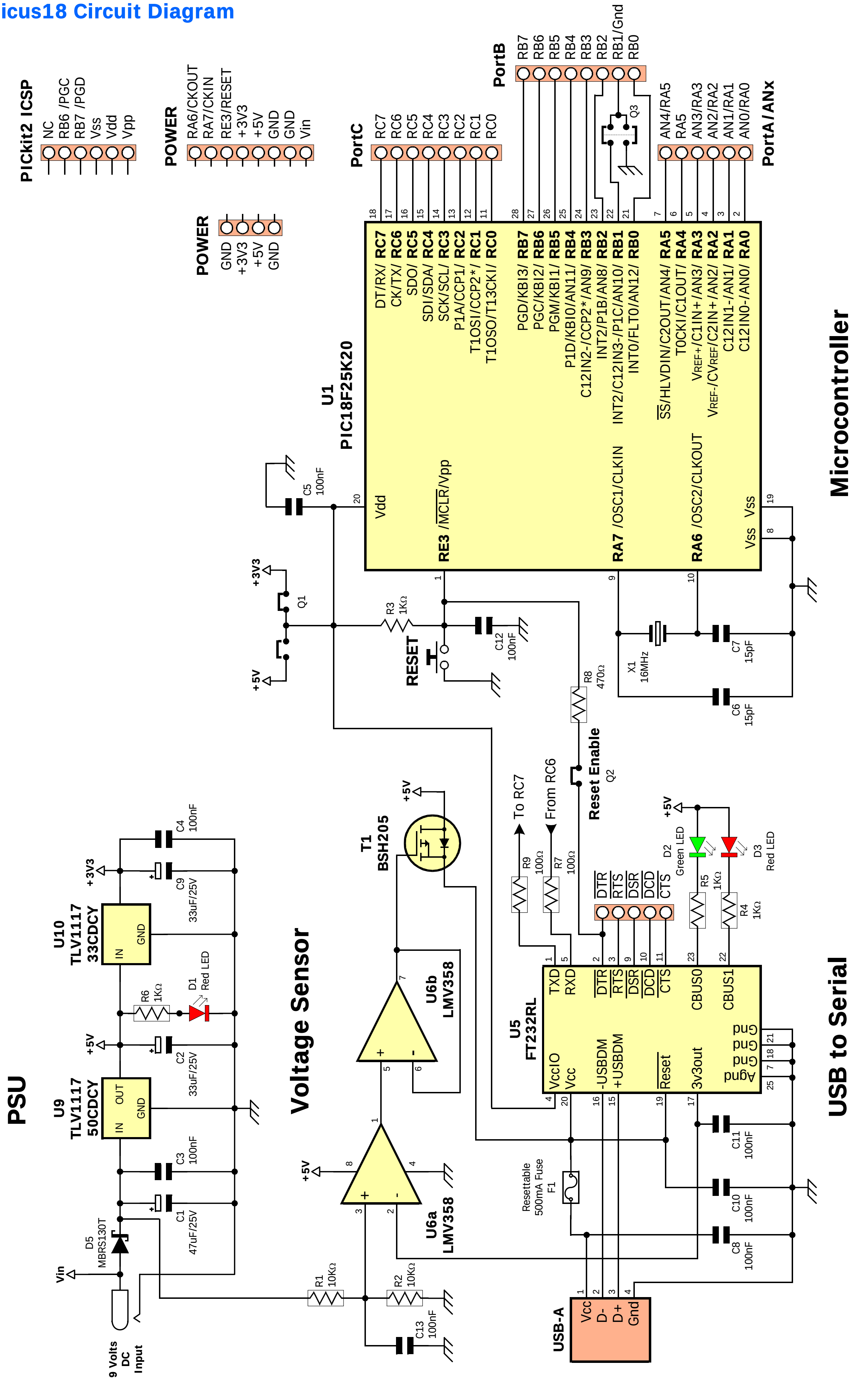
```
' Pulse both LEDs, one decreases while the other increases brightness
'
  Include "Amicus18.inc"         ' Configure the compiler to use the Amicus18 board
  Include "Amicus18_Hpwm10.inc"  ' Load the Amicus18 10-bit PWM macros into program

  Dim wDutyCycle As Word         ' Holds the duty cycle of the PWM pulses
  While 1 = 1                    ' Create an infinite loop
'
' Increase LED1 illumination, while decreasing LED2 illumination
'
    For wDutyCycle = 0 To 1023          ' Cycle the full range of 10-bits
      WriteAnalog1(wDutyCycle)          ' PWM on CCP1 (Bit-2 of PortC) (0 to 1023)
      WriteAnalog2(1023 - wDutyCycle)   ' PWM on CCP2 (Bit-1 of PortC) (1023 to 0)
      DelayMS 5                         ' A small delay between duty cycle changes
    Next                                ' Close the loop
    DelayMS 5
'
' Decrease LED1 illumination, while increasing LED2 illumination
'
    For wDutyCycle = 1023 To 0 Step -1  ' Cycle the full 10-bit range (reversed)
      WriteAnalog1(wDutyCycle)          ' PWM on CCP1 (Bit-2 of PortC) (1023 to 0)
      WriteAnalog2(1023 - wDutyCycle)   ' PWM on CCP2 (Bit-1 of PortC) (0 to 1023)
      DelayMS 5                         ' A small delay between duty cycle changes
    Next                                ' Close the loop
  Wend                                  ' Do it forever
```

The Amicus18 has its own serial terminal application that has some features specially developed for it.

This can be located by clicking on the IDE's View->Plugin menu option:

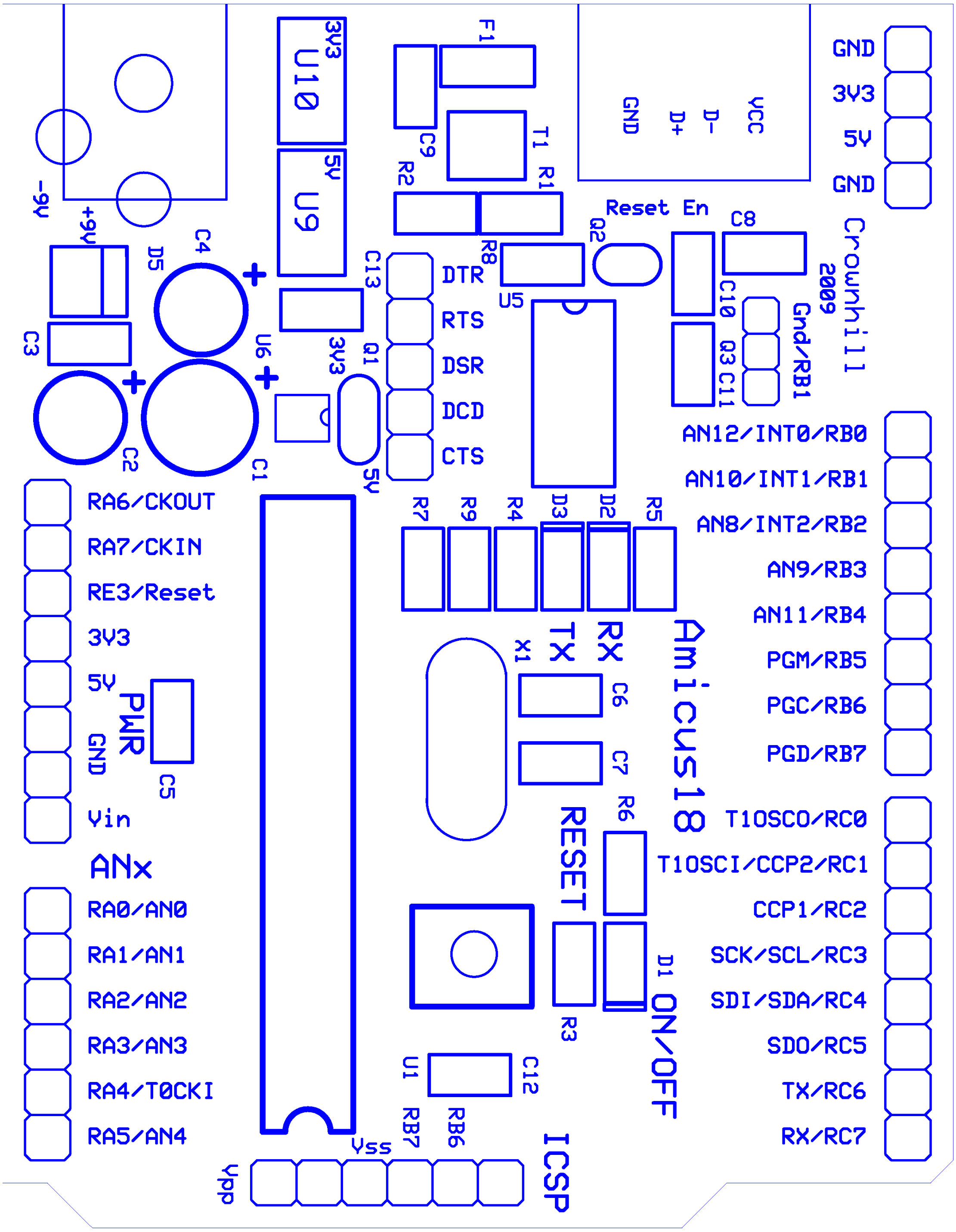


Amicus18 Circuit Diagram



 Version 1.0 06-10-2009

Amicus18 PCB Layout



 Version 1.0 06-10-2009

Installing the Amicus18 USB Driver

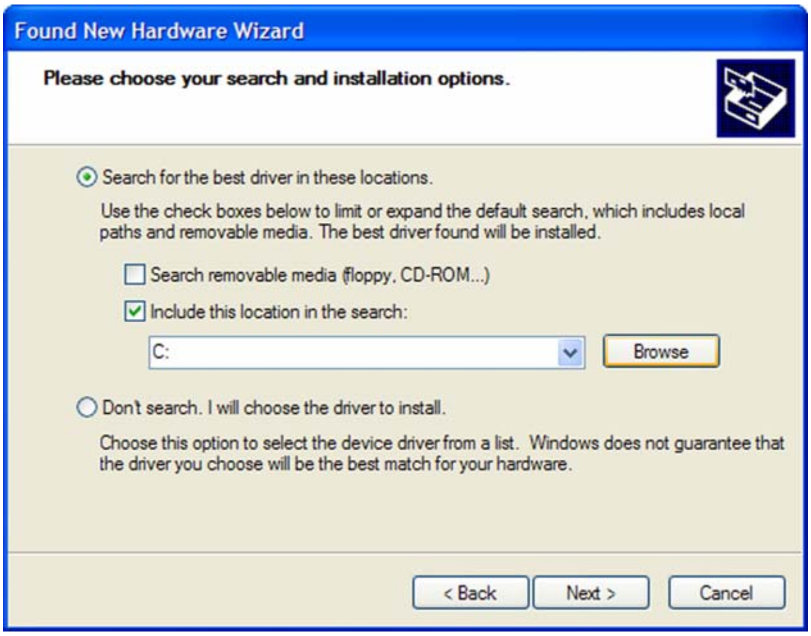
The Amicus18 board uses an FTDI serial to USB device, which presents itself as a standard com port on the PC. However, this requires USB drivers to be installed the first time the Amicus18 board is connected to your computer. This is a simple process and a step by step guide is outlined below for a Windows XP system. Note that Vista systems use the same principle, only windows and dialogues will change:

Plug the USB cable into a free USB port on the PC, and then into the Amicus18's USB port.

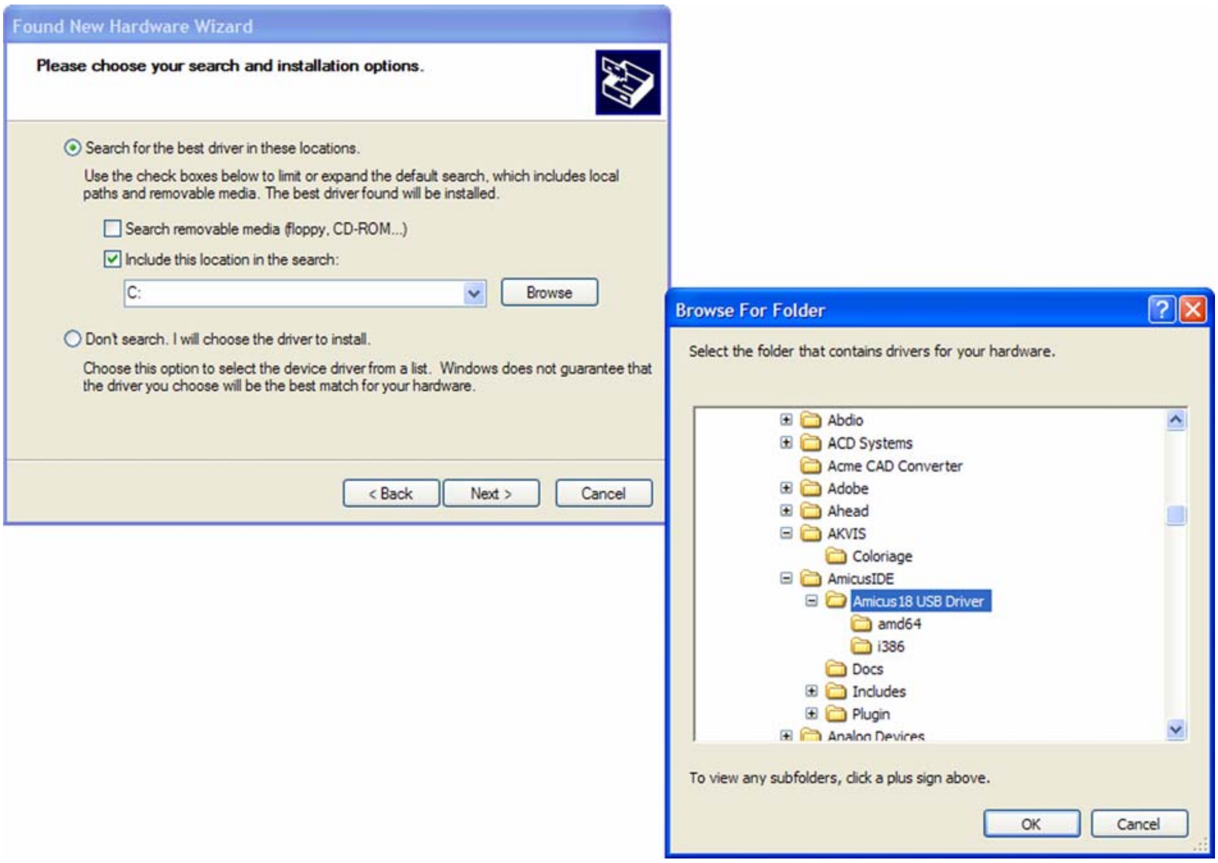
Note. Make sure you plug the Amicus18 board into a powered USB HUB or direct to the PC's USB port, as un-powered HUBs can only supply 100mA of power, instead of 500mA for powered HUBs.

The first window will inform you that a new device has been found on the USB port:

Choose the option "**Install from a list or specific location**" and click ***Next***:

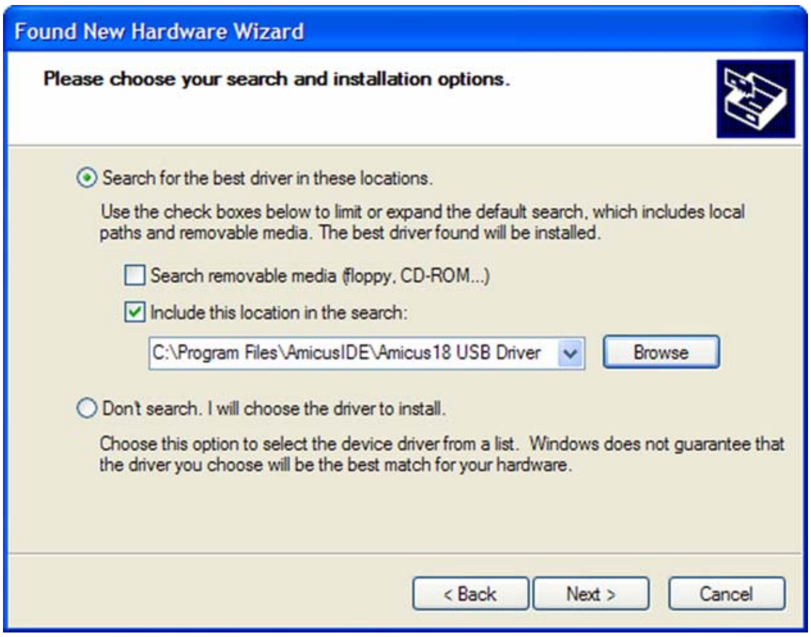


Make sure the options are ticked as in the previous window and click on the ***Browse*** button:



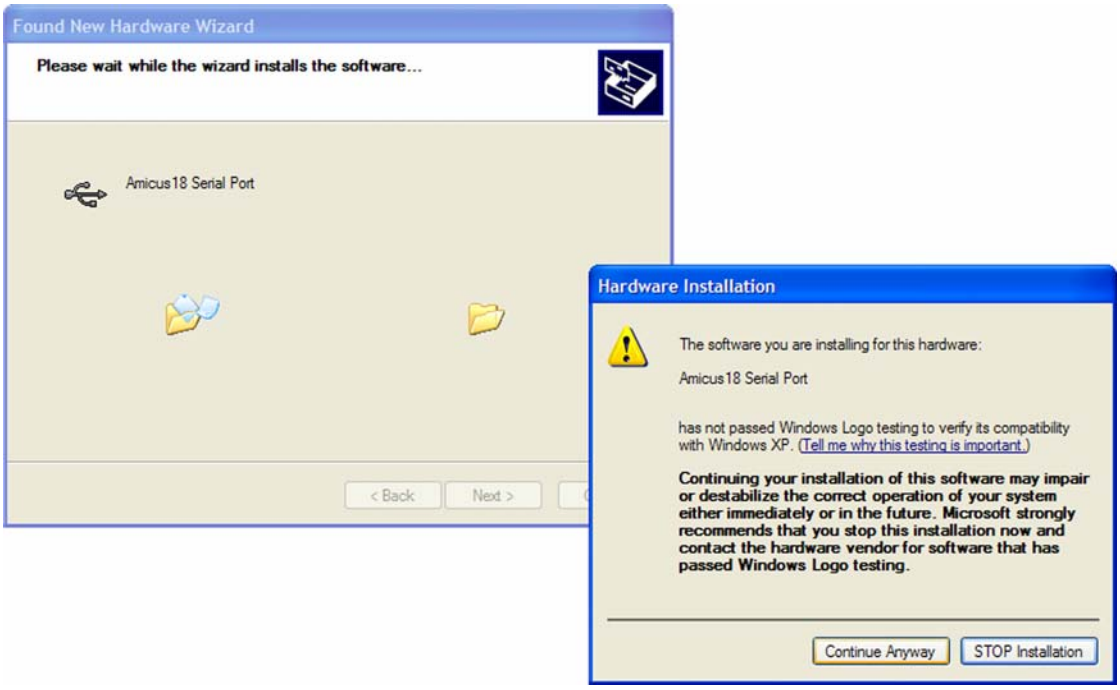
Navigate to the compiler's install path which it defaults to "**C:\Program Files\ProtonIDE**" , **"C:\Program Files (x86)\ProtonIDE"** for Windows7 64-bit, and choose the "**Amicus18 USB Driver**" folder. Click **OK**:

The windows should look like the image below:

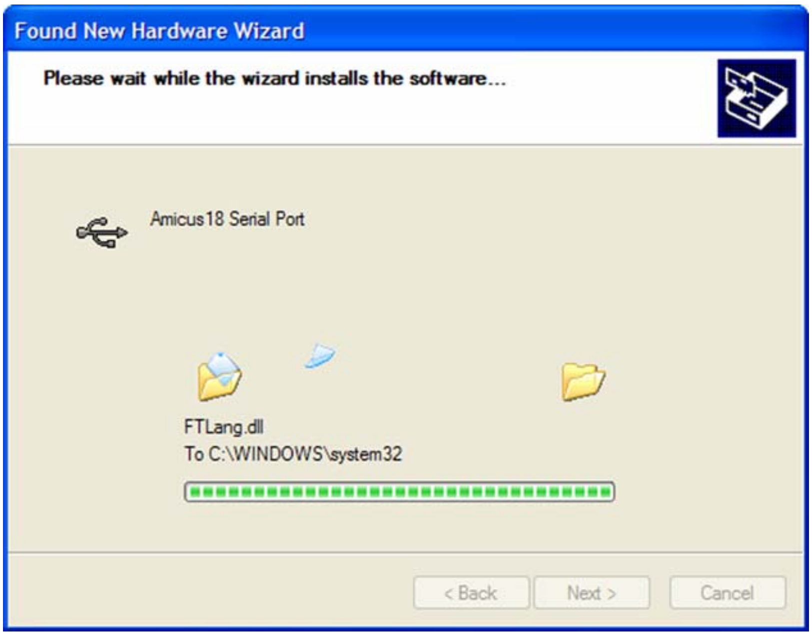


Click the ***Next*** button and the driver will begin to install.

You will see a windows message stating that the drivers have not been certified by Microsoft. This is quite normal and nothing to be worried about, just click the ***Continue Anyway*** button:

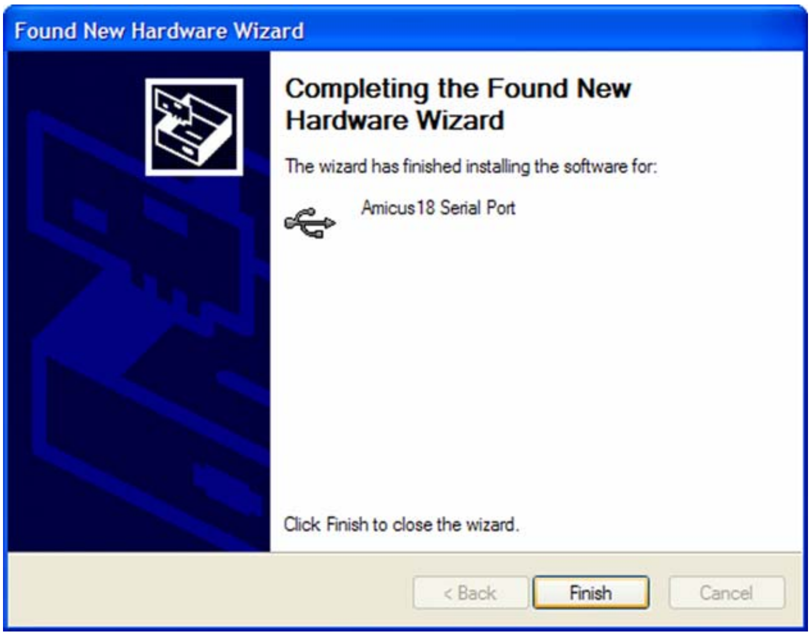


The driver will continue to install:



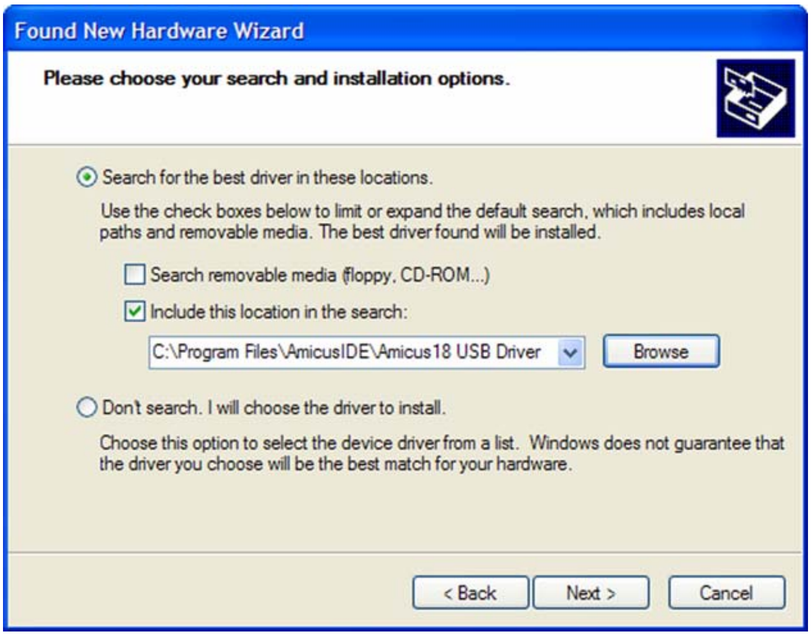
 Version 1.0 06-10-2009

Once the driver is complete it will show the window below:



Click on the ***Finish*** button.

Note that the above procedure will need to be carried out twice for the driver to be fully installed, however, the second time, the files will have already been located on the hard drive, so it may not be necessary to navigate to the driver folder:



The USB drivers are now installed and will not require re-doing, unless the Amicus board is inserted into a different USB port on the computer, in which case, choose the "**Install the software automatically**" option on the initial driver install window.

Built in Amicus18 Peripheral Macros

The compiler has several built-in macros for configuring the most popular peripheral modules contained with the Amicus18's microcontroller, these are the ADC (Analogue to Digital Converter), Timers, SPI (Serial Peripheral Interface),

ADC macros Introduction

The ADC (Analogue to Digital Converter) peripheral on the Amicus18 is supported with the following macros. The macros are a mixture of compiler types and preprocessor types, and can be found in "Includes\Sources\Amicus18_ADC.inc"

A/D Converter Macros

- BusyADC — Is A/D converter currently performing a conversion?
- CloseADC — Disable the A/D converter.
- ConvertADC — Start an A/D conversion.
- OpenADC — Configure the A/D converter.
- ReadADC — Read the results of an A/D conversion.
- SetChanADC — Select A/D channel to be used.
- SelChanConvADC — Select A/D channel to be used and start an A/D conversion.

BusyADC

Syntax
Variable = **BusyADC**()

Include file
Amicus18_ADC.inc

Overview
This macro indicates if the A/D peripheral is in the process of converting a value.

Return Value

- 1 if the A/D peripheral is performing a conversion.
- 0 if the A/D peripheral isn't performing a conversion.

CloseADC

Syntax
CloseADC()

Include file
Amicus18_ADC.inc

Overview
This macro disables the A/D converter and A/D interrupt mechanism.

ConvertADC

Syntax
ConvertADC()

Include file
Amicus18_ADC.inc

Overview
This macro starts an A/D conversion. The **BusyADC**() macro or A/D interrupt may be used to detect completion of the conversion. The result is held in registers ADRESL and ADRESH.

OpenADC

Syntax
OpenADC(pConfig, pConfig2, pPortConfig)

Include file
Amicus18_ADC.inc

Overview
This macro resets the A/D-related registers to the POR state and then Configures the clock, result format, voltage reference, port and channel.

Operators

- ***Pconfig*** A bitmask that is created by performing a bitwise AND operation ('&') with a value from each of the categories listed below. These values are defined in the file Amicus18_ADC.inc.

A/D clock source:

ADC_FOSC_2	Fosc / 2
ADC_FOSC_4	Fosc / 4
ADC_FOSC_8	Fosc / 8
ADC_FOSC_16	Fosc / 16
ADC_FOSC_32	Fosc / 32
ADC_FOSC_64	Fosc / 64
ADC_FOSC_RC	Internal RC Oscillator

A/D result justification:

ADC_RIGHT_JUST	Result in Least Significant bits (Used for 10-bit ADC result)
ADC_LEFT_JUST	Result in Most Significant bits (Used for 8-bit ADC result)

A/D acquisition time select:

ADC_0_TAD	0 Tad
ADC_2_TAD	2 Tad
ADC_4_TAD	4 Tad
ADC_6_TAD	6 Tad
ADC_8_TAD	8 Tad
ADC_12_TAD	12 Tad
ADC_16_TAD	16 Tad
ADC_20_TAD	20 Tad

- ***pConfig2*** A bitmask that is created by performing a bitwise AND operation ('&'), as shown in the example at the end of this document, with a value from each of the categories listed below. These values are defined in the file Amicus18_ADC.inc.

Channel:

ADC_CH0	Channel 0
ADC_CH1	Channel 1
ADC_CH2	Channel 2
ADC_CH3	Channel 3
ADC_CH4	Channel 4
ADC_CH5	Channel 5
ADC_CH6	Channel 6
ADC_CH7	Channel 7
ADC_CH8	Channel 8
ADC_CH9	Channel 9
ADC_CH10	Channel 10
ADC_CH11	Channel 11
ADC_CH12	Channel 12

A/D Vref+ and Vref- Configuration:

ADC_REF_VDD_VREFMINUS	VREF+ = VDD & VREF- = Ext.
ADC_REF_VREFPLUS_VREFMINUS	VREF+ = Ext. & VREF- = Ext.
ADC_REF_VREFPLUS_VSS	VREF+ = Ext. & VREF- = VSS
ADC_REF_VDD_VSS	VREF+ = VDD & VREF- = VSS

- ***pPortConfig*** The *pPortConfig* can have 8192 different combination, few are defined below:

ADC_0ANA	All digital	
ADC_1ANA	analogue:	AN0
ADC_2ANA	analogue:	AN0-AN1
ADC_3ANA	analogue:	AN0-AN2
ADC_4ANA	analogue:	AN0-AN3
ADC_5ANA	analogue:	AN0-AN4
ADC_6ANA	analogue:	AN0-AN5
ADC_7ANA	analogue:	AN0-AN6
ADC_8ANA	analogue:	AN0-AN7
ADC_9ANA	analogue:	AN0-AN8
ADC_10ANA	analogue:	AN0-AN9
ADC_11ANA	analogue:	AN0-AN10
ADC_12ANA	analogue:	AN0-AN11

Example

```
'
' Open the ADC:
'               Fosc/32
'               Right justified for 10-bit operation
'               Tad value of 2
'               Vref+ at Vcc : Vref- at Gnd
'               Make AN0 an analogue input
OpenADC(ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_2_TAD, ADC_REF_VDD_VSS, ADC_1ANA)
```

ReadADC

Syntax

Variable = **ReadADC**(pChannel)

Include file

Amicus18_ADC.inc

Overview

This macro returns the Word (10 bit) result of the A/D conversion. Based on the configuration of the A/D converter (e.g., using the **OpenADC**() macro).

Operator

pChannel is an *optional* ADC channel to take the reading from. This *must* be one of the values used for the **SetChanADC** macro.

Example

```
Dim wResult as Word

wResult = ReadADC(ADC_CH0)
```


SetChanADC

Syntax
SetChanADC(pChannel)

Include file
Amicus18_ADC.inc

Overview
Selects the pin that will be used as input to the A/D Converter.

Operator
pChannel One of the following values (defined in Amicus18_ADC.inc):

ADC_CH0	Channel 0
ADC_CH1	Channel 1
ADC_CH2	Channel 2
ADC_CH3	Channel 3
ADC_CH4	Channel 4
ADC_CH5	Channel 5
ADC_CH6	Channel 6
ADC_CH7	Channel 7
ADC_CH8	Channel 8
ADC_CH9	Channel 9
ADC_CH10	Channel 10
ADC_CH11	Channel 11
ADC_CH12	Channel 12
ADC_CH13	Channel 13
ADC_CH14	Channel 14
ADC_CH15	Channel 15
ADC_CH_CTMU	Channel 13
ADC_CH_VDDCORE	Channel 14
ADC_CH_VBG	Channel 15

SelChanConvADC

Syntax
SelChanConvADC(pChannel)

Include file
Amicus18_ADC.inc

Overview
Selects the pin that will be used as input to the A/D converter. And starts an A/D conversion. The **BusyADC**() macro or A/D interrupt may be used to detect completion of the conversion.

Operator
pChannel One of the values used for the SetChanADC macro.

Example
```
SelChanConvADC(ADC_CH0)
```


ADC_IntEnable() Enables the ADC interrupt i.e. sets PEIE and ADIE bits.
ADC_IntDisable() Disables the ADC interrupt i.e. clears ADIE bit.

Example use of the A/D Converter Macros:

```
  Include "Amicus18.inc"        ' Configure the compiler to use the Amicus18 board
  Include "Amicus18_ADC.inc"    ' Load the Amicus18 ADC macros into the program

  Dim Result as Word
'
' Open the ADC:
'              Fosc / 32
'              Right justified for 10-bit operation
'              Tad value of 2
'              Vref+ at Vcc : Vref- at Gnd
'              Make AN0 an analogue input
'
  OpenADC(ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_2_TAD, ADC_REF_VDD_VSS, ADC_1ANA)
  DelayUs 2                     ' Delay for 2 microSeconds
  Result = ReadADC(ADC_CH0)     ' Read result of AN0
  CloseADC()                    ' Disable A/D converter
```


Timer macros Introduction

The timer peripherals are supported with the following macros. The macros are a mixture of compiler types and preprocessor types, and can be found in "Includes\Sources\Amicus18_Timers.inc"

- CloseTimerx Disable timer x.
- OpenTimerx Configure and enable timer x.
- ReadTimerx Read the value of timer x.
- WriteTimerx Write a value into timer x.
- SetTmrCCPSrc Configure the timer as a clock source to CCP module.

CloseTimer0

Syntax
CloseTimer0()

Include file
Amicus18_Timers.inc

Overview
This macro disables timer0 and it's interrupt.

CloseTimer1

Syntax
CloseTimer1()

Include file
Amicus18_Timers.inc

Overview
This macro disables timer1 and it's interrupt.

CloseTimer2

Syntax
CloseTimer2()

Include file
Amicus18_Timers.inc

Overview
This macro disables timer2 and it's interrupt.

CloseTimer3

Syntax
CloseTimer3()

Include file
Amicus18_Timers.inc

Overview
This macro disables timer3 and it's interrupt.

OpenTimer0

Syntax
OpenTimer0(pConfig)

Include file
Amicus18_Timers.inc

Overview
This macro configures timer0 according to the options specified and then enables it.

Operator
pConfig A bitmask that is created by performing either a bitwise AND operation ('&'), which is user configurable, with a value from each of the categories listed below. These values are defined in the file Amicus18_Timers.inc.

Enable Timer0 Interrupt:

TIMER_INT_ON	Interrupt enabled
TIMER_INT_OFF	Interrupt disabled

Timer Width:

T0_8BIT	8-bit mode
T0_16BIT	16-bit mode

Clock Source:

T0_SOURCE_EXT	External clock source (I/O pin)
T0_SOURCE_INT	Internal clock source (Tosc)

External Clock Trigger (for T0_SOURCE_EXT):

T0_EDGE_FALL	External clock on falling edge
T0_EDGE_RISE	External clock on rising edge

Prescale Value:

T0_PS_1_1	1:1 prescale
T0_PS_1_2	1:2 prescale
T0_PS_1_4	1:4 prescale
T0_PS_1_8	1:8 prescale
T0_PS_1_16	1:16 prescale
T0_PS_1_32	1:32 prescale
T0_PS_1_64	1:64 prescale
T0_PS_1_128	1:128 prescale
T0_PS_1_256	1:256 prescale

Example
```
OpenTimer0(TIMER_INT_OFF & T0_8BIT & T0_SOURCE_INT & T0_PS_1_32)
```


OpenTimer1

Syntax
OpenTimer1(pConfig)

Include file
Amicus18_Timers.inc

Overview
This macro configures timer1 according to the options specified and then enables it.

Operator
pConfig A bitmask that is created by performing either a bitwise AND operation ('&'), which is user configurable, with a value from each of the categories listed below. These values are defined in the file Amicus18_Timers.inc.

Enable Timer1 Interrupt:

TIMER_INT_ON	Interrupt enabled
TIMER_INT_OFF	Interrupt disabled

Timer Width:

T1_8BIT_RW	8-bit mode
T1_16BIT_RW	16-bit mode

Clock Source:

T1_SOURCE_EXT	External clock source (I/O pin)
T1_SOURCE_INT	Internal clock source (Tosc)

Prescaler:

T1_PS_1_1	1:1 prescale
T1_PS_1_2	1:2 prescale
T1_PS_1_4	1:4 prescale
T1_PS_1_8	1:8 prescale

Oscillator Use:

T1_OSC1EN_ON	Enable Timer1 oscillator
T1_OSC1EN_OFF	Disable Timer1 oscillator

Synchronise Clock Input:

T1_SYNC_EXT_ON	Sync external clock input
T1_SYNC_EXT_OFF	Don't sync external clock input

Example
```
OpenTimer1(TIMER_INT_ON & T1_8BIT_RW & T1_SOURCE_EXT & T1_PS_1_1)
```


OpenTimer2

Syntax
OpenTimer2(pConfig)

Include file
Amicus18_Timers.inc

Overview
This macro configures timer2 according to the options specified and then enables it.

Operator
pConfig A bitmask that is created by performing either a bitwise AND operation ('&'), which is user configurable, with a value from each of the categories listed below. These values are defined in the file Amicus18_Timers.inc.

Enable Timer2 Interrupt:

TIMER_INT_ON	Interrupt enabled
TIMER_INT_OFF	Interrupt disabled

Prescale Value:

T2_PS_1_1	1:1 prescale
T2_PS_1_4	1:4 prescale
T2_PS_1_16	1:16 prescale

Postscale Value:

T2_POST_1_1	1:1 postscale
T2_POST_1_2	1:2 postscale
T2_POST_1_3	1:3 postscale
T2_POST_1_4	1:4 postscale
T2_POST_1_5	1:5 postscale
T2_POST_1_6	1:6 postscale
T2_POST_1_7	1:7 postscale
T2_POST_1_8	1:8 postscale
T2_POST_1_9	1:9 postscale
T2_POST_1_10	1:10 postscale
T2_POST_1_11	1:11 postscale
T2_POST_1_12	1:12 postscale
T2_POST_1_13	1:13 postscale
T2_POST_1_14	1:14 postscale
T2_POST_1_15	1:15 postscale
T2_POST_1_16	1:16 postscale

Example

```
OpenTimer2(TIMER_INT_OFF & T2_PS_1_1 & T2_POST_1_8)
```


OpenTimer3

Syntax
OpenTimer3(pConfig)

Include file
Amicus18_Timers.inc

Overview
This macro configures timer3 according to the options specified and then enables it.

Operator
pConfig A bitmask that is created by performing either a bitwise AND operation ('&'), which is user configurable, with a value from each of the categories listed below. These values are defined in the file Amicus18_Timers.inc.

Enable Timer3 Interrupt:

TIMER_INT_ON	Interrupt enabled
TIMER_INT_OFF	Interrupt disabled

Timer Width:

T3_8BIT_RW	8-bit mode
T3_16BIT_RW	16-bit mode

Clock Source:

T3_SOURCE_EXT	External clock source (I/O pin)
T3_SOURCE_INT	Internal clock source (Tosc)

Prescale Value:

T3_PS_1_1	1:1 prescale
T3_PS_1_2	1:2 prescale
T3_PS_1_4	1:4 prescale
T3_PS_1_8	1:8 prescale

Synchronise Clock Input:

T3_SYNC_EXT_ON	Sync external clock input
T3_SYNC_EXT_OFF	Don't sync external clock input

Example
```
OpenTimer3(T3_8BIT_RW & T3_SOURCE_EXT & T3_PS_1_1 & T3_SYNC_EXT_OFF)
```

ReadTimer0

Syntax
Variable = **ReadTimer0**()

Include file
Amicus18_Timers.inc

Overview
This macro reads the value of the timer0 register pair.
Timer0: TMR0L,TMR0H

ReadTimer1

Syntax
Variable = **ReadTimer1**()

Include file
Amicus18_Timers.inc

Overview
This macro reads the value of the timer1 register pair.
Timer1: TMR1L,TMR1H

ReadTimer2

Syntax
```
Var = ReadTimer2()
```

Include file
Amicus18_Timers.inc

Overview
This macro reads the value of the timer2 register.
Timer2: TMR2

ReadTimer3

Syntax
Variable = **ReadTimer3**()

Include file
Amicus18_Timers.inc

Overview
This macro reads the value of the timer3 register pair.
Timer3: TMR3L,TMR3H

WriteTimer0

Syntax
WriteTimer0(pTimer)

Include file
Amicus18_Timers.inc

Overview
This macro writes a value to the timer0 register pair:
Timer0: TMR0L,TMR0H

Operator
pTimer The value that will be loaded into timer0.

Example
```
WriteTimer0(12340)
```

WriteTimer1

Syntax
WriteTimer1(pTimer)

Include file
Amicus18_Timers.inc

Overview
This macro writes a value to the timer1 register pair:
Timer1: TMR1L,TMR1H

Operator
pTimer The value that will be loaded into timer1.

Example
```
WriteTimer1(12340)
```

WriteTimer2

Syntax
WriteTimer2(pTimer)

Include file
Amicus18_Timers.inc

Overview
This macro writes a value to the timer1 register:
Timer2: TMR2

Operator
pTimer The value that will be loaded into timer2.

Example
```
WriteTimer2(100)
```


WriteTimer3

Syntax
WriteTimer3(pTimer)

Include file
Amicus18_Timers.inc

Overview
This macro writes a value to the timer1 register pair:
Timer3: TMR3L,TMR3H

Operator
pTimer The value that will be loaded into timer3.

Example
```
WriteTimer3(10000)
```

SetTmrCCPSrc

Syntax
SetTmrCCPSrc(pConfig)

Include file
Amicus18_Timers.inc

Overview
This macro configures a timer as a clock source for the CCP module.

Operator
pConfig A constant value from the list below. The values are defined in the file TimerDefs.inc.

T3_SOURCE_CCP	Timer3 source for both CCP's
T1_CCP1_T3_CCP2	Timer1 source for CCP1 and Timer3 source for CCP2
T1_SOURCE_CCP	Timer1 source for both CCP's

Example
```
SetTmrCCPSrc(T34_SOURCE_CCP12)
```


T3_OSC1EN_ON

Syntax
T3_OSC1EN_ON()

Include file
Amicus18_Timers.inc

Overview
This Macro enables the oscillator associated with Timer1 as source of external clock input for Timer3.

T3_OSC1EN_OFF

Syntax
T3_OSC1EN_OFF()

Include file
Amicus18_Timers.inc

Overview
This Macro disables the oscillator associated with Timer1 and selects the signal on pin T13CKI as the source of the external clock input for Timer3.

Example Use of the Timer0 Macro:

```
Include "Amicus18.inc"         ' Configure the compiler to use the Amicus18 board
Include "Amicus18_Timers.Inc"  ' Load the Amicus18 Timer Macros into the program

Dim Result As Word

' Configure Timer0
OpenTimer0(TIMER_INT_OFF & T0_SOURCE_INT & T0_PS_1_32 & T0_16BIT)

HRSOut "Press a Key\r"
While 1 = 1
  While Inkey = 16 : Wend         ' Wait for a Keypress on the keypad
  Result = ReadTimer0()           ' Read Timer0
  WriteTimer0(0)                  ' Reset Timer0
  HRSOut "Timer0 Value = ", Dec Result,13 ' Display the value of Timer0
  While InKey <> 16 : Wend        ' Wait for the key to released
  DelayMS 50
Wend
CloseTimer0()                     ' Close timer0
```


SPI macros Introduction

The following macros are provided for the SPI™ peripheral:

- CloseSPI — Disable the SSP module used for SPI™ communications.
- DataReadySPI — Determine if a new value is available from the SPI buffer.
- OpenSPI — Initialise the SSP module used for SPI communications.
- ReadSPI — Read a byte from the SPI bus.
- WriteSPI — Write a byte to the SPI bus.

CloseSPI

Syntax
CloseSPI()

Include file
Amicus18_SPI.inc

Overview
This Macro disables the SSP module. Pin I/O returns under the control of the appropriate TRIS and LAT registers.

DataReadySPI

Syntax
Variable = **DataReadySPI**()

Include file
Amicus18_SPI.inc

Overview
This Macro determines if there is a byte to be read from the SSPBUF register.

Return Values
0 if there is no data in the SSPBUF register
1 if there is data in the SSPBUF register

Example
```
While DataReadySPI() = 0 : Wend
```


OpenSPI

Syntax
OpenSPI(pSyncMode, pBusMode, pSmpPhase)

Include file
Amicus18_SPI.inc

Overview
This Macro sets up the SSP module for use with a SPIx bus device.

Operators
pSyncMode One of the following values, defined in Amicsu18_SPI.inc:

SPI_FOSC_4	SPI Master mode, clock = Fosc / 4, resulting in a 1MHz interface.
SPI_FOSC_16	SPI Master mode, clock = Fosc / 16, resulting in a 4MHz interface.
SPI_FOSC_64	SPI Master mode, clock = Fosc / 64, resulting in a 16MHz interface.
SPI_FOSC_TMR2	SPI Master mode, clock = TMR2 output / 2
SLV_SSON	SPI Slave mode, /SS pin control enabled
SLV_SSOFF	SPI Slave mode, /SS pin control disabled

pBusMode One of the following values, defined in SPIdefs.inc:

MODE_00	Setting for SPI bus Mode 0,0
MODE_01	Setting for SPI bus Mode 0,1
MODE_10	Setting for SPI bus Mode 1,0
MODE_11	Setting for SPI bus Mode 1,1

pSmpPhase One of the following values, defined in SPIdefs.inc:

SMPEND	Input data sample at end of data out
SMPMID	Input data sample at middle of data out

Example
```
OpenSPI(SPI_FOSC_16, MODE_00, SMPEND)
```


ReadSPI

Syntax
Variable = **ReadSPI**()

Include file
Amicus18_SPI.inc

Overview
This macro initiates a SPI bus cycle for the acquisition of a byte of data.

WriteSPI

Syntax
WriteSPI(pDataOut)

Include file
Amicus18_SPI.inc

Overview
This Macro writes a single data byte out.

Operator
pDataOut Value to be written to the SPI bus.

Example of SPI macros

```
Include "Amicus18.inc"          ' Configure the compiler to use the Amicus18 board
Include "Amicus18_SPI.inc"      ' Load the Amicus18 SPI macros into the program

Dim bTemp as Byte

OpenSPI(SPI_FOSC_16 , MODE_01 , SMPMID)
WriteSPI($55)
bTemp = ReadSPI()
DataReadySPI()
CloseSPI()
```

Hardware PWM macro Introduction

The PWM peripheral is supported with the following macros:

CloseAnalog1	Disable the CCP1 peripheral
CloseAnalog2	Disable the CCP2 peripheral
OpenAnalog1	Enable and configure the CCP1 peripheral
OpenAnalog2	Enable and configure the CCP2 peripheral
WriteAnalog1	Output an 8-bit or 10-bit Pulse Width Modulated waveform from CCP1
WriteAnalog2	Output an 8-bit or 10-bit Pulse Width Modulated waveform from CCP2

CloseAnalog1

Syntax
CloseAnalog1()

Include file
Amicus18_hpwm8.inc for 8-bit PWM
or
Amicus18_hpwm10.inc for 10-bit PWM

Overview
Disable the CCP1 peripheral and set its appropriate pin as an input.

CloseAnalog2

Syntax
CloseAnalog2()

Include file
Amicus18_hpwm8.inc for 8-bit PWM
or
Amicus18_hpwm10.inc for 10-bit PWM

Overview
Disable the CCP2 peripheral and set its appropriate pin as an input.

OpenAnalog1

Syntax
OpenAnalog1()

Include file
Amicus18_hpwm8.inc for 8-bit PWM
or
Amicus18_hpwm10.inc for 10-bit PWM

Overview
Enable and configure the CCP1 peripheral and set its appropriate pin as an output.

OpenAnalog2

Syntax
OpenAnalog2()

Include file
Amicus18_hpwm8.inc for 8-bit PWM
or
Amicus18_hpwm10.inc for 10-bit PWM

Overview
Enable and configure the CCP2 peripheral and set its appropriate pin as an output.

WriteAnalog1

Syntax
WriteAnalog1(pValue)

Include file
Amicus18_hpwm8.inc for 8-bit PWM
or
Amicus18_hpwm10.inc for 10-bit PWM

Note. The CCP1 peripheral will be operating at the highest frequency possible for 8-bit (0 to 255) or 10-bit (0 to 1023). With the default 64MHz oscillator this will be 62.5KHz for 10-bit and 250KHz for 8-bit.

Only one of the above include files may be used within a program an any one time.

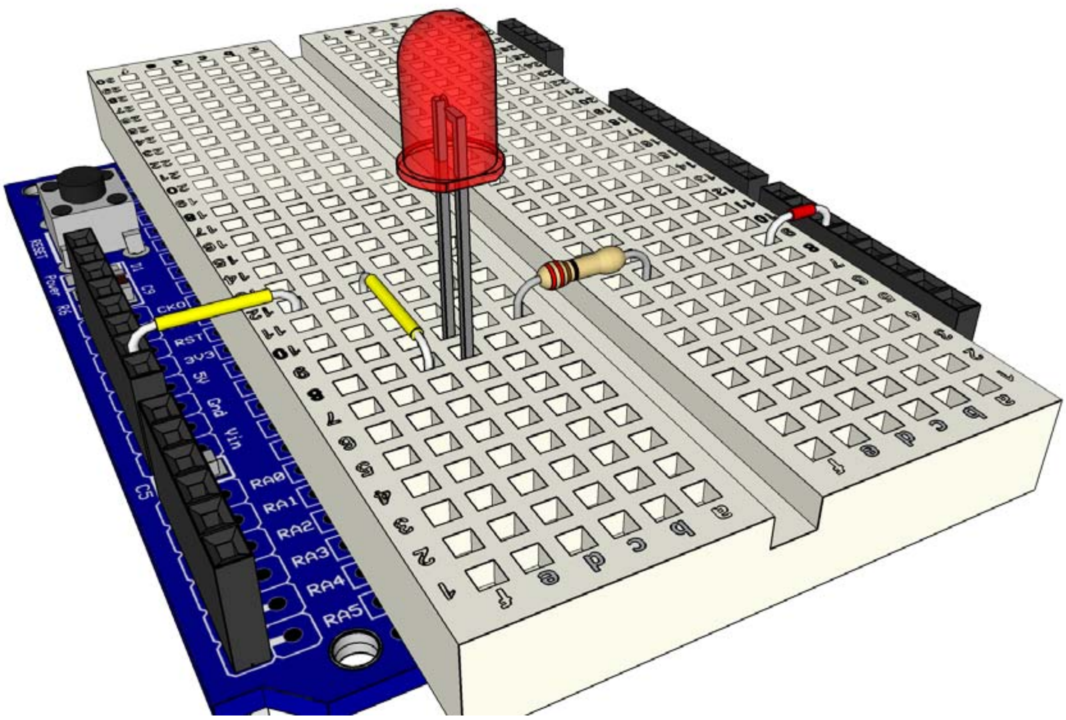
Overview
Output an 8-bit or 10-bit PWM waveform from the CCP1 peripheral's pin (RC2).

Operator
pValue a constant, variable, or expression that will alter the duty cycle of the PWM Waveform.

Example

```
' An LED attached to RC2 will increase illumination, then dim, repeatedly
' The voltage produced by the PWM signal is displayed on the serial terminal
'
Include "Amicus18.inc"          ' Configure the compiler to use the Amicus18 board
Include "Amicus18_Hpwm10.inc"  ' Load the Amicus18 10-bit PWM macros into program
Declare Float_Display_Type = fast   ' Faster, more accurate float display
Dim fVolts As Float                 ' Holds the Voltage calculation
Dim wTemp As Word                   ' Holds the duty cycle value for the PWM
'
' Quantasise the Voltage. i.e. Volts per-bit, based upon 3.3 Volts at 10-bits
'
Symbol Quanta = 3.3 / 1024
OpenAnalog1()                       ' Enable and configure the CCP1 peripheral
While 1 = 1                         ' Create an infinite loop
  '
  ' Increase LED illumation
  ' Cycle the full range of 10-bits. i.e. 0 to 1023
  For wTemp = 0 To 1023
    WriteAnalog1(wTemp)             ' PWM on CCP1 (Bit-2 of PortC)
    fVolts = wTemp * Quanta         ' Calculate the Voltage
    HRSOut Dec wTemp, " = ", Dec fVolts, " Volts", 13 ' Display Voltage
  Next
  '
  ' Decrease LED illumination
  ' Cycle the full range of 10-bits (reversed). i.e. 1023 to 0
  For wTemp = 1023 To 0 Step -1
    WriteAnalog1 (wTemp)            ' PWM on CCP1 (Bit-2 of PortC)
    fVolts = wTemp * Quanta         ' Calculate the Voltage
    HRSOut Dec wTemp, " = ", Dec fVolts, " Volts", 13 ' Display Voltage
  Next
Wend                                ' Do it forever
```

A suitable layout for the above program built on the Companion Shield using a solderless breadboard is shown below:



WriteAnalog2

Syntax
WriteAnalog2(pValue)

Include file
Amicus18_hpwm8.inc for 8-bit PWM
or
Amicus18_hpwm10.inc for 10-bit PWM

Note. The CCPx peripherals will be operating at the highest frequency possible for 8-bit (0 to 255) or 10-bit (0 to 1023). With the default 64MHz oscillator this will be 62.5KHz for 10-bit and 250KHz for 8-bit.

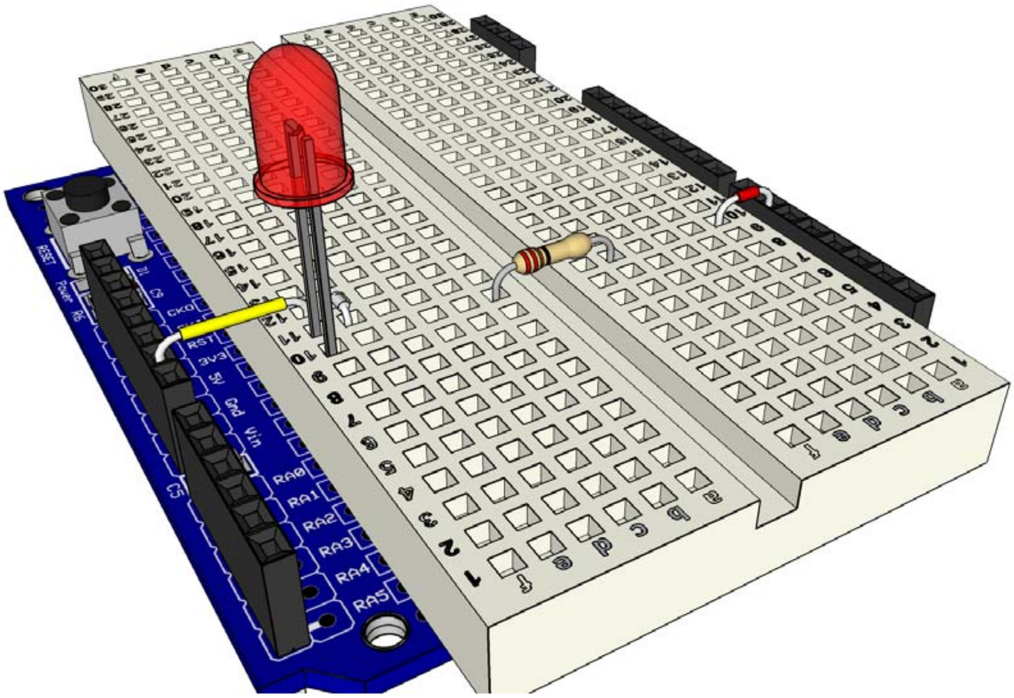
Only one of the above include files may be used within a program an any one time.

Overview
Output an 8-bit or 10-bit PWM waveform from the CCP2 peripheral's pin (RC1).

Operator
pValue a constant, variable, or expression that will alter the duty cycle of the PWM Waveform.

Example
```
' An LED attached to RC1 will increase illumination, then dim, repeatedly
' The voltage produced by the PWM signal is displayed on the serial terminal
'
Include "Amicus18.inc"         ' Configure the compiler to use the Amicus18 board
Include "Amicus18_Hpwm10.inc"  ' Load the Amicus18 10-bit PWM macros into the program
Declare Float_Display_Type = fast   ' Faster, more accurate float display

Dim fVolts As Float                 ' Holds the Voltage calculation
Dim wTemp As Word                   ' Holds the duty cycle value for the PWM
'
' Quantasise the Voltage. i.e. Volts per-bit, based upon 3.3 Volts at 10-bits
'
Symbol Quanta = 3.3 / 1023
OpenAnalog2()                       ' Enable and configure the CCP2 peripheral
While 1 = 1                         ' Create an infinite loop
  '
  ' Increase LED illumation
  ' Cycle the full range of 10-bits. i.e. 0 to 1023
  For wTemp = 0 To 1023
    WriteAnalog2(wTemp)             ' PWM on CCP2 (Bit-1 of PortC)
    fVolts = wTemp * Quanta         ' Calculate the Voltage
    HRSOut Dec wTemp, " = ", Dec fVolts, " Volts", 13 ' Display Voltage
  Next
  '
  ' Decrease LED illumination
  ' Cycle the full range of 10-bits (reversed). i.e. 1023 to 0
  For wTemp = 1023 To 0 Step -1
    WriteAnalog2 (wTemp)            ' PWM on CCP1 (Bit-1 of PortC)
    fVolts = wTemp * Quanta         ' Calculate the Voltage
    HRSOut Dec wTemp, " = ", Dec fVolts, " Volts", 13 ' Display Voltage
  Next
Wend                                ' Do it forever
```

A suitable layout for the previous program built on the Companion Shield using a solderless breadboard is shown below:



 Version 1.0 06-10-2009