

Athena
AthenaHS
Athena485
Perseus
Nemesis
NemesisHS

Microcontrollers

Athena, Perseus and Nemesis Manual

Version 4.8

Updates

Software and updated manuals can be obtained from the Kronos Robotics web site located at www.kronosrobotics.com.

Forums

A web-based discussions board is located at one of the Kronos Robotics sister sites. These forums cover everything from the Athena product line to motor controller basics and robotics. They are located at: www.kronosrobotics.com/forums.

Warranty

Kronos Robotics warrants its products against defects in material and workmanship for a period of 90 days. If you discover a defect, Kronos Robotics will, at its option, repair, replace, or refund the item's purchase price. Simply contact Kronos Robotics for an RMA. The customer is responsible for all return shipping expenses.

Disclaimer of Liability

Kronos Robotics cannot be held responsible for any incidental, or consequential damages resulting from the use of any Kronos Robotics products.

15-Day Money-Back Guarantee

It is very important to us here at Kronos Robotics that our customers are completely satisfied. If you are not happy with any product you purchase from Kronos Robotics it may be returned for a full refund or exchange within 15 days of the invoice date.

The returned items must be in unused condition and have original packaging. There will be a restocking fee of 30% only on items that do not meet this condition. Also note that this return policy does not apply to items that you have destroyed or damaged. For example if you hook up a microcontroller backwards you may not return it.

Shipping and Handling fees are non-refundable. The customer is responsible for all return shipping expenses.

Shipping Responsibility

Kronos Robotics ships all packages Priority Air Mail via the United States Postal Service with delivery confirmation. We have found this combination gives the fastest and most reliable delivery at a very reasonable cost.

Once a package has been delivered we are no longer responsible for the package. More specifically, if some one steals the package from your doorstep we will not be held responsible and no refund will be provided.

If your package has not been delivered and sufficient time has passed please email us and we will verify delivery. If delivery has been made you will be given the confirmation number and delivery details. Thereafter you must contact your local Post Office for further information.

TradeMarks

PIC is a registered trademark of Microchip Technology, Inc. Windows is a trademark of Microsoft Corporation. 1-wire is a trademark of Dallas Semiconductor.

Contacts

email: sales@kronosrobotics.com
phone: 703 779-9752
fax: 703 779-9753
web: www.kronosrobotics.com



Welcome

Welcome to the world of the Athena class microcontrollers. These chips are tiny computers (microcontrollers) complete with memory and IO ports. They have built-in hardware features such as timers, PWM generators and an interrupt driven hardware UART.

There are various carrier boards available to make hookup for your latest project a snap.

The Athena provides up to 15 IO lines depending on the chip. They have a built-in command set capable of running several thousand instructions per second. The AthenaHS runs 5 times faster than the Athena but sacrifices 2 IO ports for a 20Mhz Resonator.

The Perseus has up to 11 IO lines which include 8 Analog to Digital lines. The Perseus also supports extended chip to chip protocols that can be used in RS485 multi-drop applications.

The Nemesis is the most powerful of the Athena class microcontrollers. It has most of the features of the Athena and Perseus chips. It also supports inline assembly with the KRAssembler and has a new chipmaker feature.

The NemesisHS like the Nemesis is the most powerful of the Athena class microcontrollers. This chip runs at 20Mhz sacrificing 2 IO ports for a resonator.

Athena 4Mhz
Athena485 4Mhz
AthenaHS 20Mhz
Perseus 8Mhz
Nemesis 8Mhz
NemesisHS 20Mhz

Table of Contents

Athena Compiler Operations

Installing the PC Software	6
Connect Athena to PC	6
Writing your First Program	7
Athena Command Wizard	8
Setting Difficulty Mode	9

Hardware Overviews

Athena	11
Perseus	15
Nemesis	19

Language Overview

Language Model	24
Athena Expressions	24
Variables and Constants	25
Variable/Constant/Label names	25
Register Access	25
Math	26
Conversion	26
Macros	27

Command Syntax

Command Index	29
---------------------	----

Simulator

Starting The Simulator	92
Simulator Control	92
Runto Option	93
Simulator Form	94
Loading Stimuli Data	95
Stimuli Options	95
Simulator LCD	96
Variable Watch	96
Simulator output	97
Address Packet Simulation	98
Simulation Exceptions	98

Language Tutorial

Athena Program Model	101
Display Commands	101
Program Flow	103
Labels	103
Spaghetti Code	103
Gosub Command	104
Return Command	104
IO Ports	104
Output Mode	104
Input Mode	105
Variables	106
Dim statement	106
Constants	106
Math	107
Expressions	108
Full Expressions	108
vcn Expressions	108
For / Next Commands	109
If / Then / Else Commands	111
Delay Commands	112
pauseus	112
pause	112
longpause	112
Branch Command	113
end	114
Accessing Registers	114

Appendix

A: Athena Registers	115
B: Specifications	118
C: Chip Diagrams	119
D: Connecting the Athena to the PC	120
E: Build a RS232 Driver	121
F: BreadBoard Hookup Athena	123
F: BreadBoard Hookup Perseus	127
F: BreadBoard Hookup Nemesis	129
G: Carrier 1 Hookup	131
H: EDU/Carrier 3 Hookup	134
I: Perseus Carrier 1 Hookup	135
J: Athena/Nemesis Carrier 1u Hookup	137

Athena Compiler Operations

The Athena compiler can be used with the following chips:

- Athena
- AthenaHS
- Athena485
- Perseus
- Nemesis
- NemesisHS

Athena Operations

Installing the PC Software

Internet: Download the Athena Software from the Kronos Robotics web site. Locate the downloaded file program AthenaSetup.exe and double click it.

Note that the Athena Software is also located on the Program Editor Software CD. Just Insert the CD. Open the CD and double click on the AthenaSetup.exe file to start the install.

If you are installing over top of a previous version then you must remove it first. You will be prompted with three options. Select the Remove options and click next. Once removal is complete double click on the Athena Setup once again to start the installation process.

Note: If prompted to remove shared components just answer yes and continue.

Connect Athena to PC

Connect the RS232 driver to the PC by connecting the male end of a 9 pin serial cable to the 9 pin connector on the driver. Connect the other end of the cable to an available serial port on your PC. For actual Athena hookup check out Appendix D and E.

To test the connection start the software and from the Athena File Manager select **Change Com Port** from the **Settings** Menu.

Enter in the **Port** number corresponding with the serial port you plugged the cable into.

Select the **ChipTest** button.



Figure 1.1
Testing Athena Connection

If the Athena or Perseus chip is connected properly you will be presented with a successful test message.

If you do not get a successful test message check the following:

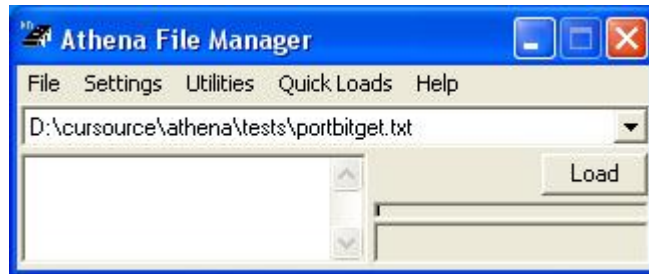
- Verify the serial port you plugged the cable into is the correct port you set in the Port field.
- Verify you have power on the Athena chip.
- Verify the correct transmit and receive leads on the driver are connected to the correct chip pin.

Note The **Loop Test** button is for doing detailed trouble shooting. Refer to the Athena 101 section of the web site for instructions.

Writing your First Program

Let's jump right in and create your first program. Start the Athena Editor Software.

Figure 1.2
Athena Software



On the Athena File manager select **New Athena File** from the File menu.

Figure 1.3
Athena File Manager



This will create an edit form with a default template ready for you to type in your program.

Figure 1.4a
Athena Edit Form

Type in the command **print "Hello World"** as shown then hit the program button.

If you are not using an Athena chip you need to add a chip directive. This tells the compiler which commands are available for the indicated chip.

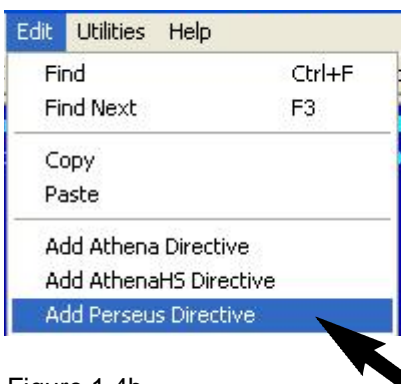


Figure 1.4b



Athena Operations

You will see the software detect your Athena and start to upload the program. This will take a few seconds and you can see the progress by watching the progress bar.

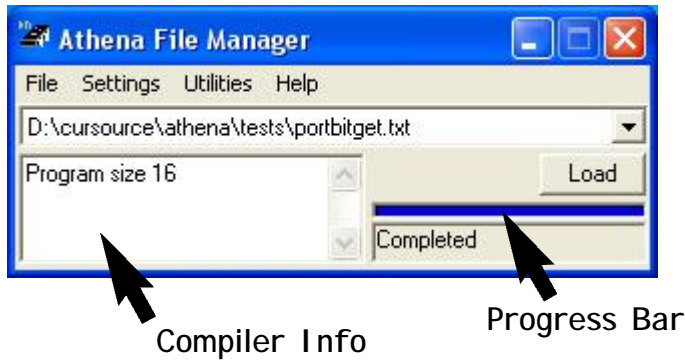


Figure 1.5
Athena Compiler

Once the program is uploaded the debug terminal will pop up automatically.

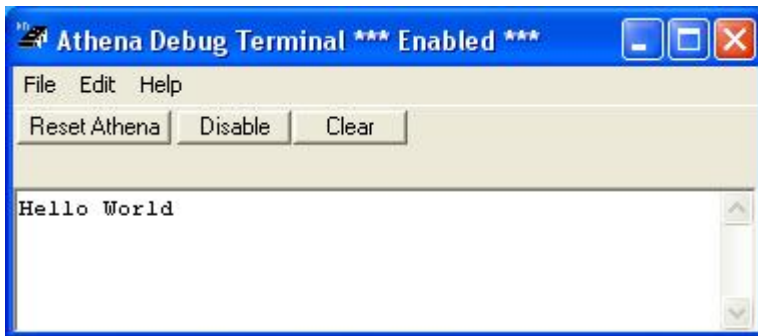


Figure 1.6
Debug Terminal

Any time the Compiler detects a debug or print command it will pop up the debug terminal for you.

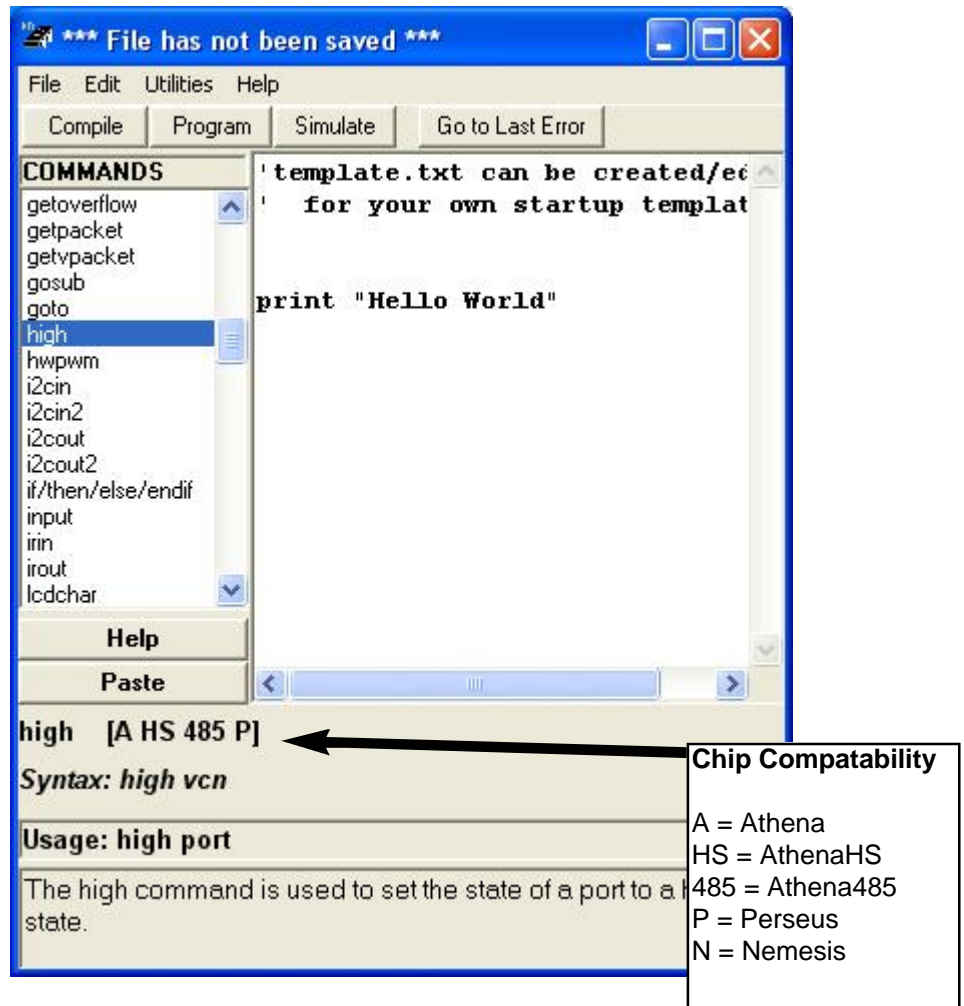
Athena Command Wizard

Each Edit form has a built in command wizard. This wizard will display a list of all the commands available along with syntax and usage. You can activate the wizard at any time by hitting F8.

The list of commands displayed will depend upon which difficulty level you have the Athena software set to.

By hitting the Paste button you can paste the usage field into your document.

Figure 1.7
Command Wizard



Setting Difficulty Mode

You can set the difficulty mode of Athena Software in the Settings menu on the File Manager.

Figure 1.8
Athena Difficulty Level



To make the new settings permanent make sure you select the save settings menu option.

Athena Operations

The three settings will determine what features get displayed on the various forms. It will also determine which commands get displayed in the command wizard.

It is recommended that you stick with Basic mode until you are familiar with the Athena software and the Athena Language.

Managing Files

The Athena software will keep track of the last 50 files you have edited. These files will be displayed in the file selection box in the order last accessed.

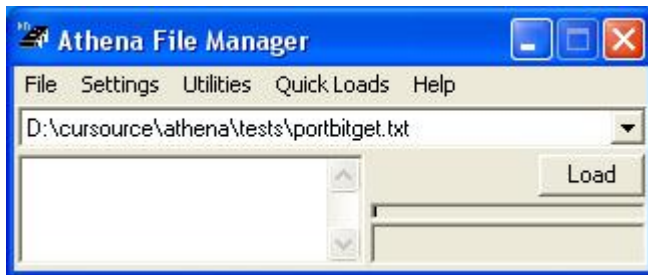


Figure 1.9
Athena Files

Just pull down the list and select the file you want to load. Once selected hit the Load button.

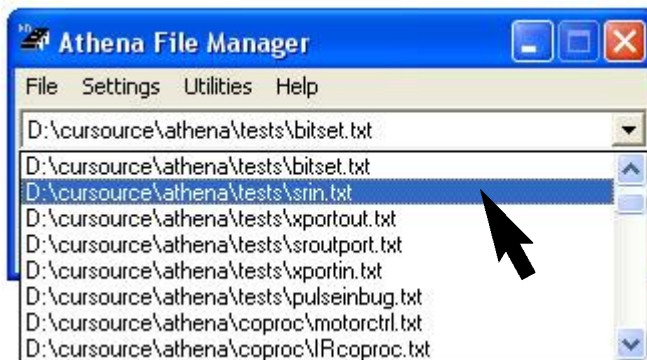


Figure 1.10
Athena Pick List

The Athena software also keeps track of the form size and location of each of these files.

Athena / AthenaHS / Athena485 Hardware Overview

IO Ports

Ports 0-7

Standard IOports. Can be accessed individually by port number or used with the portset and portget command to access all ports at once. Each port can handle a load of 25ma. These ports default to input when the Athena is started.

Ports 8-9

Standard IOports. Each port can handle a load of 25ma. These ports default to input when the Athena is started.

Port 10

Standard IO port however the output is open drain. This port must be pulled high with a 1k-10k resistor when used as output port. This port can sink 25ma. This port defaults to input when the Athena is started.

Ports 11-12

Standard IO port however they are used to program the Athena and to provide debug information to the terminal program. They can be placed in normal IOmode by issuing the following command.

```
RCSTA=0
```

This turns off the internal UART and places IO ports 11 and 12 into Standard IO mode. Each port can handle a load of 25ma. These ports default to UART mode when the Athena is started.

Ports 13-14

Standard IOports. Available on the Athena and Athena485 only. Used for 20mhz resonator on AthenaHS. Each port can handle a load of 25ma. These ports default to input when the Athena is started. Note that Port 14 is used when support for multi drop RS485 is enabled.

Internal Pullups

Ports 2-7 have a built-in pullup resistor. This resistor can be activated by using the pullupon and pulupoff commands.

Ports 11-12 also have this pullup option as well when placed in standard IOport mode.

ATN pin

In order for the Athena to operate properly this pin must be held high with a 10k resistor.

IRQ

Port 7 can be configured as a 8 bit counter that is driven by an internal IRQ. The register P7COUNTER contains the counts. See the p7irq command for more details

4 bit AtoD

Ports 0-1 can be used to access a 4 bit AtoD converter. See the miniad command for more details.

LCD Interface

Ports 0-5 can be used to access a parallel LCD. The LCD must be controlled by a Hitachi 44780 or equivalent controller. The lcdinit command places the ports in the proper mode of operation. See the lcdinit command for more details. **(Athena and AthenaHS only)**

IR Modulator output

Ports 6 can be used with to create a modulated output to drive an IR LED. See the irout command for more details.

PWM Signal Generator

Ports 6 can be used with to create a signal generator. See the hwpwm command for more details.

Built-in UART

The Athena has a built-in hardware UART. It is tied to port 11 (Rx) and port 12 (Tx). The UART is interrupt driven and handled by the internal Athena Engine. It has its own 80 byte buffer.

The UART is used to program the Athena and defaults to 9600 baud. See the debug,debugbaud and debugin commands for more details.

The UART on the **Athena485** can also be set up in 9-bit address mode so it can be used with a RS485 driver for multi drop support. You can enable port 14 as the transmit select pin for both full and half duplex operations on a multi drop bus.

TIMER

The Athena has an internal timer. This timer can use an internal or external reference. When an external reference is used connect a 32Khz crystal between ports 2 and 3.

The timer can be used as a timing device or as a clock. Several commands have been provided to access the timer and clock functions.

See the following commands for more information:

```
timerASexternalclock  
timerASinternalclock  
timercfg  
timerget  
timergetclock  
timeroff  
timeron  
timerset  
timersetclock
```

Counter

The Athena has hardware counter that can be setup for use on port 3. Note that when setup as counter you can not use the internal timer as a clock.

See the timerAScounter command for more details. Use the timerget command to retrieve the the 16bit counter data.

Sleep Mode

The Athena can be placed in a sleep state that can reduce power consumption to as little as .2ua. The IRQ's and Timers can be used to wake up the Athena from its sleep.

Perseus Hardware Overview

Perseus Hardware Overview

IO Ports

Ports 0-7

Standard IOports. Can be accessed individually by port number or used with the portset and portget command to access all ports at once. Each port can handle a load of 25ma. These ports default to input when the Perseus is started.

Ports 8-9

Standard IO port however they are used to program the Perseus and to provide debug information to the terminal program. They can be placed in normal IOmode by issuing the following command.

```
RCSTA=0
```

This turns off the internal UART and places IO ports 8 and 9 into Standard IOmode. Each port can handle a load of 25ma. These ports default to UART mode when the Perseus is started.

Port 10

Standard IOport. Port can handle a load of 25ma. These ports default to input when the Perseus is started. This port is also used when the UART is configured for RS485 support.

Internal Pullups

Ports 0,1,2,6,7 have a built-in pullup resistor. This resistor can be activated by using the pullupon and pulupoff commands.

ATN pin

This port on the Perseus is used to reset the chip. It is held high with an internal resistor.

IRQ

Port 2 can be configured as a 8 bit counter that is driven by a internal IRQ. The register P2COUNTER contains the counts. See the p2irq command for more details

Ports 0-2,6,7 can also be configured as a whole to increment the PORTCOUNTER register. See the portirq command for more details.

Analog to Digital Ports

Ports 0-5, 7,10 can be placed into analog mode. This allows them to be used with the atod command to convert analog voltages to digital values.

Built-in UART

The Perseus has a built-in hardware UART. It is tied to port 8 (Rx) and port 9 (Tx). The UART is interrupted driven and handled by the internal Perseus Engine. It has its own 80 byte buffer.

The UART is used to program the Perseus and defaults to 9600 baud. See the debug,debugbaud and debugin commands for more details.

The UART can also be placed into special 9 bit mode for use with the RS485 networks. Port 10 can be enabled for use as a transmit enable pin for support on a multi port RS485 bus.

The UART can be used to wake the Perseus from sleep.

System Clock

You can change the Perseus system clock on the fly with the systemclock command

The default system clock is 8 Mhz (Twice as fast as the Athena)

Watchdog Timer

You can set the watchdog timer to automatically wake up the Perseus when it is sleeping. The watchdog timer can also be used to reset the Perseus.

Cap Timer

Port 0 may be setup as a special timer that uses a capacitor to fire as it discharges. See the startcaptimer command.

Perseus Hardware Overview

TIMER

The Perseus has an internal timer. This timer can use an internal or external reference. When an external reference is used connect a 32Khz crystal between ports 2 and 3. Note that you will also need to place a 47pf capacitor between each of these ports and Vss.

The timer can be used as a timing device or as a clock. Several commands have been provided to access the timer and clock functions.

See the following commands for more information:

```
timerASexternalclock  
timerASinternalclock  
timercfg  
timerget  
timergetclock  
timeroff  
timeron  
timerset  
timersetclock
```

Counter

The Perseus has a hardware counter that can be setup for use on port 3. Note that when setup as counter you can not use the internal timer as a clock.

See the timerAScounter command for more details. Use the timerget command to retrieve the 16bit counter data.

Sleep Mode

The Perseus can be placed in a sleep state that can reduce power consumption to as little as .2ua. The IRQ's and Timers can be used to wake up the Perseus from its sleep.

Nemesis Hardware Overview

Nemesis Hardware Overview

IO Ports

Ports 0-7

Standard IOports. Can be accessed individually by port number or used with the portset and portget command to access all ports at once. Each port can handle a load of 25ma. These ports default to input when the Athena is started.

Note that Port 4 defaults to the Debug transmit and can not be used as an IOport unless the built-in UART is disabled.

Ports 8-11

Standard IOports. Each port can handle a load of 25ma. These ports default to input when the Athena is started.

Ports 4 and 12

Standard IO port however they are used to program the Nemesis and to provide debug information to the terminal program. They can be placed in normal IOmode by issuing the following command.

```
RCSTA=0
```

This turns off the internal UART and places IO ports 4 and 12 into Standard IO mode. Each port can handle a load of 25ma. These ports default to UART mode when the Nemesis is started.

Ports 13-14

Standard IOports. Available on the Nemesis only. Used for 20mhz resonator on NemesisHS. Each port can handle a load of 25ma. These ports default to input when the Nemesis is started. Note that Port 14 is used when support for multi drop RS485 is enabled.

Internal Pullups

Ports 2-7 , 11, 12 have a built-in pullup resistor. This resistor can be activated by using the pullupon and pulupoff commands.

The pullups on ports 4 and 12 will not function unless the UART is disabled.

ATN pin

In order for the Nemesis to operate properly this pin must be held high with a 10k resistor.

IRQ

Port 7 can be configured as a 8 bit counter that is driven by an internal IRQ. The register P7COUNTER contains the counts. See the p7irq command for more details

10 bit AtoD

Ports 0-3, 8-10 can be used to access a 10 bit AtoD converter. See the initatod and atod commands for more details.

LCD Interface

IR Modulator output

Ports 6 can be used with to create a modulated output to drive an IR LED. See the irout command for more details.

PWM Signal Generator

Ports 6 can be used with to create a signal generator. See the hwpwm command for more details.

Built-in UART

The Nemesis has a built-in hardware UART. It is tied to port 4 (Tx) and port 12 (Rx). The UART is interrupt driven and handled by the internal Nemesis Engine. It has its own 80 byte buffer.

The UART is used to program the Nemesis and defaults to 9600 baud. See the debug,debugbaud and debugin commands for more details.

The UART on the **Nemesis** can also be set up in 9-bit address mode so it can be use with a RS485 driver for multi drop support. You can enable port 14 as the transmit select pin for both full and half duplex operations on a multi drop bus.

Nemesis Hardware Overview

TIMER

The Nemesis has an internal timer. This timer can use an internal or external reference. When an external reference is used connect a 32Khz crystal between ports 2 and 3.

The timer can be used as a timing device or as a clock. Several commands have been provided to access the timer and clock functions.

See the following commands for more information:

```
timerASexternalclock  
timerASinternalclock  
timercfg  
timerget  
timergetclock  
timeroff  
timeron  
timerset  
timersetclock
```

Counter

The Nemesis has hardware counter that can be setup for use on port 3. Note that when setup as counter you can not use the internal timer as a clock.

See the timerAScounter command for more details. Use the timerget command to retrieve the the 16bit counter data.

Sleep Mode

The Nemesis can be placed in a sleep state that can reduce power consumption to as little as .2ua. The IRQ's and Timers can be used to wake up the Nemesis from its sleep.

System Clock

You can change the Perseus system clock on the fly with the systemclock command.

The default system clock is 8 Mhz (Twice as fast as the Athena)

Watchdog Timer

You can set the watchdog timer to automatically wake up the Perseus when it is sleeping. The watchdog timer can also be used to reset the Perseus.

Athena Language Overview

The Athena Compiler can be used on the following chips:

- Athena
- AthenaHS
- Athena485
- Perseus
- Nemesis
- NemesisHS

These chips all use the Athena language.

Language Model

The Athena language model is a single flat model much like that of other popular microcontrollers. All code is placed in a single program space and all variables are global.

This model deviates from the model used on the Dios however all commands on the Athena will also be made available on the Dios to make programs on the Athena more portable.

The Athena has 256 bytes of program space. This program space is nonvolatile which means it won't go away when the power is removed. Don't let this amount of program space fool you. The Athena Compiler/Tokenizer can pack quite a program into that amount of space.

The Athena has 64 8-bit variables and 80 bytes of UART buffer.



The Nemesis and NemesisHS has 2048 bytes of program memory. All other Athena class chips have 256 bytes.

The Nemesis will actually compile and import certain commands into the engine as needed. While this model is not as efficient as the Athena mode you will get 4-8 times the program size on the Nemesis.

Athena Expressions

The Athena supports a couple of different expression types.

True Expressions

A few commands like print,serout,lcdwrite,debug,if/then/else and variable assignments support true expressions. These expressions can contain numbers, variables, registers, constants and mathematical operations.

True Expression Examples

```
A=b+5
print A+7
if A+2 > 4 then
```

VCN Expressions

VCN expressions can contain a single number, variable or constant. They cannot contain math expressions or registers. Most of the Athena commands use this type of expression.

VCN Expression Examples

```
A=25
A=C
if A = B then
output 5
```

Special note

The if/then/else and variable assignment commands have internal VCN counter parts to reduce memory requirements. The compiler will automatically convert to these high efficiency commands when possible. This is part of the KRCompression II Technology.

Variables and Constants

Variables

For simplicity and speed there is only one variable type in the Athena Language. Its a byte variable and can contain a value of 0-255.

To create a variable use:

```
dim xyz
```

You may create more than one variable with a single dim statement:

```
dim xyz,abc
```

Constants

Constants are fixed numbers that are referenced by a useable name. For example, **const CLK 5** would allow us to use the word CLK to represent the number 5. That way you can have several references to the CLK port. To change the port from 5 to 6 means only having to change 1 line of code as shown in **example 2.2**.

Once a constant has been defined the value cannot be changed. In other words you can't have the statement CLK = 25.

example 2.1

```
dim myvarb1  
myvarb1 = 100  
  
print myvarb1
```

example 2.2

```
const CLK 5  
  
output CLK  
high CLK  
pulseout CLK,100
```

Variable/Constant/Label names

Name length

There is no limit to the size of the names.

Valid Characters

The first digit in a name must be a letter or underscore. After the first digit is defined you may use numbers in the name.

How many

You may define (dim) up to 64 variables.

You may define up to 500 constants. These have no affect on Dios memory.

You may define up to 500 labels. These have no affect on Dios memory.

Register Access

There are several hardware and software registers available to allow you to change the way some of the commands operate. A good example is the PULSINTIMEOUT register. This can be used speed up the pulsein command.

The registers will be listed where appropriate in the particular command they are used. Others will be listed at the end of the command syntax section.

Math

The Athena language supports 8-bit integer math. If your math result is greater than 8-bits it will be truncated to 8-bits.

All integer math is unsigned.

The following operations are supported:

*Multiplication

/Division

+Addition

-Subtraction

&And

|Or

^Xor

You can use math anywhere true expressions are supported.

Example:

```
dim res
```

```
res = 10 + 5
```

Conversion

The compiler supports three types of number or constant conversion.

Binary Conversion

You can assign a binary number to a variable or as part of an expression by preceding the number with a %.

Example:

```
res = %100
```

res will contain 4. Note that the MSB is on the left side.

Hex Conversion

You can assign a hexadecimal number to a variable or as part of an expression by preceding the number with a \$.

Example:

```
res = $41
```

res will contain 65.

ASCII conversion

You can convert an ASCII character to a byte by enclosing the character in single quotes.

Example:

```
res = 'A'
```

res will contain 65

Macros

The Athena language supports the use of both simple and complex macros.

Simple Macros

Simple macros are the easiest to create. You start a **macro** with the macro command. You end a macro with the **endmacro** command. Each macro is given a name that will be used throughout your code.

Here is an example of a simple macro.

'Macro Example

```
macro hello  
  print "hello"  
endmacro
```

```
hello
```

Each time you use the hello macro the word hello will be printed. With macros you can create your own commands. You can define up to 100 macros and they don't use program memory unless they are used.

You can also define a macro on the same line as in the following example.

'Macro Example

```
macro hello print "hello" : endmacro
```

```
hello
```

You can supply parameters to your macro like you would a command by using a [exp0] substitution field. Note that you can up to 20 parameters do they are named [exp0] through [exp19]. To use them just place them in your code where you want the parameter to be substituted.

'Macro Example

```
macro hello  
  print "hello [exp0]"  
endmacro
```

```
hello mike
```

To use the parameter just pass it as a argument with the macro name.

There is no processing of the text or paramaters. The text is simply substituted be for the compiler compiles that piece of code.

When using labels within macros you should always end the label name with a [#]. This allows the macro compiler a way of creating unique instances of the label.

'Macro Example

```
macro waitforhigh
loop[#]:
  onportgoto [exp0],loop[#],done[#]
done[#]:
endmacro
```

```
'Wait for port to go high
waitforhigh 4
```

Complex Macros

Complex macros are 2 part macros. With a two part macro you can start the macro include normal code then finish the macro. You split a macro into 2 parts with a **splitmacro** command. You must supply the 2nd half of the macro a name as well.

Here is an example of a complex macro.

'Macro Example

```
macro while
loop[#]:
  if [exp0] then
    splitmacro wend
    goto loop[#]
  endif
endmacro
```

```
'-----
dim x
x = 0
while x < 5
```

```
  print x
  x = x + 1
wend
```

This complex macro creates a new command while/wend.

apinit	31	onportgosub	53
apoffline	31	onportgoto	54
aponline	31	output	53
aptx	32	owreset	54
arrayget	35	owrx	54
arrayset	35	owtx	55
atod	34	p2irq	55
atodinit	33	p7irq	55
bintodec	36	pause	56
bintohex	36	pauseus	57
bitreset	37	portbitget	57
bitset	37	portbitset	57
bitsetvalue	37	portget	58
branch	38	portset	58
clearall	38	pot	59
configio	39	print	59
const	39	pullupoff	61
cpslave	39	pullupon (Athena & Nemesis)	61
cpslaveinit	40	pullupon (Perseus)	61
cptxmt	40	pulsein	60
cptxoff	40	pulseout	62
data	41	pulseoutms	62
debug	41	random	62
debugbaud	41	rccount	63
debugin	42	return	63
dim	42	serin	63
dimblock	42	serout	64
eeread	42	servo	64
eewrite	43	setbaud	65
end	43	setio	65
for/next/step	43	shiftin	66
getpacket	44	shiftout	66
getvpacket	44	signal	67
gosub	45	sleep	67
goto	45	srin	68
high	46	srinport	68
hwpwm	46	sROUT	68
i2cin	46	sROUTport	69
i2cin2	47	startasm	70
i2cout	47	startasmcommand	85
i2cout2	47	startcaptimer	69
if / then / else	48	systemclock	86
inp	48	timerAScounter	86
input	48	timerASexternalclock	86
irin	49	timerASinternalclock	87
irout	49	timercfg	87
KRAssembler Commands	72	timerget	87
lcdchar	50	timergetclock	87
lcdcontrol	50	timeroff	88
lcdinit	50	timeron	88
lcdwrite	51	timerset	88
longpause	51	timersetclock	88
lookdown	51	toggle	89
lookup	52	TW523read	89
low	52	TW523write	90
miniad	52	waitport	90
nop	53	watchdog	91

Command Syntax Overview

Each command name will be followed by a syntax example and a syntax parameter type. Commands may be upper, lower or mixed case.

If the command syntax example is omitted then the command has no parameters. Some commands can take repeating arguments. These will be indicated by the

For example:

output variable

output varb

output variable

This syntax example is used as a command reference and each parameter will be explained in the command description.

output varb

This syntax example shows the parameter type. It indicates the type for each parameter. Each type will be explained in detail below.

- **exp**: Any variable type, literal mathematical expression. Bit and Byte extensions are supported. Registers and array elements are also supported.
- **varb**: Variable accepted only. No arrays, bit or byte extensions.
- **vcn**: Can contain number, variable or constants
- **label**: Valid location label.
- **number**: Can contain only a number or constant. The value must be 0-255.
- **operator**: Valid operator as in > < >= <= <> != ==

_ Continuation Operator

This character can be placed at the end of a line. This tells the compiler that the current line continues to the next line. This is useful when doing tables as well as lookup and lookdown commands. The continuation operator can not be contained in quotes.

It is recommended that you try out the examples in this section and experiment. The examples are also located in the directory called `command_examples` in the directory you installed the Athena software.

Chip Indicators

Not all commands are compatible with all chips. For instance the `atod` command is only available on the Perseus and Nemesis chips. Each command will have a list of icons indicating the chips it supports.

A Athena

N Nemesis

HS AthenaHS

NH NemesisHS

485 Athena485

P Perseus

apinit

NH N 485 P

apinit address,mode
apinit vcn,keyword

Description

This command places the internal UART into 9 bit address mode. In this mode the UART will ignore all incoming data unless the 9th address bit is set. Once set the UART will check to see if the remaining 8 bits match address. If they match the UART will remain in normal mode and recognize all data until another address bit is sent again. If an address is relieved that does not match the UART's address the UART will go off line until another address is sent.

- **address** - The address to assign this device. 0-255
- **mode** - Valid modes:

RS485FULL : Full duplex mode with RS485 driver chips. IOport 14 is used as the transmit enable control.

RS485HALF : Full duplex mode with RS485 driver chips. IOport 14 is used to switch transmit and recive modes.

DIRECT : This is used when connecting master UART to slave UARTS in 9 bit address mode.

Note that once the apinit command has been issued all remaining commands are used as normal. The UART just wont fire unless the chip has been addressed.

Perseus
 'address packet example

```
dim dat
apinit 21
```

```
loop:
debugin loop,dat
debug dat
goto loop
```

apoffline

NH N 485 P

apoffline
apoffline

Description

Forces a device that has been initilized with the apinit command offline.

aponline

NH N 485 P

aponline
aponline

Description

Forces a device that has been initilized with the apinit command online.

aptx

NH N 485 P

```
aptx address[,data1,data2,.....]  
aptx vcn[vcn,vcn,...]
```

Description

This command is used to transmit a 9 bit address onto the bus. You may follow the address with multiple normal bytes of data.

- **address** - The address of the device you wish to transmit to
- **dataN** - You can transmit as many bytes as you wish after the device has been addressed. These bytes will be transmitted as normal 8 bit bytes.

Note you can also use this command to force all remote devices off line by sending to an address that does not exist.

```
Athena485  
'aptx example  
  
dim dat  
  
apinit 7,15  
  
aptx 21,3,3  
debugin -,dat  
print "[Dat=",dat,"]"
```

atodinit**NH** **N** **P**

atodinit port,port,....
 atodinit number,number,.....

Description

Sets up the analog to digital routines. This command must be called prior to using the atod command. You pass the the ports you wish to use for atod.

P The Perseus has 8 valid ports that can be used for atod.
 0,1,2,3,4,5,7,10.

N The Nemesis has 7 valid ports that can be used for atod.
 0,1,2,3,8,9,10

NH The NemesisHS has 7 valid ports that can be used for atod.
 0,1,2,3,8,9,10

- **port** - The port to be configured for analog input.

Perseus
 'atod example

```
dim res2
atodinit 2
```

loop:

```
atod 2,res2
```

```
print res2
```

```
goto loop
```

atod

atod port,result
 atod vcn,varb

Description

This command does the conversion analog conversion. The result is an 8 bit number

- **port** - The port to read. Can be 0,1,2,3,4,5,7,10 for Perseus and 0,1,2,3,8,9,10 for Nemesis
- **result** - 8 bit result will be placed into this variable.

The Perseus and Nemesis atod resolution is 10bits. However since these microcontrollers use 8 bit variables the accuracy is modified to 8 bit.

You can access the higher resolution results by reading the result from ADRESL and ADRESH registers. Use the ATODMODE register to change the format of these registers.

Perseus
 'atod example

```
dim res2
atodinit 2
```

loop:

```
atod 2,res2
```

```
print res2
```

```
goto loop
```

Registers

ATODMODE - This register is used to change the internal configuration of the analog to digital conversion.

Bit 0 - Voltage Reference (Only Valid on Perseus)

- 0 : Use Vdd (default)
- 1: Use port 1 as voltage Reference

Bit 4-6 - Conversion Frequency (Only Valid on Perseus)

- 000: clock/2 (default)
- 001: clock/8
- 010: clock/32
- 011: Special internal clock (500Khz)
- 100: clock/4
- 101: clock/16
- 110: clock/64

Bit 7: Result Format

0: Left Justification. Top 8 high order bits are placed in ADRESH. Note that ADRESH is also the register returned in the result variable. The 2 low order bits are returned as bits 7 and 8 in ADRESL. (default)

1: Right Justification. lower 8 bits are placed in ADRESL and top 2 bits are placed in ADRESH as bits 0 and 1. This format is normally used for returning a 16bit value.

arrayget

NH N 485 A HS P

arrayget startvarb,index,resultvarb
arrayget varb,vcn,varb

Description

All variables are stored in contiguous memory in the order in which they are declared with the dim statement. By passing the startvariable and using index you can retrieve values from variables in memory. This will allow you to cycle through variables like arrays.

- **startvarb** - The reference point into memory.
- **index** - Expression that will be added to the startvarb reference.
- **resultvarb** - The integer variable to store the retrieved value.

'arrayget and arrayset example

```

dim a,b,c,d,e,f,g
dim x,y

for x = 0 to 6
  arrayset a,x,99
next

c=50

for x = 0 to 6
  arrayget a,x,y
  print y
next

```

arrayset

NH N 485 A HS P

arrayset startvarb,index,value
arrayset varb,vcn,vcn

Description

All variables are stored in contiguous memory in the order in which they are declared with the dim statement. By passing the start variable and using index you can assign values to variables in memory. This will allow you to cycle through variables like arrays.

- **startvarb** - The reference point into memory.
- **index** - Expression that will be added to the startvarb reference.
- **value** - The value to assign to the indexed variable.

bintodec

NH N 485 A HS P

bintodec valueh,valuel,v100,v10,v1
bintodec vcn,vcn,varb,varb,varb

Description

Converts a variable or value into 3 individual variables. This is primarily for display systems like LCD's

- **valueh** - A value or variable that represents the top 8bits of the value to convert.
- **valuel** - The value or variable that represents the lower 8bits of the value to convert.
- **v100** - Where the hundreds indicator will be stored
- **v10** - Where the tens indicator will be stored
- **v1** - Where the ones indicator will be stored

Note: In order to display the returned values you will need to add the ASCII value 48 to each one as shown in the example.

```
'bintodec example
dim vhund,vtens,vones
dim x

x = 200

bintodec 0,x,vhund,vtens,vones

debug vhund
debug vtens
debug vones
```

bintoheX

NH N A HS

bintoheX value,hinib,lonib
bintoheX vcn,varb,varb

Description

Converts a 8-bit value to a two part hexadecimal

- **value** - The value or variable that will be converted.
- **hinib** - This is the high order nibble in hex format.
- **onib** - This is the low order nibble in hex format.

```
'bintoheX example
dim HInib,LOnib

bintoheX 255,HInib,LOnib

print "-----"
debug HInib
debug LOnib
debug 13,10
print "-----"
print hex 255
```

bitreset

NH N 485 A HS P

bitreset varb,bit
bitreset varb,vcn

Description

Will reset a bit in a variable.

- **varb** - The variable to modify.
- **bit** - The bit (0-7) to reset.

' bitreset and bitreset example

```
dim x

clear x
print x

bitset x,1
print x

bitreset x,1
bitreset x,7
print x
```

bitset

NH N 485 A HS P

bitset varb,bit
bitset varb,vcn

Description

Will set a bit in a variable.

- **varb** - The variable to modify.
- **bit** - The bit (0-7) to set.

bitsetvalue

NH N 485 A HS P

bitsetvalue value,varb,bit
bitsetvalue vcn,varb,vcn

Description

Will set or reset a bit in a variable based on bit 0 of value.

- **value** If bit 0 in value is 0 then bit will be reset. If bit 0 is 1 then bit will be set.
- **varb** - The variable to modify.
- **bit** - The bit (0-7) to set.

' bitsetvalue example

```
dim x

clear x
print x

bitsetvalue 1,x,1
print x

bitsetvalue 0,x,1
bitsetvalue 1,x,7
print x
```

branch

NH N 485 A HS P

branch offset,location,location,location....

branch varb,label,label,....

Description

Based on a given offset branch to a specified location.

- **offset** - A variable that specifies which location to jump to.
- **location** - Specifies the labels to jump to. If you specify a - for a given location the branch command will fall through when the offset matches that location.

Note that you can supply a < or - for location.

< will cause the command to jump to the beginning of the branch command when that location is indexed.

- will cause the command to fall through when that location is indexed.

'branch example

```
dim x
for x = 0 to 4
  branch x,do0,do1,do2,do3
  print "Fall Through"
```

```
cont:
  next
```

```
end
```

```
do0:
  print "Its 0"
  goto cont
```

```
do1:
  print "Its 1"
  goto cont
```

```
do2:
  print "Its 2"
  goto cont
```

```
do3:
  print "Its 3"
  goto cont
```

clearall

NH N 485 A HS P

clearall

clearall

Description

Clears all variables

'clearall example

```
dim a
dim b
```

```
a=1
b=2
print a," ",b
clearall
```

```
print a," ",b
```

configio

NH N 485 A HS P

configio ioport,ioport,...
configio number,number,...

Description

Special backdoor to Athena's IO ports. Provides a quick and fast way to configure the direction of all the IO ports. No matter how many ports you configure the command only takes 3 bytes.

- **ioport** - The IO port to set as output. All IO ports on the Athena not defined will be set as input.

N 485 A Ports 0-14 are valid for Athena.

NH HS Ports 0-12 are valid for AthenaHS.

P Ports 0-10 are valid for Perseus.

```
'configio example
'Sets ports 0-3 as output and high state.

configio 0,1,2,3
setio 0,1,2,3
```

const

NH N 485 A HS P

const name number
const name number

Description

This allows you to assign a name to a number. For instance say you are using the signal command to create sounds. You can create a constant to represent the port for your speaker and reference that name.

- **name** - The name you wish to give the constant.
- **number** - The number to the constant will represent.

```
'const example

const speaker 10
signal speaker,100,75
```

cpslave

NH N 485 A HS P

cpslave variableindex,faillabel
cpslave varb,label

Description

Checks to see if a valid CoProc Packet has been recieved. If not the program will exit to location indicated by faillabel.

- **variableindex** - This variable is the first variable of 9. It tells the cpslave routine where the variables are. These 9 variables must be contiguous. See example.
- **faillabel** - The location to jump to if a packet has not been recieved

```
'cpslave example

dim x

'These variables must be contiguous
dim CAddress,CPcmd,CPDATA1,CPDATA2
dim CPDATA3,CPDATA4,CPDATA5,CPDATA6
dim CPDATA7

const type 1
const id 3
const version 24

cpslaveinit type,id,version

loop:
cpslave CAddress,loop,cmd6,cmd7
print "Fall through cmd=",CPcmd
goto loop

cmd6:
for x = 97 to 122
cptxmt x
next
goto loop

cmd7:
cptxmt 10
cptxmt 11
cptxmt 12
cptxmt 13
goto loop
```

cpslaveinit

NH N 485 A HS P

cpslaveinit type,id,version
cpslaveinit vcn,vcn,vcn

Description

Sets up the address and version for use with cpslave.

- **type** - Used to set up the type portion of the Address byte.
- **id** - Used to setup the id portion of the address byte.
- **version** - This is the value returned when the master requests a version.

'cpslaveinit example

```
dim x
```

```
"These variables must be contiguous  
dim CAddress,CPcmd,CPDATA1,CPDATA2  
dim CPDATA3,CPDATA4,CPDATA5,CPDATA6  
dim CPDATA7
```

```
const type 1  
const id 3  
const version 24
```

```
cpslaveinit type,id,version
```

```
loop:
```

```
cpslave CAddress,loop,cmd6,cmd7  
print "Fall through cmd=",CPcmd  
goto loop
```

```
cmd6:
```

```
for x = 97 to 122  
  cptxmt x  
next  
goto loop
```

```
cmd7:
```

```
cptxmt 10  
cptxmt 11  
cptxmt 12  
cptxmt 13  
goto loop
```

cptxmt

NH N 485 A HS P

cptxmt value
cptxmt vcn

Description

Sends a byte to the UART Transmitter if the transmitter is off it will turn it on.

- **value** - The byte of data to send to the master.

'cptxmt and cptxoff example

```
cptxoff
```

```
pause 255  
pause 255  
pause 255  
pause 255  
pause 255
```

```
cptxmt 65
```

cptxoff

NH N 485 A HS P

cptxoff
cptxoff

Description

Shuts down the UART transmitter. Places it into a high impedance state. This allows multiple chips to connect to the master.

data

NH N 485 P

data value1,value2,...
data number,number,number,...

Description

This command reserves some of the program memory space as eeprom data space.

- **value1** - Default eeprom data

```
Athena485
'data / eeread / eewrite example

dim dat

data 10,20,30

eeread 1,dat
print "Before Write ",dat

eewrite 1,45

eeread 1,dat
print "After Write ", dat
```

debug

NH N 485 A HS P

debug value1,value2,...
debug exp,exp,...

Description

This command will print values to the debug terminal.

- **value1** - An expression that can contain any combination of numbers and variables. This can also be a quoted string.

Switches

- **dec** - Does a binary to decimal conversion before printing the
- **hex** - Does a binary to hexadecimal conversion before displaying the value. (Athena, AthenaHS and Nemesis only)

```
'debug example

dim dat

loop:

debugin loop,dat
debug dat
goto loop
```

debugbaud

NH N 485 A HS P

debugbaud baudrate
debugbaud number

Description

This command sets the baud rate for the built-in UART (Ports 9,10). The Default is 9600. Note that if the baud rate is changed from 9600 you can not communicate with the Athea from the Athena Software. It is recommended that you do all your debugging and testing at 19200 then change the baud rate once complete.

- **baudrate** - The baud rate setting

Valid rates:

- HBAUD9600-19200 (Athena / Athena485)
- HBAUD9600-1250000 (AthenaHS / NemesisHS) Nemesis rates:
- HBAUD300-115200 (Perseus 8Mhz)
- HBAUD300-115200 (Perseus 4Mhz)
- HBAUD300-57600 (Perseus 2Mhz)
- HBAUD300-19200 (Perseus 1Mhz)
- HBAUD300-9600 (Perseus 500k)
- HBAUD300-4800 (Perseus 250k)
- HBAUD300-2400 (Perseus 125k)
- HBAUD300-2400 (Perseus 31k)
- HBAUD2400-57600 (Nemesis 8Mhz)
- HBAUD1200-19200 (Nemesis 4Mhz)
- HBAUD1200-9600 (Nemesis 2Mhz)
- HBAUD1200-4800 (Nemesis 1Mhz)

```
'debugbaud example

debugbaud HBAUD19200

print "Hello"
```

Note that the Perseus also supports a value of AUTO. If uses the remote unit should send a 55h character (U).

debugin

NH N 485 A HS P

debugin nodata,variable,...
debugin label,varb,...

Description

Reads a byte in from the debug port and places it in a variable. If not data is available the program will jump to the location indicated by the label.

- **nodata** - The location to jump to if no data is present.

Note that you can supply a < or - for location.

< will cause the command to jump to the beginning of the command when the command jumps.

- will cause the command to fall through when the command jumps.

- **variable** - This is the variable to place received byte into.

'debugin example

```
dim dat
```

loop:

```
debugin loop,dat  
debug dat  
goto loop
```

dim

NH N 485 A HS P

dim variable,variable,variable.....
dim varb.....

Description

The Dim statement is used to define all variables. All variables are 8-bit unsigned.

You may create more than one variable with a single dim statement.

'dim example

```
dim myvarb  
of  
dim myvarb1,myvarb2
```

dimblock

NH N 485 A HS P

dimblock variable bytes
dim varb num

Description

The Dimblock statement is used to define a single variable and a block of memory. You supply the variable and the number of bytes to assign. This command is ment to create variables for use with the getarray and setarray commands.

Athena485

'data / eeread / eewrite example

```
dim dat
```

```
data 10,20,30
```

```
eeread 1,dat  
print "Before Write ",dat
```

```
eewrite 1,45
```

```
eeread 1,dat  
print "After Write ", dat
```

eeread

NH N 485 P

eeread address,result
eeread vcn,variable

Description

Reads a byte from the reserved eeprom space.

- **address** - The eeprom address to read from.
- **result** - The variable you wish to place the read data into.

eewrite

NH N 485 P

eewrite address,data
eewrite vcn,vcn

Description

Writes a byte to the reserved eeprom space.

- **address** - The eeprom address to write to.
- **data** - The data you wish to write.

Note: When writing data the chip can not receive any kind of interrupt. If it does the write will fail.

```
Athena485  
'data / eeread / eewrite example
```

```
dim dat
```

```
data 10,20,30
```

```
eeread 1,dat  
print "Before Write ",dat
```

```
eewrite 1,45
```

```
eeread 1,dat  
print "After Write ", dat
```

end

NH N 485 A HS P

end
end

Description

This command ends normal processing on the Athena. Note that some background processing will still take place.

```
'end example
```

```
print "Hello"
```

```
end
```

```
print "Will never make it here"
```

for/next/step

NH N 485 A HS P

for counter = start to end
for varb = vnc to vnc [step [-]number]

Description

The for command allows you to cycle through a variable with a start and stop point. You can count forward or backward any number of units using the step option. To count backward use the step -x option.

- **counter** - The variable used for the counter.
- **start** - The starting value of the counter.
- **end** - The ending value of the counter.
- **step number** - The amount to step forward or backwards.

```
'for/next example
```

```
dim x,y,z
```

```
for z = 10 to 200 step 10  
print z  
next
```

```
for z = 200 to 10 step -10  
print z  
next
```

getpacket

NH N 485 A HS P

getpacket faillabel,indexvarb,valuevarb1,valuevarb2...
getpacket label,varb,varb,varb...

Description

This command will collect data from remote device. Once all data elements are filled it will fall through. This command works with the hardware UART. This means it must use the debug ports.

- **faillabel** - This is the location to jump to if all the data has not been populated.

Note that you can supply a < or - for this label.

< will cause the command to jump to the beginning of the command when the command fails.

- will cause the command to fall through when the command fails.

- **indexvarb** - Is a variable that will contain the current index (byte number) of the populated data.

- **valuevarb1-valuevarbn** - The variables to be populated.

'getpacket example

```
dim cmdidx,data1,data2,data3  
clearall
```

loop:

```
'You can execute commands here  
getpacket loop,cmdidx,data1,data2,data3
```

```
'Will only fall through when all data is populated  
print "data1 ",data1  
print "data2 ",data2  
print "data3 ",data3
```

```
goto loop
```

getvpacket

NH N 485 P

getvpacket faillabel,indexvarb
getvpacket label,varb

Description

This command will collect data from remote device. The first byte transmitted will dictate the number of bytes to collect. Once all bytes are collected the command will fall through.

- **failable** - This is the location to jump to if all the data has not been populated.

Note that you can supply a < or - for this label.

< will cause the command to jump to the beginning of the command when the command fails.

- will cause the command to fall through when the command fails.

- **indexvarb** - Is a variable that will contain the current index (byte number) of the populated data. Note that all variables to be populated need to be consecutive (Define one after another).

'getvpacket example

```
Athena485  
'getvpacket Demo
```

```
dim cmdidx,data1,data2,data3  
clearall
```

loop:

```
'You can execute commands here  
getvpacket loop,cmdidx
```

```
'Will only fall through when all data is populated  
print "data1 ",data1  
print "data2 ",data2  
print "data3 ",data3
```

```
goto loop
```

gosub NH N 485 A HS P

gosub label
gosub label

Description

The gosub command allows us to create subroutines. A gosub is always paired with a return statement.

- **label** - The name of the label(location) to jump to.

```
'gosub example
dim age

age = 10 : gosub printage
age = 20 : gosub printage
age = 30 : gosub printage

end

printage:
print age
return
```

goto NH N 485 A HS P

gotob label
goto label

Description

The goto command allows to change program flow and unconditionally jump to any location.

- **label** - The name of the label(location) to jump to.

```
'goto example

goto point1

point2:
print "P2"
goto point3

point1:
print "P1"
goto point2

point3:
print "P3"

end
```

high

NH N 485 A HS P

high port
high vcn

Description

The high command is used to set the state of a port to a high state.

- **port** - The port number of the IO port to change. Can be variable/constant/number.

'high example

```
'Outputs 1000Hz signal on port 0
output 0
```

```
loop:
high 0
pauseus 109
low 0
pauseus 109
goto loop
```

hwpwm

NH N 485 A HS

hwpwm range,period,duty
hwpwm vcn,vcn,vcn

Description

This command will set up a signal generator on port 6 that runs independent of the Athena once set up.

- **range** Sets the range the PWM will perform. 0-2 is valid.
 - Range 0 Each unit is 1us (.5us Nemesis / .2us NemesisHS)
 - Range 1 Each unit is 4us (2us Nemesis / .8us NemesisHS)
 - Range 2 Each unit is 16us (8us Nemesis / 3.2us NemesisHS)
- **period** - This sets the frequency. 0-255 is valid.
- **duty** - This sets the duty cycle.

To get the frequency divide the total period into 1,000,000. For example range 1 with a period set to 100 would be 400us. This would calculate to 2,500Hz.

'hwpwm example

```
dim x

'Athena
'39.8Khz
hwpwm 0,24,12

'AthenaHS
'39.9Khz
'hwpwm 0,124,62

output 6
```

i2cin

NH N 485 A HS P

i2cin sda,scl,slaveaddr,dataaddress,resultvarb
i2cin vcn,vcn,number,vcn,vcn

Description

Reads a byte of data to a single byte I2c device.

- **sda** - IO port connected to the SDA port on the I2c Device
- **scl** - IO port connected to the SCL port on the I2c Device
- **slaveaddr** - The Slave address on the I2c device. Note that the r/w bit is set for you.
- **dataaddress** - The register address of the I2c device.
- **resultvarb** - The variable to hold the result retrieved value.

'i2cin example

```
dim x,z
clearall

for z = 0 to 255
x = 255 - z
i2cout 10,9,160,z,x
pause 3 'Need some delay
next

for z = 0 to 255
i2cin 10,9,160,z,x
print z," ",x
next
```

i2cin2**NH N 485 A HS P**

i2cin2 sda,scl,slaveaddr,dataaddressH,dataaddressL,resultvarb
i2cin2 vcn,vcn,number,vcn,vcn,varb

Description

Reads a byte of data to a dual byte I2c device.

- **sda** - IO port connected to the SDA port on the I2c Device
- **scl** - IO port connected to the SCL port on the I2c Device
- **slaveaddr** - The Slave address on the I2c device. Note that the r/w bit is set for you.
- **dataaddressH** - The high byte register address of the I2c device. (bank)
- **dataaddressL** - The low byte register address of the I2c device.
- **resultvarb** - The variable to hold the result retrieved value.

'i2cin2 example

```
const bank0 0
const bank1 1
const bank2 2

dim x,z,y
clearall

for z = 0 to 255
  x = 255 - z
  i2cout2 10,9,160,bank0,z,x
  pause 3
  i2cout2 10,9,160,bank1,z,z
  pause 3
  next

for z = 0 to 255
  i2cin2 10,9,160,bank0,z,x
  i2cin2 10,9,160,bank1,z,y
  print z, " ",x, " ",y
  next
```

i2cout**NH N 485 A HS P**

i2cout sda,scl,slaveaddr,dataaddress,datavalue
i2cout vcn,vcn,number,vcn,vcn

Description

Writes a byte of data to a single byte I2c device.

- **sda** - IO port connected to the SDA port on the I2c Device
- **scl** - IO port connected to the SCL port on the I2c Device
- **slaveaddr** - The Slave address on the I2c device. Note that the r/w bit is set for you.
- **dataaddress** - The register address of the I2c device. (bank)
- **datavalue** - The actual value to write to the I2c device.

'i2cout example

```
dim x,z
clearall

for z = 0 to 255
  x = 255 - z
  i2cout 10,9,160,z,x
  pause 3 'Need some delay
  next

for z = 0 to 255
  i2cin 10,9,160,z,x
  print z, " ",x
  next
```

i2cout2**NH N 485 A HS P**

i2cout2 sda,scl,slaveaddr,dataaddressH,dataaddressL,datavalue
i2cout2 vcn,vcn,number,vcn,vcn,vcn

Description

Writes a byte of data to a dual byte I2c device.

- **sda** - IO port connected to the SDA port on the I2c Device.
- **scl** - IO port connected to the SCL port on the I2c Device.
- **slaveaddr** - The Slave address on the I2c device. Note that the r/w bit is set for you.
- **dataaddressH** - The high byte register address of the I2c device.
- **dataaddressL** - The low byte register address of the I2c device.
- **datavalue** - The actual value to write to the I2c device.

'i2cout2 example

```
const bank0 0
const bank1 1
const bank2 2

dim x,z,y
clearall

for z = 0 to 255
  x = 255 - z
  i2cout2 10,9,160,bank0,z,x
  pause 3
  i2cout2 10,9,160,bank1,z,z
  pause 3
  next

for z = 0 to 255
  i2cin2 10,9,160,bank0,z,x
  i2cin2 10,9,160,bank1,z,y
  print z, " ",x, " ",y
  next
```

inp / input / if / then / else

inp

NH N 485 A HS P

inpPort or inp.Port
inpNumber or inp.Number

Description

This command is used to retrieve the state of a port. It can be used with the if or print command.

- **port** - IO port to test or retrieve.

' example

```
dim x
x = inp1
const myport 4
print inp.myport

if inp.1 = 1 then
  print "high"
else
  print "low"
endif
```

input

NH N 485 A HS P

input port
input vcn

Description

The input command sets indicated port as input. When an IO port is setup as input it is in a high impedance state. When connected to other devices you can read the actual state the port has been forced to.

- **port** - The port number of the IO lead to be set as input. Can be variable/constant/number.

'input example

```
input 0

loop:
  print inp0
  goto loop
```

if / then / else

NH N 485 A HS P

if exp operator exp then

Description

The if/then statements evaluate two expressions and executes all commands between the then and endif statement if found to be true. If false and the else option is supplied the commands between the else and endif statements will be executed.

Operators are used to indicate the comparison of expression 1 to expression 2.

Valid operators are:

- = or == equal
- <> or >< or != not equal
- > Greater than
- < Less than
- >= Greater than equal to
- <= Less than equal to

'if/then/else example

```
dim myvarb
myvarb= 22

if myvarb > 25 then
  print "more than 25"
endif

if myvarb > 25 then
  print "more than 25"
else
  print "Not more than 24"
endif
```

irin

NH N 485 A HS P

irin port,cmd,device
 irin vcn, varb, varb

Description

This command will read a IR module and decode the Sony IR protocol.

- **port** - The port number to read. The port the IR module is connected to.
- **cmd** - This is a variable that will contain the cmd code.
- **device** - This is a variable that will contain the device code. If this variable contains 0 the command has timed out.

Registers

- **IRMODE** - Sets the mode of operation
 - 0 - Standard Sony 7 bit command and 5 bit device. (default)
 - 1 - Extended Kronos Robotics Mode 8 bit command and 8 bit device.
- **PULSEINTIMEOUT** - When waiting for a pulse this parameter will increment. When it overflows a time out will occur. Valid values are 0-255. The higher the number the faster the timeout. 250 is a good starting value for most IRremote's.

```
'irin example
dim cmd,device

loop:
  irin 0,cmd,device
  if device = 0 then
    goto loop
  endif

  print cmd
  goto loop
```

irout

NH N 485 A HS

irout cmd,device
 irout vcn,vcn

Description

This command transmits a command and device code using the Sony IR protocol. The default modulator frequency is 40Khz. This command is hard wired to port 6.

- **cmd** - The command to transmit (7 bits).
- **device** - The Device code to transmit. (5 bits)

Registers

- **IRMODE** - Sets the mode of operation
 - 0 - Standard Sony 7 bit command and 5 bit device. (default)
 - 1 - Extended Kronos Robotics Mode 8 bit command and 8 bit device.
- **IROUTPERIOD** - sets the modulation frequency.

```
'irout example
dim x

again:
  for x = 1 to 50
    irout x,9
    pause 255
    pause 255
  next

  goto again
```

Icdchar

NH N A HS

Icdchar value
Icdchar vcn

Description

Send a character to the LCD.

- **value** - The character you wish to send to the LCD.

Icdcontrol

NH N A HS

Icdcontrol value
Icdcontrol vcn

Description

Send a control character to the LCD.

- **value** - The control character you wish to send to the LCD.

Icdinit

NH N A HS

Athena Syntax
Icdinit
Icdinit

Nemesis Syntax
Icdinit RS,E,D4,D5,D6,D7
Icdinit number,number,number,number,number,number

Description

Sets up the LCD for output. Only 2 line LCD's are supported. The LCD RW lead is wired to (Vss)

Note that the Athena and AthenaHS LCD routines are hard coded to

```
'D4 = Port 0
'D5 = Port 1
'D6 = Port 2
'D7 = Port 3
'RS = Port 4
'E = Port 5
```

With the Nemesis you may supply the ports you wish to use for each LCDport. If omitted they resort to the Athena defaults. If you do supply them you must supply all.

Notice in the Nemesis example that we turned off the UART. If you wish to use the UART with the LCD you will need to connect RS on the LCD to a port other than 4.

'LCD examples

```
dim x
Icdinit
```

'Note that the LCD routines are hard coded to

```
'D4 = Port 0
'D5 = Port 1
'D6 = Port 2
'D7 = Port 3
'RS = Port 4
'E = Port 5
```

```
Icdwrite "Kronos Robotics", control 195, "Welcome"
```

```
pause 250
pause 250
pause 250
pause 250
pause 250
pause 250
```

```
Icdcontrol 1 'Clear LCD
pause 2 'Need some time after cls
```

```
for x = 65 to 80
  Icdchar x
next
```

Nemesis

```
'Icd demo
RCSTA=0 'Turn off uart
dim x
```

'Use these with PCB3 You must hold pin 5 high

```
const LCDD4 0
const LCDD5 1
const LCDD6 2
const LCDD7 3
const LCDRS 4
const LCDE 5
```

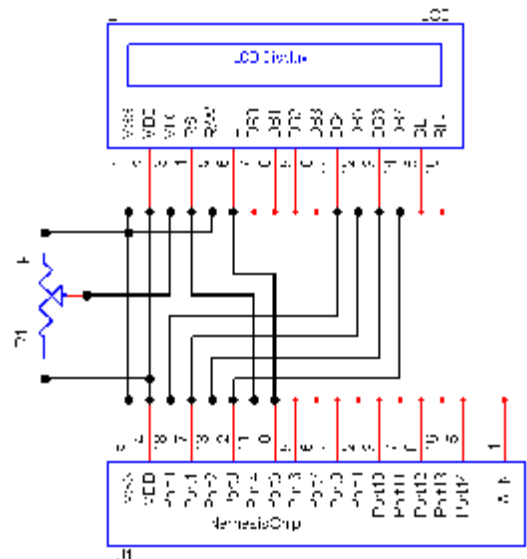
```
Icdinit LCDRS,LCDE,LCDD4,LCDD5,LCDD6,LCDD7
```

```
Icdwrite "Kronos Robotics", control 195, "Welcome"
```

```
longpause 250,6
```

```
Icdcontrol 1 'Clear LCD
pause 2 'Need some time after cls
```

```
for x = 65 to 80
  Icdchar x
next
```



lcdwrite

NH N A HS

lcdwrite value1,value1,value2
lcdwrite exp,exp,....

Description

Send a string of characters and numbers to a LCD.

Switches

- **dec** - Does a binary to decimal conversion before printing the value
- **hex** - Does a binary to hexadecimal conversion before displaying the value

'longpause example

```
dim x
for x = 1 to 50
  print x
  longpause 250,4 '1 second
next
```

longpause

NH N 485 A HS P

pause milliseconds,count
pause number,number

Description

This command will allow you to do very long pause. You can pause for 255 milliseconds for up to 255 times. That's over a minute

- **milliseconds** - This is a number that represents number of milliseconds to delay. 0-255
- **count** - The number of times to do pause the millisecond amount. 1-255

'lookdown example

```
dim x
LOOKDOWNMODE = 0
x=12
lookdown x,x,10,100,12

print x
```

lookdown

NH N 485 A HS P

lookdown resultvarb,searchvalue,item1,item2,item3,....
lookdown varb,vcn,number,number,number...

Description

Performs a search for search value. Will return the index number where found.

- **resultvarb** - This is where the result will be placed.
- **searchvalue** - The value to search for.
- **item1-item2** - The values to search for. These must be numbers 0-255.

Registers

- **LOOKDOWNMODE** - 0: The index returned in the resultvarb will be 0-n and the result variable will remain unchnaged if searchvalue is not found. (default)
- **LOOKDOWNMODE** - 1: The index returned in the resultvarb will be 1-n and the result variable will contain 0 if the searchvalue is not found.

lookup / low / miniad

lookup

NH N 485 A HS P

lookup resultvarb,index,item1,item2,item3,...
lookup varb,vcn,number,number,number...

Description

Performs a lookup for number translations

- **resultvarb** - This is where the result will be placed.
- **index** - The item index to translate
- **item1-item2** - The calculated conversion. These must be numbers 0-255.

'lookup example

```
dim dat
dat = 3
lookup dat,dat,0,10,20,30
print dat
```

low

NH N 485 A HS P

low port
low vcn

Description

The low command is used to set the state of a port to a low state.

- **port** - The port number of the IO port to change. Can be variable/constant/number.

'low example

```
'Outputs 1000Hz signal on port 0
output 0

loop:
high 0
pauseus 109
low 0
pauseus 109
goto loop
```

miniad

485 A HS

miniad port,resultvariable
miniad vcn,varb

Description

This command will perform a 4-bit analog to digital conversion on port 0 or 1. While 4 bits is not very high as AtoD ports go it is very fast and will give you 16 different levels.

- **port** - The IO port you wish to read. Must be port 0 or 1. If you select port 1 both port 0 and 1 are place in AtoD mode.
- **resultvariable** - This is the variable you wish to place the result into.

Note:

There are two ranges for the miniad command. By setting the high bit on the port parameter you can select the high range. See examples

Low Range 0 - 3.125v (default)

High Range 1.25 - 3.57v

Note: When using miniad on port 0, port 8 is unusable. When using port 1, port 9 is unusable.

'miniad example low range

```
dim level1,level2

loop:
miniad 0,level1
miniad 1,level2
print level1," ",level2
goto loop
```

'miniad example high range

```
dim level1,level2

loop:
miniad 128,level1
miniad 129,level2
print level1," ",level2
goto loop
```

nop

NH N 485 A HS P

nop
nop

Description

The nop command does nothing but spends time. It eats about 50us of time.

'nop example

output 0

```
loop:
toggle 0
nop *Small delay
goto loop
```

output

NH N 485 A HS P

output port
output vcn

Description

The output command sets indicated port as output. When an IO port is setup as output it can be set to high or low with the high and low commands.

- **port** - The port number of the IO lead to be set as input. Can be variable/constant/number.

'output example

*Outputs 1000Hz signal on port 0
output 0

```
loop:
high 0
pauseus 109
low 0
pauseus 109
goto loop
```

onportgosub

NH N 485 A HS P

onportgosub port,lowlabel,highlabel
onportgosub vcn,label,label

Description

Will perform a gosub to a location based on the stat of an IO port. Once return is reached the program will resume just after the portgosub.

- **port** - The port number to test for high or low state.
- **lowlabel** - The location to jump to if state is low.
- **highlabel** - The location to jump to if state is high.

'onportgosub example

input 0

```
loop:
onportgosub 0,lowlabel,highlabel
goto loop
```

lowlabel:
print "Low"
return

highlabel:
print "High"
return

onportgoto

NH N 485 A HS P

onportgoto port,lowlabel,highlabel
onportgoto vcn,label,label

Description

Will branch (jump) to a location based on the state of an IO port.

- **port** - The port number to test for high or low state.
- **lowlabel** - The location to jump to if state is low. You can force the command to fall through to the next command when the port is low by substituting a - for the lowlabel.
- **highlabel** - The location to jump to if state is high. You can force the command to fall through to the next command when the port is high by substituting a - for the highlabel

Note that you can supply a < or - for both highlabel and lowlabel.

< will cause the command to jump to the beginning of the command when the command fails.

- will cause the command to fall through when the command fails.

'onportgoto example

```
input 0
loop:
  onportgoto 0,lowlabel,highlabel

lowlabel:
  print "Low"
  goto loop

highlabel:
  print "High"
  goto loop
```

Note: When using the 1Wire commands on the Perseus it must be at 8Mhz clock speed.

Perseus
'1wire check for device

```
dim stat
owreset 5,stat
if stat = 1 then
  print "No Device"
else
  print "Device on bus"
endif
```

owreset

NH N 485 P

owreset port,status
owreset vcn,varb

Description

This command will reset the 1wire device.

- **port** - The port to use for communicating with the 1wire device. Note that this port must be held high with a 4.7k resistor.
- **status** - Once the reset is complete the status variable will hold the 0 if a device is detected and a 1 if not.

owrx

NH N 485 P

owrx port,result,result,....
owrx vcn,varb,varb,.....

Description

This command will read 1 or bytes from a 1 wire device.

port - The port to use for communicating with the 1wire device. Note that this port must be held high with a 4.7k resistor.

varb - For each varb passed the command will attempt to retrieve 1 byte of information from the device.

Important

The 1-wire commands are implemented in software. The use of IRQ's can cause disruption to there use.

Currently the following commands can generate an interrupt.

timerASinternalclock,
timerASexternalclock,
timerAScounter, p7irq, p2irq

Also the receipt of data on the debug port can cause and interrupt

Perseus
'1wire read rom example

```
dim stat
dim rom0,rom1,rom2,rom3,rom4,rom5,rom6,rom7
```

```
owreset 5,stat
owtx 5,$33
owrx 5,rom0,rom1,rom2,rom3,rom4,rom5,rom6,rom7
```

```
print rom0," ",rom1," ",rom2," ",rom3," ",rom4," ",rom5," ",rom6," ",rom7
```

owtx

NH N 485 P

owtx port,[reset,] byte,byte,.....
owtx vcn,[reset,] vcn,vcn,.....

Description

This command will send 1 or more bytes to a 1 wire device. Note you may add a reset before the data is transmitted.

port - The port to use for communicating with the 1wire device. Note that this port must be held high with a 4.7k resistor.

byte- For each byte passed the command will send 1 byte of information from the device. The first byte sent may be a reset sequence.



owtx reset option

Some devices require data to follow too quickly to use separate reset and owtx commands. For this reason the reset option was added to the owtx command.

This will cause a reset before the data is transmitted.

1-Wire command sequences start with a reset.

p2irq

P

p2irq config
p2irq number

Description

Enables or disables internal handling of Port 2 IRQ. Normally used to wake up the Perseus chip.

config - Broken down into bits.
bit 0 - 0=disables IRQ 1=Enables IRQ
bit 1 - 0=falling edge 1=rising edge

Registers

- **P2COUNTER** - This register will increment once each time this IRQ fires.

Perseus
'p2irq example

p2irq 3 'This will wake up Perseus with port 2

loop:

print "going to sleep"
sleep
print "wake up"

goto loop

p7irq

NH N 485 A HS

p7irq config
p7irq number

Description

Enables or disables internal handling of Port 7 IRQ. Normally used to wake up the Athena chip.

config - Broken down into bits.
bit 0 - 0=disables IRQ 1=Enables IRQ
bit 1 - 0=falling edge 1=rising edge

Registers

- **P7COUNTER** - This register will increment once each time this IRQ fires.

'p7irq example

configio 7
p7irq 3 'This will wake up Athena with port 7

loop:

print "going to sleep"
sleep
print "wake up"

goto loop

pause

NH 485 A HS P

pause milliseconds
pause vcn

Description

Will cause a delay in milliseconds.

- **milliseconds** - This is a number that represents number of milliseconds to delay. 0-255

'pause example

```
loop:  
  print "Hello World"  
  pause 250  
  pause 250  
  pause 250  
  pause 250  
  goto loop
```

pauseus

NH N 485 A HS P

pause microseconds
pauseus vcn

Description

Will cause a delay in 3 microsecond units.

- **microseconds** - The number of microseconds to delay. 0-255. Note that each unit represents 3 microseconds. A value of 3 would delay 9 microseconds. Also note that each command has a 50 microsecond overhead.

' pauseus example

```
output 0
```

```
loop:  
toggle 0  
pauseus 250  
goto loop
```

portbitget

NH N 485 A HS P

portbitget resultvarb,bit,port
portbitget varb,vcn,vcn

Description

This command sets the indicated bit in the result variable if the port is high. Resets the bit if the port is low.

- **resultvarb** - The variable to be modified.
- **bit** - The bit in the variable to modify.
- **port** - The IO port to check for high or low condition.

'portbitget example

```
dim x  
clear x
```

```
input 4
```

```
loop:  
portbitget x,7,4  
print x  
goto loop
```

portbitset

NH N 485 A HS P

portbitset value,bit,port
portbitset vcn,vcn,vcn

Description

Sets a port to high or low state depending on a particular bit in a given value.

- **value** - The value (vcn) to be analyzed.
- **bit** - The bit in the value to check.
- **port** - The IO port to set to high or low.

' portbitset example

```
dim x  
clear x
```

```
output 4
```

```
x = 2  
portbitset x,1,4
```

portget

NH N 485 A HS P

portget resultvarb
portget varb

Description

Sets the bits in the result variable based on the state of IO ports 0-7.

- **resultvarb** - The variable to be modified.

' portget example

```
dim dat
```

```
loop:  
  portget dat  
  print dat  
  goto loop
```

portset

NH N 485 A HS P

portset value
portset vcn

Description

Sets IO ports 0-7 based on the bits in the value passed.

- **value** - The 8 bits in value will be used to set the IO ports 0-7

'portset example

```
dim x  
output 0  
output 1  
output 2  
output 3  
output 4  
output 5  
output 6  
output 7
```

```
loop:  
  for x = 0 to 255  
    portset x  
  next  
  goto loop
```

' example

pot

NH N 485 A HS P

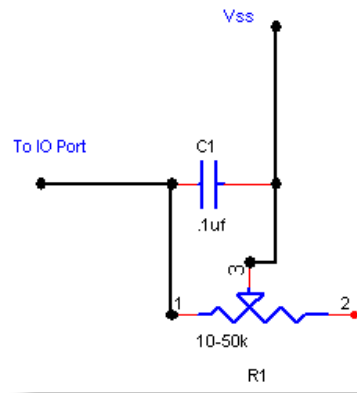
pot port,divisor,variable
pot vcn,vcn,varb

Description

This command will place the given port high for 5 milliseconds. Then sets it to input and wait till the port returns to a low stat. The command will return a value of 1-255 for a valid count and 0 if the main counter overflows.

- **port** - The port to test
- **divisor** - This is a scale. A value of 1 will mean that the main counter will increment once each cycle. A value of 100 will divide the total number of cycles by 100.
- **variable** - This is where the result will be placed

```
'pot example
dim x
loop:
  pot 0,4,x
  print x
  goto loop
```



print

NH N 485 A HS P

print value1,value2,value3,...
print exp,exp,exp,.....

Description

This command will print values to the debug terminal.

A cr/lf will automatically be added to the end of each command.

You may also specify text as long as it is contained in quotes. All numeric values will be converted for printout. Unlike other Athena commands the print command accepts full expressions and math.

Use the print command wisely as it is a memory hog.

- **value** - An expression that can contain any combination of numbers and variables. This can also be a quoted string. You may also supply various switches to change the way the value is displayed.

Switches

- **dec** - Does a binary to decimal conversion before printing the value. Note that the print command always does this conversion.
- **hex** - Does a binary to hexadecimal conversion before displaying the value. (Athena, AthenaHS and Nemesis only)

```
'print example
dim x
x = 25
print "-----"
print x
print hex x
print x + 1
```

pulsein

NH N 485 A HS P

pulsein port,state,varb
pulsein vcn,vcn,varb

Description

This command will measure a pulse. It will return a value of 1-255 for a valid count and 0 if the main counter over flows.

- **port** - The port to test.
- **state** - This sets the mode.
 - bit 0: The state to test 0=low 1=high.
 - bit 1: Force a complete cycle when set to 1. This can only be done for repetitive pulse trains.
- **variable** - This is where the result will be placed

Registers

- **PULSEINSCALE** (Athena / Athena485)

- 1: units = 4us
- 2: units = 8us
- 3: units = 16us (default)
- 4: units = 32us
- 5: units = 64us
- 6: units = 128us
- 7: units = 256us

- **PULSEINSCALE** (AthenaHS / NemesisHS)

- 1: units = .8us
- 2: units = 1.6us
- 3: units = 3.2us
- 4: units = 6.4us
- 5: units = 12.8us (default)
- 6: units = 25.6us
- 7: units = 51.2us

- **PULSEINSCALE** (Perseus & Nemesis @8mhz)

- 1: units = 2us
- 2: units = 4us
- 3: units = 8us (default)
- 4: units = 16us
- 5: units = 32us
- 6: units = 64us
- 7: units = 128us

- **PULSEINTIMEOUT**

When waiting for a pulse this parameter will increment. When it overflows a timeout will occur. Valid values are 0-255. The higher the number the faster the timeout. Default = 32.

'pulsein example

```
dim count
```

```
'Create a 1600us pulse  
hwpwm 2,200,100  
output 6
```

```
again:
```

```
pulsein 0,3,count  
print count  
goto again
```

pullupon (Athena & Nemesis) NH N 485 A HS

pullupon/pullupoff
pullupon/pullupoff

Description

Turns on the weak pullup resistors on the following ports:
Ports 2,3,4,5,6,7,11,12

'pullupon and pullupoff example

```
pullupon ' Turns on weak pullups
```

```
pause 200
```

```
pullupoff 'Removes the weak pullup resistors
```

pullupon (Perseus) P

pullupon port,port,...
pullupon number,number,...

Description

Turns on and off the weak pullup resistors. Unlike the Athena with the Perseus you can specify the ports you wish to pull high.

- **port** - The port you wish to pull high. Valid ports are 0,1,2,6,7

To turn off the ports you can issue the pullupoff command. If you don't specify any ports then all pullups will be turned on.

Perseus

'pullupon and pullupoff example

```
pullupon 0,1,2 ' Turns on weak pullups for ports 0-2
```

```
pause 200
```

```
pullupoff 'Removes the weak pullup resistors
```

pullupoff NH N 485 A HS P

pullupoff
pullupoff

Description

Turns on off the internal weak pullup resistors.

pulseout

NH N 485 A HS P

pulseout port,delay
pulseout vcn,vcn

Description

This will pulse the given port from its current state to the opposite state then back again.

You specify the amount of time the stay in the new state.

- **port** - This is the port you wish to pulse.
- **delay** - The number of microseconds to delay. Each unit is 3 microseconds. The pulse has a 21us overhead. 9us overhead on Perseus and Nemesis.

'pulseout example

```
output 0
low 0

loop:
pulseout 0,100
goto loop
```

pulseoutms

NH N 485 A HS P

pulseoutms port,delay
pulseoutms vcn,vcn

Description

This will pulse the given port from its current state to the opposite state then back again.

You specify the amount of time the stay in the new state.

- **port** - This is the port you wish to pulse.
- **delay** - The number of milliseconds to delay. The pulse has a 21us overhead.

'pulseoutms example

```
output 0
low 0

loop:
pulseoutms 0,2
goto loop
```

random

NH N A HS

random max,result
random vcn,varb

Description

Returns a pseudo-random number.

- **max** - This sets the maximum number. A number between 0 and max will be returned.
- **result** - This variable will contain the result. This number is also used to seed the random number generator.

'random example1

```
dim x

loop:
random 200,x 'Returns 0-200
print x
pause 255
goto loop
```

'random example2

```
dim x
high 0

loop:
output 0
pause 3
rccount 0,4,1,x
print x
goto loop
```

rccount

NH N 485 A HS P

rccount port,divisor,state,variable
rccount vcn,vcn,vcn,varb

Description

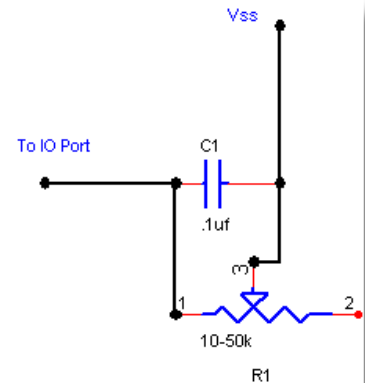
This command will set the given port to input and will increment its main counter until the port is no longer in the state given. The command will return a value of 1-255 for a valid count and 0 if the main counter over flows.

- **port** - The port to test.
- **divisor** - This is a scale. A value of 1 will mean that the main counter will increment once each cycle. A value of 100 will divide the total number of cycles by 100.
- **state** - The state you wish to measure.
- **variable** - This is where the result will be placed

'rccount example

```
dim x
high 0

loop:
output 0
pause 3
rccount 0,4,1,x
print x
goto loop
```



return

NH N 485 A HS P

return
return

Description

This command returns you to the point just after the gosub command. Note that if you issue a return command and a gosub command has not been called the Athena program will crash.

'return example

```
dim age

age = 10 : gosub printage
age = 20 : gosub printage
age = 30 : gosub printage

end

printage:
print age
return
```

serin

NH N 485 A HS P

serin timeoutlabel,port,varb,varb,varb,...
serin label,vcn,varb,varb,varb

Description

This command will wait for serial data on an IO port. This command expects the port provided to be in input mode. Use the setbaud command to set the baudrate registers.

- **label** - The label to jump to if no data is recieved within the timeout period. The default period is 125ms.

Note that you can supply a < or - for this label.

< will cause the command to jump to the beginning of the command when the command times out.

- will cause the command to fall through when the command times out.

'serin example

```
output 1
high 1
dim a

setbaud SBAUD4800

loop:
serin loop,0,a 'Input tied to port 0
serout 1,a 'Output tied to port 1
goto loop
```

serin / serout / servo

- **port** - The IO port you wish to use to send the data.
- **varb** - The variable to place the data.

Registers

- **SERTIMEOUTL** - Timeout low register default = 0.
- **SERTIMEOUTH** - Timeout high register default = 20.

Note: When using the serin command you must use the setbaud command to properly setup the baudrate.

'serout example

```
output 1
high 1
dim a

setbaud SBAUD4800

loop:
serin loop,0,a 'Input tied to port 0
serout 1,a 'Output tied to port 1
goto loop
```

serout

NH N 485 A HS P

serout port,value1,value2,value3,....
serout vcn,exp,exp,exp,.....

Description

This command will send serial values to any IO port. Use the setbaud command set the baud rate.

- **port** - The IO port you wish to use to send the data. This port must be in output mode and set to high.
- **value** - An expression that can contain any combination of numbers and variables. This can also be a quoted string.

'servo example

```
dim x
output 0

loop:
for x = 100 to 200
servo 0,x
pause 15
next
for x = 200 to 100 step -1
servo 0,x
pause 15
next

goto loop
```

servo

NH N 485 A HS P

servo port,servopulse
servo vcn,vcn

Description

servo will place a 10us - 2550us servo pulse on the indicated IO port. Note that you must make a call to the servo command about once every 20ms.

- **port** - IO port number to place a servo pulse on.
- **servopulse** - This is the length of the servo pulse. It is in 10us units.



Important

The serin and serout commands are implemented in software. The use of IRQ's can cause disruption to there use.

Currently the following commands can generate an interrupt.

timerASinternalclock,
timerASexternalclock,
timerAScounter, p7irq, p2irq

Also the receipt of data on the debug port can cause and interrupt

setbaud**NH N 485 A HS P**

setbaud baudrate
 setbaud vcn

Description

This command sets the baud rate for the software based serial IO.

- **baudrate** - The rate setting.

Valid rates:

SBAUD1200 (Athena / Athena485)
 SBAUD2400 (Athena / Athena485)
 SBAUD4800 (Athena / Athena485)
 SBAUD9600 (Athena / Athena485)
 SBAUD19200 (AthenaHS / NemesisHS)
 SBAUD57600 (AthenaHS / NemesisHS)

'setbaud example

```
output 1
high 1
dim a
```

```
setbaud SBAUD4800
```

loop:

```
serin loop,0,a 'Input tied to port 0
serout 1,a 'Output tied to port 1
goto loop
```

N P Perseus and Nemesis valid baud rate settings:

SBAUD1200-19200 Perseus @8Mhz
 SBAUD1200-9600 Perseus @4Mhz
 SBAUD1200-4800 Perseus @2Mhz
 SBAUD1200-2400 Perseus @1Mhz
 SBAUD1200 Perseus @500Khz

Note the Software serial command should not be used at frequencies below 500Khz.

setio**NH N 485 A HS P**

setio ioport,ioport,...
 setio number,number,...

Description

Special backdoor to Athena's IO ports. Provides a quick and fast way to set state of all the IO ports. No matter how many ports you configure command only takes 3 bytes.

- **ioport** - The IO port to set to the high state

N 485 A Ports 0-14 are valid for Athena and Nemesis.

NH HS Ports 0-12 are valid for AthenaHS.

P Ports 0-10 are valid for Perseus.

'setio example

'Sets ports 0-3 as output and high state.

```
configio 0,1,2,3
setio 0,1,2,3
```

shiftin / shiftout

shiftin

NH **N** **485** **A** **HS** **P**

shiftin IOport,CLKport,mode,varb
shiftin vcn,vcn,vcn,varb

Description

Loads a variable with data that is clocked in on IOpin.

- **IOport** - This is the data port used in the shift-in process. This port should be set to input.
- **CLKport** - This is the Clock port used in the shift-in process. This port should be set to output.

The **Mode** parameter can be broken down as:

Bits 0-3 Number of bits

Bit 4 NA
Bit 5 Clock transition. 0=low to high 1=high to low
Bit 6 0=Data then clock 1=Clock then data
Bit 7 Bit Order 0=LSB first 1=MSB first

- **varb** - This is where the shifted in value will be stored.

'shiftin example

```
dim retdat
```

```
const DAT 0  
const CLK 1  
const CS 2
```

```
output CS  
output CLK  
input DAT
```

loop:

```
gosub getdata  
print retdat  
pause 100  
goto loop
```

getdata:

```
low CS  
low CLK  
pulseout CLK,1  
shiftin DAT,CLK,200,retdat  
high CS  
return
```

shiftout

NH **N** **485** **A** **HS** **P**

shiftout IOport,CLKport,mode,value
shiftout vcn,vcn,vcn,vcn

Description

Sends a sequence of bits out the IOpin. A clock pulse is sent out for each bit via the CLKpin.

- **IOport** - This is the data port used in the shift-out process. This port should be set to output.
- **CLKport** - This is the Clock port used with the shift-out process. This port should be set to output.

The **Mode** parameter can be broken down as:

Bits 0-3 Number of bits

Bit 4 NA
Bit 5 Clock transition. 0=low to high 1=high to low
Bit 6 NA
Bit 7 Bit Order 0=LSB first 1=MSB first

- **value** - This 8 bit value will be shifted out.

' shiftout example

```
dim x,y
```

```
const latch595 2  
const dat595 0  
const clk595 1
```

```
output latch595  
high latch595  
output clk595  
output dat595
```

```
dim dat
```

loop:

```
x = 1  
for y = 1 to 8  
  shiftout dat595,clk595,136,x 'Send data to 75HC595  
  pulseout latch595,1  
  pause 50  
  x = x * 2  
next
```

```
goto loop
```



shiftin/shiftout clock frequencies

Athena 12Khz
Athena485 12Khz
AthenaHS 60Khz
Perseus 24Khz
Nemesis 24Khz

signal

NH N 485 A HS P

signal port,cycles,period
signal vcn,vcn,vcn

Description

signal will output a pulse train to a given port. You can use it to generate tones or other signals.

- **port** - IO port number to place signal on.
- **cycles** - the number of pulses to output.
- **period** - length of 1 complete cycle in 18.15us units. Note that there is a 123us overhead. range is 1-255.

Registers

- **SIGNALRES** - allows you to change the resolution of the signal command.

Bits 0-3 allow you to change the period resolution.

- xxxx0000 - each unit = 6.05us with an overhead of 82us.
- xxxx0001 - each unit = 12.05us with overhead of 103us.
- xxxx0011 - each unit = 18.15us with overhead of 123us. (default)
- xxxx0111 - each unit = 24.2us with overhead of 143us.
- xxxx1111 - each unit = 30.35us with overhead of 162us.

Bits 4-7 allow you to change the cycles resolutuon

- 0000xxxx - Cycle units 1:1. Will cycle 1 x the amount given.
- 0001xxxx - Cycle units 2:1. Will cycle 2 x the amount given.
- 0011xxxx - Cycle units 4:1. Will cycle 4 x the amount given.
- 0111xxxx - Cycle units 8:1. Will cycle 8 x the amount given. (default)
- 1111xxxx - Cycle units 16:1 Will cycle 16 x the amount given.

'signal example

```
loop:
signal 0,50,7
signal 0,25,20
goto loop
```

sleep

NH N 485 A HS P

sleep
sleep

Description

Place the Athena or Perseus in low power sleep mode.

'sleep example

```
configio 1,0
p7irq 3 'This will wake up Athena with port 7
```

loop:

```
print "going to sleep"
sleep
print "wake up"
```

```
goto loop
```

srin / srinport / srout

srin

NH N 485 A HS P

srin data,clock,load,resultvarb
srin vcn,vcn,vcn,varb

Description

Reads a byte from a 74165 shift register.

- **data** - DI pin on 165 chip (pin 9).
- **clock** - CLK pin on 165 chip (pin 2).
- **load** - Load pin on 165 chip (pin 1).
- **resultvarb** - The variable to place the byte.

'srin example

```
dim dat
```

```
loop:  
srin 0,1,2,dat  
print dat  
goto loop
```

srinport

NH N A HS P

srin data,clock,load,port,resultvarb
srin vcn,vcn,vcn,port,varb

Description

Reads a single port on a 74165 shift register.

- **data** - DI pin on 165 chip.
- **clock** - CLK pin on 165 chip.
- **load** - Load pin on 165 chip.
- **port** - The Port to read 0-7 on the 165 shift register.
- **resultvarb** - The variable to place the byte.

'srinport example

```
const load165 2  
const dat165 0  
const clk165 1
```

```
dim dat,x
```

```
loop:  
for x = 0 to 7  
srinport dat165,clk165,load165,x,dat  
print x,"=",dat  
next  
print  
pause 255  
goto loop
```

srout

NH N 485 A HS P

srout data,clock,latch,value
srout vcn,vcn,vcn,vcn

Description

Writes a byte to a 74595 shift register. This command will place the ports into the proper state.

- **data** - DI pin on 595 chip (pin 14).
- **clock** - CLK pin on 595 chip (pin 11).
- **latch** - Latch pin on 595 chip (pin 12).
- **value** - The byte to write to the 595 shift register.

'srout example

```
dim x
```

```
loop:  
for x = 0 to 255  
srout 0,1,2,x  
pause 10  
next  
  
goto loop
```

srouport

NH N A HS P

srouport data,clock,latch,statusvarb,port,value
srouport vcn,vcn,vcn,vcn,varb,vcn,vcn

Description

Set a single port on a 74595 shift register. This command will place the ports into the proper state.

- **data** - DI pin on 595 chip.
- **clock** - CLK pin on 595 chip.
- **latch** - Latch pin on 595 chip.
- **varb** - This variable is used to keep track of which ports have been set.
- **port** - The port 0-7 on the shift register to set or reset.
- **value** - If 0 then port will be reset. If 1 then port will be set.

```
'srouport example

dim buffer,y

const latch595 2
const dat595 0
const clk595 1

clear buffer

loop:
for y = 0 to 7
srouport dat595,clk595,latch595,buffer,y,1
pause 100
srouport dat595,clk595,latch595,buffer,y,0
next

for y = 7 to 0 step -1
srouport dat595,clk595,latch595,buffer,y,1
pause 100
srouport dat595,clk595,latch595,buffer,y,0
next

goto loop
```

startcaptimer

P

startcaptimer chargetime
startcaptimer number

Description

This command charges a capacitor on port 0 then starts a 1 shot irq that allows it to fire when the capacitor has discharged to a certain point.

- **chargetime** - Time is in milliseconds. You should start around 10ms for

1uf = 0m:7s
 10uf = 1m:17s
 100uf = 12m:52s

These times are independent of the system clock.

```
Perseus
'startcaptimer demo

loop:
startcaptimer 100 'Good for 100uf cap

print "sleep"
sleep
print "wake"
print PORTCOUNTER
goto loop
```



startasm

startasm
startasm

Description

The Nemesis microcontroller gives you the ability to insert Kronos assembly language commands.

The Kronos assembler is very sophisticated and works much like the Athena Basic language. In many cases there is no difference in the commands except the fact that they run 30-60 times faster.

To start an assembly block use the startasm command. To end the block use the endasm command. All commands between these two commands will be assembled. They will be executed when the Athena engine reaches the startasm command. This means you can jump in and out of the assembler at will.

There are differences in syntax between the Kronos assembler and Athena language. The following are the exceptions:

Labels

All labels must be placed on a separate line and must start on the first column of that line. All labels must end with a :

Labels are case sensitive and can be lower, upper or mixed case.

Using the jump command you can jump to labels defined outside of the assembler. In other words you may jump to an Athena language label.

Note that if you receive the following error:
Symbol not previously defined (Label_myloopy)

This is telling you that the Athena label myloopy does not exist. Click the goto error button and the cursor will be placed on the line that attempted to make the call.

Commands

Each command must be placed on a separate line. Commands may not be placed on the first column of a line. The first column is reserved for labels.

Variables

There are two types of variables for use in your assembly routines.

Variables that are defined in the Athena language and variables that are defined in the assembler.

To create a variable inside the assembler use the dim statement within the startasm and endasm commands.

Nemesis
'startasm example

```
startasm
dim a
a = 25
add a,25
print a
```

endasm

Nemesis
'startasm example

```
startasm
output 0
loop:
high 0
low 0
goto loop
```

endasm

Nemesis
'startasm example

```
startasm
print "Hello world"
```

endasm



Note: While the print, debug and serout commands are similar to those in the Athena language they don't support the formatting options.

Nemesis

```
dim AthenaVarb
AthenaVarb = 25
```

```
startasm
dim AssemblyVarb
AssemblyVarb = 75
```

```
print AthenaVarb," ",AssemblyVarb
```

endasm

The Kronos assembler gives you an additional 96 variables that you may define. Note that you can not access the assembly variables from the Athena language.

Registers

Most of the Nemesis hardware and software registers are available and accessible just like variables.

All registers are in upper case to make them consistent with the Athena language.

Comments

The assembler uses the same comment character as the Athena language. Any thing following the ' will be ignored.

Accessing Ports

You set ports direction and state with the input,output,high or low commands.

To test a port use the inpX statement where X is the port number 0-14.

Expressions

In assembly there are no true expressions like there are in the Athena language.

You **can not** use the expression $A = 4 + 5$.

In assembly you must use one of the math commands as in:

```
A= 4
add A,5
```

To perform math on a variable you use the math commands.

```
shiftright
shiftright
inc
dec
add
sub
mul
div
and
or
xor
```

```
Nemesis
startasm

dim a

loop:
a = inp1
print a
goto loop

endasm
```



You can use the inpX command to test the state of an IO port. X is the port number 0-14.



Some commands like mul and div actually call the Athena language math routine

KRAssembler Commands

KRAssembler Commands

Syntax Notation

The following notation is used to represent the various arguments for the commands.

V	Variable or Register
n	Number or Constant
vcrn	Variable, Constant, Register, Number
vcrrt	Variable, Constant, Register, Number, Quoted text
label	Assembly label
Alabel	Athena label
...	Indicates you can have as many arguments as you wish.

add Add to a register or variable

syntax `add varb,value`
 add V,vcrn

Operands: **varb**: Variable or register to add to.

value: Value to add to Variable. 8-bit.

Description: This command adds a value to the variable or register.

add16 Does a 16-bit add to a register or variable

syntax `add16 varbL,varbH,valueL [,valueH]`
 add16 V,V,vcrn [,vcrn]

Operands: **varbL**: Variable that hold low byte that you want to add to.

varbH: Variable that hold high byte that you want to add to.

valueL: Low byte to add to low/high variables.

valueH: High byte to add to low/high variables. Optional

Description: This command does a 16-bit add to a high/low register or variable pair. Note that you can pass as single 16-bit number and it will break it down into the high/low pair for you. As in :**add16 VL,VH,1000**

and And a value to a register or variable

syntax `and varb,value`
 and V,vcrn

Operands: **varb**: Variable or register to and to.

value: Value to and with Variable. 8-bit.

Description: This command ands a value to the variable or register.

arrayget **Read from a block of variables**

syntax `arrayget Vstart,Index,ResultVarb`
`arrayget V,vcrn,V`

Operands: **Vstart:** Variable that marks the starting point into bank.
Index: Index to byte you wish to retrieve.
ResultVarb: The variable or register to place the retrieved value into.

Description: This will allow you to pull a byte from a set of variables using and index variable.

arrayset **Write to a block of variables**

syntax `arrayset Vstart,Index,Value`
`arrayset V,vcrn,vcrn`

Operands: **Vstart:** Variable that marks the starting point into bank.
Index: Index to byte you wish to write to.
Value: The byte value that you wish to write.

Description: This will allow you to write a byte to a set of variables using and index variable.

bitset **Set a bit**

syntax `bitset Varb,Bit`
`bitset V,n`

Operands: **Varb:** Variable that holds the bit you wish to set.
Bit: Bit to set

Description: Sets a bit in a variable or register.

bitreset **Reset a bit**

syntax `bitreset Varb,Bit`
`bitreset V,n`

Operands: **Varb:** Variable that holds the bit you wish to reset.
Bit: Bit to reset

Description: Resets a bit in a variable or register.

KRAssembler Commands

checkbuff

Check the UART Buffer

syntax `checkbuff Vstat,Vdat`
 checkbuff V,V

Operands: **Vstat**: Bit 0 will be set if data is available.

Vdat: If data available will contain the next byte

Description: Checks the UART buffer. If data is in the buffer the Vstat variable's bit 0 will be set and the Vdat variable will contain the byte.

debug

Send data to debug terminal

syntax `debug data,.....`
 debug vcrnt,.....

Operands: **data**: Data element to send to the debug port. You may also send quoted data and each character will be sent.

Description: Sends one or more data elements to the debug port.

dec

Decrement a Variable

syntax `dec varb`
 dec V

Operands: **varb**: Variable to decrement.

Description: This commands subtracts 1 from the variable.

dec16

16-bit Decrement

syntax `dec16 varbL,varbH`
 dev16 V,V

Operands: **varbL**: Variable that holds low byte that you want to decrement.

varbH: Variable that holds high byte that you want to decrement.

Description: This command does a 16-bit decrement on a High/Low variable or register pair.

dim Create a KRAssembly Variable

syntax `dim Varb,Varb,.....`
`dim V,V....`

Operands: **varb**: Variable to create.

Description: The dim statement allows you to create a KRAssembly variable.
variables declared with the dim statement by KRAssembly are global and are accessible by all KRAssembly routines.
Checkout the redim statement to create local variables.

div Divide a Register or Variable

syntax `div varb,value`
`div V,vcrn`

Operands: **varb**: Variable or register to be divided.
value: Amount to divide. 8-bit.

Description: This commands divides a Register or Variable by a value.

eeread Read from EEprom

syntax `eeread Addr,Varb`
`eeread vcrn,V`

Operands: **Addr**: Eeprom address. 0-255

Description: **Varb**: The variable to place the read byte into.
This command reads a byte from the on-board EEprom.

eewrite Write to EEprom

syntax `eewrite Addr,Value`
`eewrite vcrn,V`

Operands: **Addr**: Eeprom address. 0-255

Description: **Value**: The value to write to the EEprom.
This command writes a byte to the on-board EEprom.

KRAssembler Commands

exit Exit from KRAssembly

syntax exit
 exit

Description: When your program reaches the endasm command it will exit and return to the Athena language. This command lets you exit to the endasm command from anywhere in your assembly block.

gosub Call a KRAssembly Routine

syntax gosub Label
 gosub Label

Description: Calls and assembly subroutine. You must use the return command to exit the assembly routine.

goto Jump to a KRAssembly Routine

syntax goto Label
 goto Label

Description: Jumps to another assembly location.

high Set Port High

syntax high Port
 high n

Operands: **Port:** The port to set high. Note the port must be in output mode. You may only use a number or constant here.

Description: Sets a port high.

if / then / else / endif

Test a Condition

syntax

```
if condition1 operator condition2 then
if V operator vcrn then
```

Operands:

Condition1: The item to test. This can be any variable or register. Note that you may test bits of a variable or constant by specifying the bit with the syntax: register,bit

Condition2: The item to test against. This can be any variable or register number or constant. You can not use a bit here.

operator: This is the kind of comparison. You can use the following:

<, >, <=, >=, <>, !=, =, ==

Description:

This command allows you to test various states and conditions then process code based on the results. You may use the else statement to process alternate code if the condition fails.

Example:

```
if PORTB,0 = 1 then
  print "its high"
else
  print "its low"
endif
```

inc

Increment a Variable

syntax

```
inc varb
inc V
```

Operands:

varb: Variable to increment.

Description:

This commands adds 1 to the variable.

inc16

16-bit Increment

syntax

```
inc16 varbL,varbH
inc16 V,V
```

Operands:

varbL: Variable that holds low byte that you want to increment.

varbH: Variable that holds high byte that you want to increment.

Description:

This command does a 16-bit increment on a High/Low variable or register pair.

KRAssembler Commands

input **Set Port to Input Mode**

syntax input Port
 input n

Operands: **Port:** The port to set to input. You may only use a number or constant here.

Description: Places the port into input mode.

inp **Test a Port**

syntax inpX
 inpX

Operands: **X:** The port to set to test. It must be a number.

Description: This command is intended to be used with the print and if/then as well as the while/wend command.

jump **Jump to an Athena Label**

syntax jump label
 or
 jump varbL,varbH

Description: You can jump to a declared Athena label or to an actual address in two variables.

lcdchar **Sends a character to a LCD**

syntax lcdchar character
 lcdchar vrcn

Operands: **Character:** The single character you wish to send to the LCD.

Description: Sends a single character to an LCD. The LCD must be initialized with the Athena language lcdinit command.

lcdcontrol **Sends a control code to a LCD**

syntax lcdcontrol code
 lcdcontrol vrcn

Operands: **code:** The single code you wish to send to the LCD.

Description: Sends a single control code to an LCD. The LCD must be initialized with the Athena language lcdinit command.

low **Set Port low**

syntax low Port
 low n

Operands: **Port:** The port to set low. Note the port must be in output mode. You may only use a number or constant here.

Description: Sets a port low.

mul **Multiply a Register or Variable**

syntax mul varb,value
 mul V,vcrn

Operands: **varb:** Variable or register to be multiplied.

value: Amount to multiply. 8-bit.

Description: This commands multiplies a register or variable by a value.

or **Or a value to a register or variable**

syntax or varb,value
 or V,vcrn

Operands: **varb:** Variable or register to or to.

value: Value to or with Variable. 8-bit.

Description: This commands or's a value to the variable or register.

output **Set Port to Output Mode**

syntax output Port
 output n

Operands: **Port:** The port to set to output. You may only use a number or constant here.

Description: Places the port into output mode.

KRAssembler Commands

pause **Pause in 1ms units**

syntax `pause Value`
`pause vcrn`

Operands: **Value:** The amount to pause. 0-255 ms.

Description: Cause a delay from 0-255 ms.

pauseus **Pause in 3us units**

syntax `pauseus Value`
`pauseus vcrn`

Operands: **Value:** The amount to pause. 0-255 x 3 us.

Description: Cause a delay from 0-765 us.

print **Send data to debug terminal**

syntax `print data,.....`
`print vcrnt,.....`

Operands: **data:** Data element to send to the debug port. You may also send quoted data and each character will be sent.

Description: Sends one or more data elements to the debug port. The print command does a little formatting for you. It will automatically convert any single value to decimal. It will also add a CR/LF to the end unless you place a ; on the end of the command string.

redim Create a reusable KRAssembly Variable

syntax `redim Varb,Varb,.....`
`redim V,V....`

Operands: **varb**: Variable to create.

Description: The redim statement allows you to create a reusable KRAssembly variable. There are 96 bytes available for creation in the KRAssembler. Each time the dim statement is used one byte is allocated off the top for each variable created as shown in Figure 1.

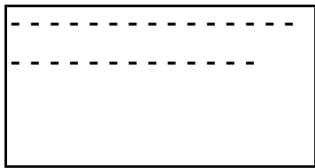


Figure 1

These are global and will not change during the life of the program.

With the redim statement one byte is allocated off the bottom for each variable created as shown in Figure 2.

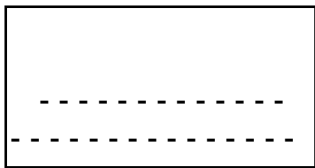


Figure 2

Each time the redim is issued it starts from the very bottom overwriting the variables from the last redim. This makes them perfect for creating assembly routines that are called from the Athena language.

KRAssembler Commands

return **Return from KRAssembler Subroutine**

syntax `return`
 return

Description: Use to return from a routine that you called with a gosub command.

serout **Send Serial Data to any IOport**

syntax `serout port,data,.....`
 serout vcrn,vcrnt,.....

Operands: **port**: The port to send the data to. This port must be in output mode.

Description: **data**: Data element to send to an IO port. You may also send quoted data and each character will be sent.

Sends one or more data elements to a IOport.

shiftright **Shift Bits Right**

syntax `shiftright varb`
 shiftright V

Operands: **varb**: Variable or register to shift.

Description: This commands shifts the bits in a variable or register left.

shiftright **Shift Bits Right**

syntax `shiftright varb`
 shiftright V

Operands: **varb**: Variable or register to shift.

Description: This commands shifts the bits in a variable or register right.

sub Subtract from a register or variable

syntax `sub varb,value`
`sub V,vcrn`

Operands: **varb**: Variable or integer to subtract from.

value: Value to subtract from Variable. 8-bit.

Description: This commands subtracts a value from a variable or register.

sub16 Does a 16-bit subtract from a register or variable

syntax `sub16 varbL,varbH,valueL [,valueH]`
`sub16 V,V,vcrn [,vcrn]`

Operands: **varbL**: Variable that hold low byte that you want to subtract from.

varbH: Variable that hold high byte that you want to subtract from.

valueL: Low byte to subtract from the low/high variables.

valueH: High byte to subtract from the low/high variables. Optional

Description: This command does a 16-bit subtract from a high/low register or variable pair
Note that you can pass as single 16-bit number and it will break it down into the high/low pair for you. As in : **sub16 VL,VH,1000**

toggle Toggle a IO Port

syntax `toggle Port`
`toggle n`

Operands: **Port**: The port to toggle. Note the port must be in output mode. You may only use a number or constant here.

Description: If the port is high this command will set it low. If low it will be set high.

waitport Waitfor a condition on port

syntax `waitport Port,State,Timeout,Label`
`waitport vcrn,vcrn,vcrn,label`

Operands: **Port**: The port to test.

State: The state to waitfor.

Description: **Timeout**: Adds a bit of time out before label call.

Label: The location to jump to if condition is not met.

The command will fall through if the state is met.

KRAssembler Commands

while / wend

Loop on a Condition

syntax	<code>while condition1 operator condition2</code> <code>while V operator vcrn</code>
Operands:	<p>Condition1: The item to test. This can be any variable or register. Note that you may test bits of a variable or constant by specifying the bit with the syntax: register,bit</p> <p>Condition2: The item to test against. This can be any variable or register number or constant. You can not use a bit here.</p> <p>operator: This is the kind of comparison. You can use the following:</p> <p><, >, <=, >=, <>, !=, =, ==</p>
Description:	<p>This command allows you to test various states and conditions. As long as the condition is true the commands between the while and wend will be executed</p> <p>Example:</p> <pre>dim5 x x = 0 while x < 10 print x inc x wend</pre>

xor Xor a value to a Register or Variable

syntax	<code>xor varb,value</code> <code>xor V,vcrn</code>
Operands:	<p>varb: Variable or register to xor to.</p> <p>value: Value to xor with Variable. 8-bit.</p>
Description:	This commands xor's a value to the variable or register.

startasmcommand**NH** **N**

startasmcommand
startasmcommand

Description

The startasmcommand command works much like the startasm command. The difference is that they are not issued in place but are given names and are called by the athena language.

Please refer to the startasm command for a description of the KRAssembler and its commands.

Advanced Usage

you can also pass simple arguments to these routines. You must tell the Athena engine what kind of arguments you wish to pass.

The actual usage for these options are very advanced and require a through understanding of how parameters and expressions are stored in the Athena Engine.

This command is intended for use by Kronos Robotics to provide additional command support for various applications and projects.

At some point in the future we may write a manual to goes into detail enough for customers to build there own commands and pass and return command data.

Please check the Nemesis forums on the web site for more information if you want to experiment with these commands.

Nemesis

```
dohello
dohello
dohello
```

```
startasmcommand dohello
  print "Hello"
endasmcommand
```

Nemesis

```
dotext "hello"
```

```
startasmcommand dotext(text)
  redim bytes,char
```

```
  getbyte bytes
```

```
  while bytes > 0
    getbyte char
    debug char
    dec bytes
  wend
```

```
  debug 13,10
```

```
endasmcommand
```

systemclock

N P

systemclock clock
systemclock number

Description

You can change the Perseus or Nemesis system clock on the fly. By changing the frequency you can lower battery consumption.

• clock - Valid clock settings:

0=32Khz
1=125Khz
2=250Khz
3=500Khz
4=1Mhz
5=2Mhz
6=4Mhz
7=8Mhz (default)

Note that changing the system clock will affect timing of certain commands. For instance if you change system clock you will need to issue the debugbaud command to reset the baud rate for debug.

```
Perseus
'System clock example

systemclock 6 'Change to 4Mhz
```

timerAScounter

NH N 485 A HS P

timerAScounter
timerAScounter

Description

This command sets up the built-in timer hardware as a counter. The counting takes place on port 3 of the Athena, Athena485, AthenaHS and Nemesis. On the Perseus it takes place on port 6.

```
'timerAScounter example

dim Tlo,Thi
'32470Hz
hwpwm 2,24,12
output 6

'Counter mode on port 3
loop:
timerAScounter
pause 10
timeroff
timerget Tlo,Thi
print Tlo," ",Thi

goto loop
```

timerASexternalclock

NH N 485 A HS P

timerASexternalclock
timerASexternalclock

Description

This command sets up the built-in timer as an external 24 hour clock. Requires a 32khz crystal connected to IO ports 2 and 3 of the Athena, Athena485, AthenaHS and Nemesis. On the Perseus use ports 6 and 7.

Note: You may need to place a 22-47pf capacitor across each port and Vss.

```
'timerASexternalclock example

dim secs,mins,hours

timerASexternalclock
timersetclock 12,21,00

loop:
timergetclock hours,mins,secs
print hours,":",mins,":",secs
goto loop
```

timerASinternalclock NH N 485 A HS P

timerASinternalclock
timerASinternalclock

Description

This command sets up the built-in timer as a internal 24 hour clock.

'timerASinternalclock example

```
dim secs,mins,hours
```

```
timerASinternalclock
```

```
timersetclock 11,19,30
```

```
loop:
```

```
timergetclock hours,mins,secs
```

```
print hours,":",mins,":",secs
```

```
goto loop
```

timercfg NH N 485 A HS P

timercfg value
timercfg vcn

Description

This command sets up the built-in timer. Normaly its used to clear the internal timer register.

- **value** - The timer register value.

Typical values

0: Sets up timer 1:1 prescale

16: Sets up timer 1:2 prescale

32: Sets up timer 1:4 prescale

48: Sets up timer 1:8 prescale

'timercfg and timerget example

```
dim Tlo,Thi
```

```
timercfg 0
```

```
loop:
```

```
timeron
```

```
pause 2
```

```
timeroff
```

```
timerget Tlo,Thi
```

```
print Tlo," ",Thi
```

```
goto loop
```

timerget NH N 485 A HS P

timerget TMRL,TMRH
timerget varb,varb

Description

Get the internal timer values

timergetclock NH N 485 A HS P

timergetclock hours,minutes,seconds
timergetclock varb,varb,varb

Description

Get the internal timer values when in clock mode

Returns:

seconds - 0-59

minutes - 0-59

hours - 0-24

'timergetclock example

```
dim secs,mins,hours
```

```
timerASinternalclock
```

```
timersetclock 11,19,30
```

```
loop:
```

```
timergetclock hours,mins,secs
```

```
print hours,":",mins,":",secs
```

```
goto loop
```

timeroff

NH N 485 A HS P

timeroff
timeroff

Description

This command stops the built-in timer.

timeron

NH N 485 A HS P

timeron
timeron

Description

This command starts the built-in timer.

timerset

NH N 485 A HS P

timerset TMRL,TMRH
timerset vcn,vcn

Description

Set the internal timer values

timersetclock

NH N 485 A HS P

timersetclock hours,minutes,seconds
timersetclock vcn,vcn,vcn

Description

Set the internal timer when in clock mode

seconds - 0-59
minutes - 0-59
hours - 0-24

'timeroff and timeroff example

```
dim Tlo,Thi

timercfg 0
loop:
timeron
pause 2
timeroff
timerget Tlo,Thi
print Tlo," ",Thi

goto loop
```

'timersetclock example

```
dim secs,mins,hours

timerASinternalclock

timersetclock 11,19,30

loop:
timergetclock hours,mins,secs
print hours,":",mins,":",secs
goto loop
```

toggle

NH N 485 A HS P

toggle port
toggle vcn

Description

The toggle command is used to reverse the current state of a port.

- **port** - The port number of the IO port to change. Can be variable/constant/number.

```
'toggle example
```

```
output 0
```

```
loop:
toggle 0
nop 'Small delay
goto loop
```

TW523read

NH N 485 A HS P

TW523read nodatalabel,zeroport,inputport,housecode,keycode
TW523read label,vcn,vcn,varb,varb

Description

This command will allow you to receive X10 codes from a TW523 interface. You must hold the zeroport and inputport high with a 10k resistor.

- **nodatalabel** - label to jump to if no data is available. If data is available it will be placed in house code and key code variables.

Note that you can supply a < or - for this label.

< will cause the command to jump to the beginning of the command when the command fails.

- will cause the command to fall through when the command fails.

- **zeroport** - The port connected to pin 1 on the TW523 interface.
- **inputport** - The port connected to pin 3 on the TW523 interface (X10 out).
- **housecode** - A variable that will hold the house code value.
- **keycode** - A variable that will hold the key code value.

Note

Always follow then TW523read command with a pauseus 100 command. This will assure that you don't reenter the the TW523 command too fast. If extensive processing or displaying is taking place you may leave the delay out.

Note: The Perseus must be at 8mhz system clock when using the TW523 commands.

```
'TW523read example
```

```
const TW523Zero 0
const TW523in 1
```

```
dim TW523house,TW523keycode
```

```
loop:
TW523read loop,TW523Zero,TW523in,TW523house,TW523keycode
pauseus 100 'Slow things down a bit
print "house=",TW523house
print "code=",TW523keycode

goto loop
```

TW523write

NH N 485 A HS P

TW523write zeroport,outputport,housecode,keycode
TW523write vcn,vcn,vcn,vcn

Description

This command will allow you to send X10 codes to a TW523 interface. You must hold the zeroport and outputport high with a 10k resistor.

- **zeroport** - The port connected to pin 1 on the TW523 interface.
- **outputport** - The port connected to pin 4 on the TW523 interface (X10 in).
- **housecode** - The house code value to send to the remote X10 device.
- **keycode** - The unit or command code to send to the remote X10 device.

'TW523write example

```
const TW523Zero 0
const TW523in 1
const TW523out 4
```

```
TW523write TW523Zero,TW523out,6,7 'Device A-2
pause 50 'Need a delay between commands
TW523write TW523Zero,TW523out,6,20 'Turn on command
```

waitport

NH N A HS

waitport port,state,timeout,timeoutlabel
waitport vcn,vcn,vcn,label

Description

Will wait for a particular state. if a timeout occurs a jump to timeout label will occur.

- **port** - The port to test.
- **state** - The state to wait for.
- **timeout** - The number of tests * 256 to make before we cause a timeout.
- **timeoutlabel** - The label to jump to if a timeout occurs.

'waitport example

```
loop:
  print "Loop"
  waitport 0,1,255,loop

  print "Have it"

  goto loop
```

watchdog

NH N P

watchdog setting
waitport number

Description

The watchdog can be used to wake up or reset the Perseus

- **setting** - The setting determines the watchdog timer interval.

0 = Watchdog Off

1 = 67ms

2 = 135ms

3 = 270ms

4 = 540ms

5 = 1.08s

6 = 2.17s

7 = 4.34s

8 = 8.68s

9 = 17.37s

10 = 34.75s

11 = 69.5s

12 = 139s

13 = 278s

```
Perseus
'Watch dog timer
```

```
watchdog 6
```

```
loop:
print "sleep"
sleep
print "wake"
goto loop
```

Simulator

The Athena compiler has a built-in simulator that allows you to test your code without ever connecting to a chip.

Note:

The simulator does not currently have Nemesis support

Starting The Simulator

To start the simulator click the simulate button. Your program will compile and provided you did not have any errors it will enter simulator mode.

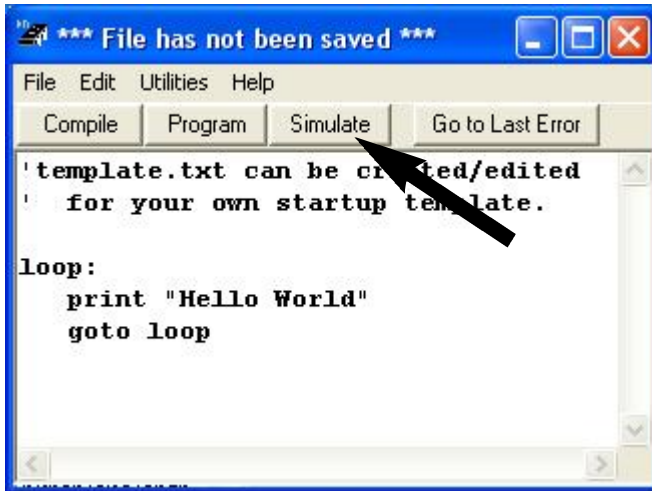







Figure 5.1
Athena Simulate **Button**

Simulator Control

Once the simulator has started the simulator is in Simulator Mode. The simulator controls will be displayed. Also the current program instruction will be highlighted.



Figure 5.2
Athena Simulator Mode

-  Stop the simulator. Returns the program back to Editor Mode. The **ESC** key will also stop the simulator.
-  Run the program in continuous mode. The **Space** key will also toggle the run and pause modes.
-  Pause the program. This is the default starting mode. The **Space** key will also toggle the run and pause modes.
-  Single Step the program. Will cause the program to jump to next instruction. Only valid when paused. The **Enter** Key can also be used to single step.
-  Resets the program to starting point. The **R** key can also be used to reset the program.



Sets the run speed when in continuous mode.

Note that the **F1** Key can also be used to pop up help on the current instruction.

Runto Option

Any time you click on a command the simulator will start running in continuous mode.

It will continue to run until it reaches the command you clicked.

Note that it is possible to click a command that can not be reached. In cases like this the program will not stop.

You can pause the program normally any time you use the runto option.

Simulator Form

Once the simulator starts the simulator form will be displayed. The directive you choose will determine which chip is displayed.

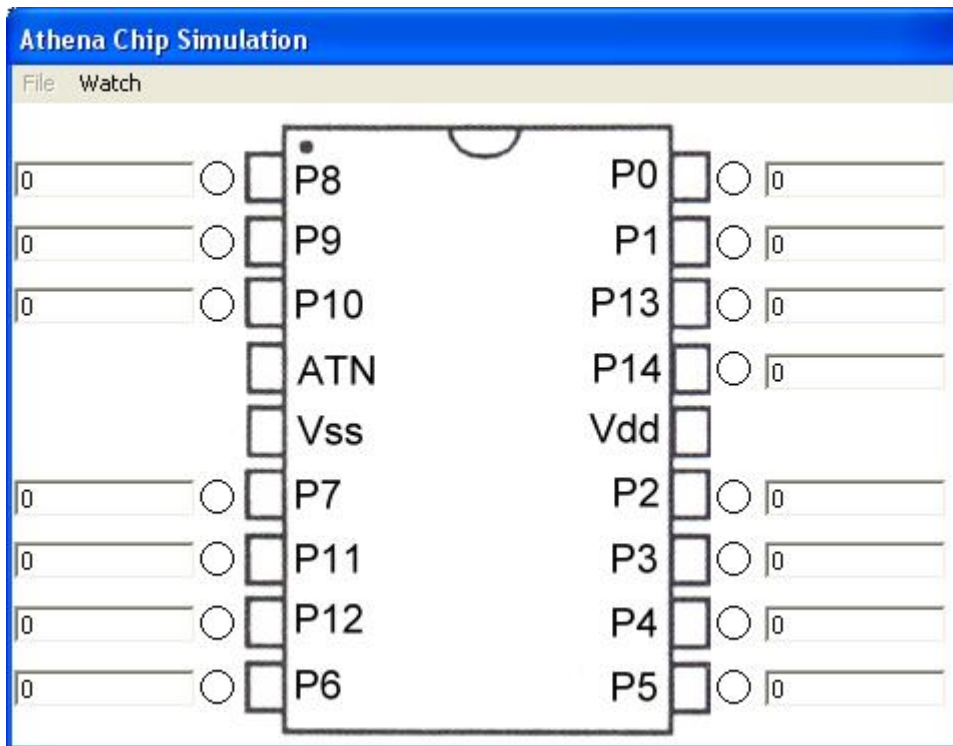


Figure 5.3
Athena Simulator Form

Next to each port is a small LED. The color of this LED indicates what mode the IO port is in.

- White = Input mode
- Grey = Output mode Low state
- Red = Output mode High state

Also associated with each port is the stimuli data field.

Each time the port is read this is the value the port will return. You can quickly change this data by double clicking on the field. If the value was 0 it will be changed to a 1 and visa versa.

You can also supply multiple readings for a port.

Each time the status is requested the next reading in the list will be returned. Once the end is reached it wraps back to the beginning.

Some commands read data other than simple IO state. For example the pulsein command returns a number between 0 and 255 depending on the length of the pulse it is measuring.

Other commands like the atod command are looking for a level reading.

Commands like shift in are looking for shifted input data. In order to accommodate these commands and others you can supply more complex data to the stimuli field.

```
100,110,130
```

Like the port state data this data will be read in by the command each time it is executed. For commands that are associated with multiple ports it will always be the data port that returns the data.

Loading Stimuli Data

Figure 5.4
Stimuli preload

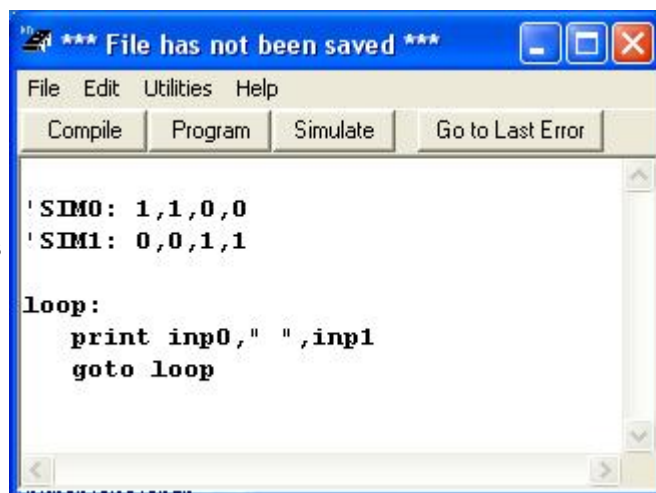
You can preload simulator stimuli data by creating comments in your source code in the form of 'SIMX: data,data,dat.... as shown in figure 5.4

You can also add a switch that sets the simulator atod to simple 8 bit mode.

```
'SIMATOD=SIMPLE
```

This switch forces simulator data to 8 bit mode only and does not update ADRESH or ADRESL registers. This mode is good for use in simple Atod simulation and you don't want to mess with converting to 10bit in stimuli data.

The stimuli data is loaded as soon as the simulator is started. Once loaded you can still change the data in the stimuli fields.



Stimuli Options

There are a couple of smart stimuli settings you can place in the data.

Repeat Option X:Y Will repeat Y value X times when requested.

Random Option X~Y Will return a random number between X and Y.

You can combine the two options as in 6:1~5. This will return 6 random numbers between 1 and 5 when requested.

Simulator LCD

To load the LCD form select the Show LCD command from the Watch menu on the Simulator form.



Figure 5.5
Show LCD

The LCD supports most of the LCD control codes used on a Hitachi 44780 controller. Note that the character generator codes are not supported.

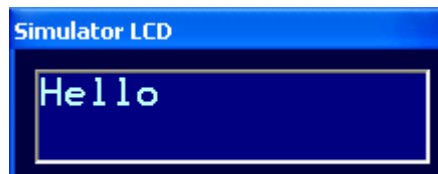


Figure 5.6
LCD Form

Variable Watch

You can watch all the variables in your program by opening the variable watch form. You do this by selecting the Show Variables from the Watch menu on the Simulator form.



Figure 5.7
Load Variable Watch

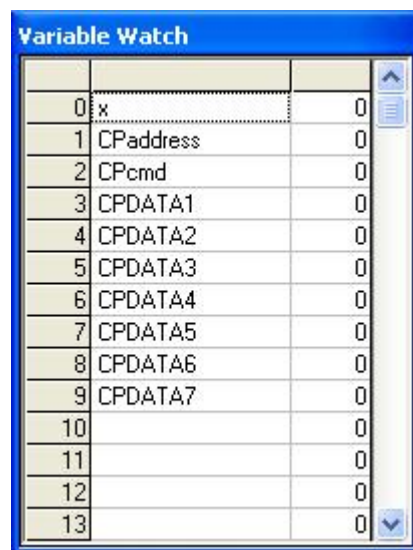


Figure 5.8
Variable Watch Form

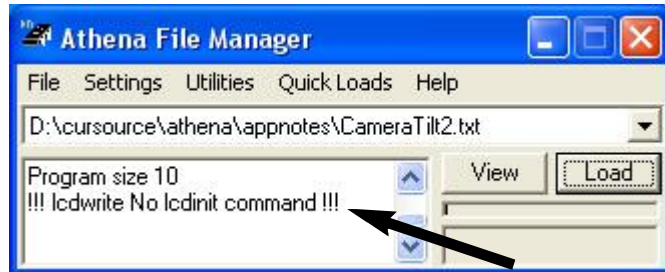
To change a variable value just double click on the variable name.

Simulator output

Simulator messages can appear in two locations.

As runtime errors are encountered they will be displayed in the compiler info field on the File Manager form.

Figure 5.8
Runtime Errors



These are errors that the compiler can't catch. Yet will keep your program from running properly when programmed into the chip. For instance trying to access IOport 99 will generate a runtime error but not a compiler error.

General output is presented in the debug terminal. Display commands like print and debug will display data just like it does when the chip sends debug info to the terminal.

Figure 5.9
Debug Output



Many commands create output for external devices. The debug window is used to display this data. This data is prefaced with a [command:port] with its associated data following.

Figure 5.10
Command Output



Address Packet Simulation

One of the most powerful features of the simulator is the ability to help you develop CoProc's. To fully simulate a CoProc we have added the AP field to the debug terminal.

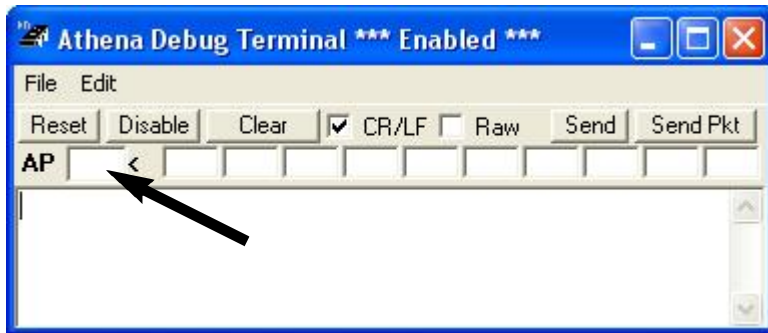


Figure 5.11
AP Field

When populated the the value will be transmited to the simulator in 9bit address mode. The remaining fields will sent normally.

This will allow you to use the `getpacket`, `debugin`, `getvpacket`, and `cpslave` commands without modification for true AP packet and RS485 simulation.

The AP field is only used in simulation mode and only when the `Send` and `Send Pkt` buttons are hit.

Note: The data fields will only be visable when in advanced dificulty mode.

Simulation Exceptions

The simulator will work with every command for all the Athena class microcontrollers. IRQ's, Timers, and many internal registers are supported as well.

The following list of commands will document any exceptions to normal operation of the command.

Ignored Commands

These commands are executed but ignored as they have no affect on the program during simulation.

- debugbaud
- pullupon
- pullupoff
- setbaud
- systemclock

General Commands

These commands operate as expected by the simulator

apinit	cpslaveinit	input	pauseus
apoffline	cptxmt	lcdwrite	portbitget
aponline	cptxoff	lcdcontrol	portbitset
aptx	data	lcdchar	portget
atodinit	dim	lcdinit	portset
arrayget	debug	lcdcls	print
arrayset	debugin	longpause	pulseout
bintodec	eeread	lookdown	pulseoutms
bintohehex	eewrite	lookup	random
bitreset	end	low	return
bitset	for/next/step	nop	setio
bitvalue	getpacket	onportgosub	sleep
branch	getvpacket	onportgoto	toggle
clearall	gosub	output	waitport
configio	goto	p2irq	watchdog
const	high	p7irq	
cpslave	if/then/else	pause	

Display Only Commands

These commands are executed and ignored but display data in the debug window. Note that IO port ranges are still checked.

hwpwm	servo
i2cout	shiftout
i2cout2	signal
irout	srin
owtx	srinport
serout	TW523write

Stimuli Input Commands

These commands read the input from the stimuli field.

atod	owreset	rccount
l2cin	owrx	serin
l2cin2	pot	srin
irin	pulsein	srinport
miniad	shiftin	TW523read

Timer Commands

The Simulator has a built-in timer that can be setup just like the timer on the chips. However it increments 1300 units every 1ms. This should not affect the actual operation but timed results may vary.

Athena Language Tutorial

This tutorial will cover the basic command set. These are the commands that are presented in the command wizard when the basic mode is selected.

You are encouraged to try each of the code examples presented.

Experiment by making modifications and group with other commands.

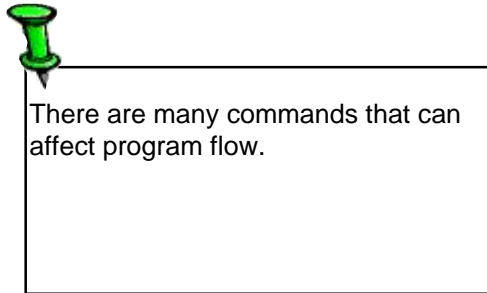
This tutorial will work with all chips. If you are using a chip other than the Athena make sure you use the correct chip directive.

You are encouraged to use the simulator to run through the tutorial.

Athena Program Model

The Athena Uses a flat program model similar to that of many popular microcontrollers on the market today. This means that the program flows from top to bottom in a linear motion.

Some commands can affect this program flow. Commands like goto or gosub can jump to predefined locations in your program.



Example 4.1

```
print "Now"
print "is"
print "the"
print "time"
```

Program starts here. (arrow pointing to the first line)

Ends here. (arrow pointing to the last line)

Display Commands

Before moving forward lets look at the display commands. These commands are used to send text, characters and control codes to various devices, including the debug terminal.

- debug** - Display to debug terminal
- print** - Display to debug terminal with some preprocessing.
- serout** - Display to IO port via Asynchronous output.
- lcdwrite** - Display to LCD. (Athena and AthenaHS only)

These commands will take multiple pieces of data separated by commas and output them to the corresponding device.

Data types that may be output:

Variables

Use the name of a variable you have defined.

Registers

Use the name of the internal Athana register.

IO Port States

Use inpx where x is the IO port to display. This will display 1 or 0.

Constants

Use the name of the constant you have defined.

Raw Number Values

Just a number.

Quoted Strings

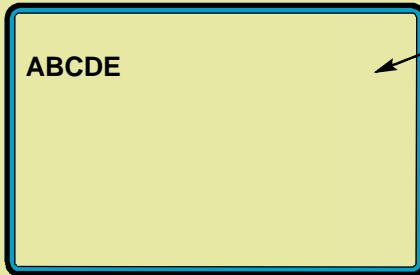
Any data in quotes will be sent one character at a time.



When you output a piece of data its raw ascii value will be output. For instance if you send the value 65 to the debug terminal it will display an A on the terminal.

Example 4.2

debug 65,66,67,68,69

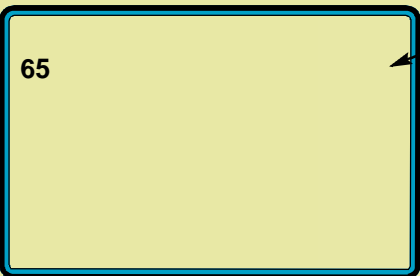


Will display this.

If you want to display the actual values use the dec modifier.

Example 4.3

debug dec 65



Will display this.



There are 3 modifiers.

dec

Converts to decimal for display.

hex

Converts to hexadecimal for display.

control

Used to send LCD control codes.
(Athena and AthenaHS only)

The display commands will not add a newline by default. If you want to add a new line place a 13,10 at the end of the data stream.

The **print** command sends the output to the debug terminal just like the debug command. There are a few subtle differences however.

- The **print** command always converts values to decimal first unless the hex option is used.
- The **print** command always adds a CR/LF (newline) to the end of the stream.

Program Flow

The program flows from top to bottom executing commands as they are encountered. We call this program flow. One way to change the program flow is using the **goto** command. In order to use the **goto** command you must be able to tell the program where to go. We do this with a **label**.

Labels

A **label** is a place holder for a location in your program code. To declare **label** just use a name followed by a colon. The name is alphanumeric and must beg with a letter. The names are case sensitive, so the **label** "loop:" is different than the **label** "Loop:".

Example 4.4 shows a simple loop to demonstrate both the **goto** and **label** commands.



You can define up to 500 labels. They do not take any program memory in your code.

Example 4.4

```
loop:  
  print "Hello World"  
  goto loop
```



This is the program that never ends.

Remember labels are just place holders. They represent a location in your code.

The program will print the words **Hello World** over and over again to the debug window. It will do this forever.

Spaghetti Code

Lets look at a common programming error. Example 4.5 shows what we call spaghetti code. It jumps all over the place and is hard to follow. Unfortunately the use of extensive goto commands can lead to some pretty bad spaghetti code.

Example 4.5

```
goto Step1  
  
Step2:  
  print "In step 2"  
  goto done  
  
Step1:  
  print "In step 1"  
  goto step2  
  
Done:
```



Im so confused!!!

Gosub Command

One way to eliminate spaghetti code is to create small subroutines within your program. You can do this with the **gosub** command. The **gosub** command works identical to the **goto** command with one exception. When a **gosub** command is executed it saves its location so that the program may return where it left off.

Return Command

The return command is used to tell the program to return back to where the **gosub** command was issued.

Example 4.6

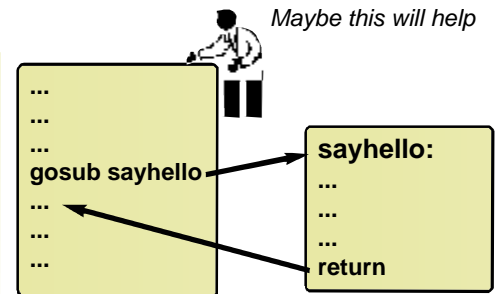
```

loop:
  gosub sayhello
  goto loop

sayhello:
  print "Hello"
  return
  
```

When the *gosub* is reached we jump to the *sayhello* label.

When we reach the *return* we go back to the point just after the *gosub* command.



IO Ports

The IO ports are how we control the outside world.

The Athena IO ports can be set up for input or output. We sometimes refer to this as the direction. To change the direction of the IO ports use the **input** and **output** commands.



At powerup, the Athena IO ports default to input mode.

Output Mode

In Example 4.7 we set IO port 5 to output mode. Once a port is in output mode you can change its state by using the **high**, **low** and **toggle** commands. When a port is in the high state it will measure pretty close to 5v (or Vdd). When in the low state it will measure pretty close to 0v or (Vss).

Example 4.7

```
output 5
```

Take a look at Example 4.8. In this example we change port 5 from high to low very fast. It will generate a 5Khz signal on port 5.



Athena IO ports really have three states.

- High
- Low
- High impedance

When an IOport is in input mode it asserts no level on the port. This feature will allow us to do some advanced operations such as I2c communications without external hardware.

Example 4.8

```
output 5
```

```
loop:
  high 5
  low 5
  goto loop
```



You can only change a port's state with the high and low commands when it is on output mode.

Input Mode

Now lets take a look at accessing input ports. In Example 4.9 we set the port to input and access it via the **inp** command. Here the **print** command will print 0 or 1 depending on the state of the port.



A great many of the built-in commands access and change the IO ports.

Example 4.9

```
input 5
```

```
loop:
  print inp5
  goto loop
```



*We set the port to input mode with the **input** command.*



The inp command can be used in many ways.

```
inpX
inp.X
inp.Const
```

Where X is a port number and Const is a constant indicating a port number.

The Athena IO ports will register a low if the voltage level is below 1.2 volts. The IO port will register high if over 1.35 volts. Between these voltage values is a grey area that could make the port swing either way.

Variables

If all you could do was jump from place to place in the code you would not get much done. We want to be able to manipulate some data. Data space is different than program space and is normally accessed by using variables.

Dim statement

To declare (create) a variable, use the **dim** statement as shown in Example 4.10.

Example 4.10

```
dim age
```

To put a variable to work we need to store information in that variable and retrieve information from it. One way to store information in a variable is to use the assignment operator. The value on the right side is stored in the variable on the left side. In Example 4.11 the variable age will contain the value of 25.

Example 4.11

```
dim age
age = 25
print age
```

The = operator can be used to assign a value to a variable.

The print statement is very versatile. It can print variables directly.



All Athena variables are unsigned 8-bit integers they can store a value of 0-255 They cannot contain signed numbers or decimal points.

Constants

Constants allow you to assign a name to a number. Unlike variables these can not be changed once they are set.

Use the **const** command to define a constant as shown in Example 4.12

Example 4.12

```
const togport 6
output togport

loop:
toggle togport
goto loop
```

This is where we define the constant.



You can define up to 500 constants. Constant definitions do not use program memory.

Example 4.13

```

dim value
value = 2 * 7
print value

```

Math

Math on the Athena works as you may expect with just a few exceptions. As shown in Example 4.13, we simply place the result variable on the left side of the equal sign and the math expression on the right side. In this case $2 * 7$ will be evaluated to 14 and placed in the variable called value.

The Athena calculates expression in the order that they are encountered. This is done for speed and simplicity and actually works quite well.

In Example 4.14 the math calculation will work like this: 2 and 1 will be added together. 7 will then be multiplied by the result.

Example 4.14

```

dim value
value = 2 + 1 * 7
print value

```



2 is placed in result.

1 is added to result.

Result is multiplied by 7.

Result is placed in variable.

You can also use other variables in your math expressions. In Example 4.15 we first assign values to both y and z variables.. Next we take the two variables and use them in an expression assigning them to the x variable.

Example 4.15

```

dim x,y,z
y = 10
z = 5
x = y / z
print x

```



You can place multiple variables on the same line.



10 is placed in y.
5 is placed in z.

y (10) is divided by z (5)
The result is placed in x.

A few notes on integer math. In integer math the values will contain whole numbers only. If a result contains a decimal number it is simply removed. The result in Example 4.16 will be 2 not 2.5. There is no rounding in integer math.

Example 4.16

```
dim x
x = 10 / 4
print x
```

Expressions

There are two types of expressions used with the Athena. We will refer to them as full expressions and vcn expressions. In the command syntax definitions you will see **exp** and **vcn** as a reference to full expressions and vcn expressions.

Full Expressions

Full expressions may contain mathematical calculations including input port definitions.

Only certain commands support full expressions.

print
debug
serout
lcdwrite
if/then/else
variable assignments

Example 4.17

```
print 4 * 5
```

```
print 1+2+3+4+5
```



*The expression is evaluated,
then the result is printed*

```
20
15
```

vcn Expressions

This is a much simpler expression. vcn expressions may not contain mathematical calculations or input port definitions.


VCN expressions may contain one of the following:

- variable
- constant
- number

Most of the Athena commands use VCNexpressions.

Example 4.18 *These are vcn expressions.*

```
output 1 ←
loop:
toggle 1 ←
goto loop
```



For / Next Commands

While you can use the **goto** command to create loops, the **for/next** command allows you to create some very efficient and specific loops. In order to use a **for** loop you must use an integer variable to act as an index while the **for/next** command is executed. Let's take a closer look at the syntax used with the **for** command.

```
for AAA = BBB to CCC
```

- AAA is the variable to used as the counting index
- BBB is the starting vcn expression
- CCC is the ending vcn expression

Example 4.19 will count from 1 to 10. As long as the counting values are within the range of 1 to 10 then commands between the for and **next** commands will be executed.




The **for/next** command is always checked before the actual loop is executed. This means if the index is out of range it will not execute even on the first pass.

Example 4.19

```
dim x

for x = 1 to 10
  print x
next
```



This program will loop 10 times. Each time the x variable will be incremented by 1.

An optional step argument that can be supplied when using **for/next** command. This is the **step** argument. This allows you to specify how much you want to increment the index during each iteration. In Example 4.20 the **for/next** loop will count from 2 to 10 by 2's.

Example 4.20

```
dim x

for x = 2 to 10 step 2
  print x
next
```



*This program will count by 2's because of the **step** command.*



Step must be a number.

You can modify the index variable freely as the for command iterates. Once the end count is reached either by counting or by user intervention, the loop will exit once the next command is reached.

In Example 4.21 we show how to manipulate the index.

Example 4.21

```
dim x
for x = 1 to 5
  print x
  x = 10
  print x
next
```



*The **for/next** loop will execute 1 loop. Notice that the index variable will contain the correct index until you change it.*

To count backwards you must use the **step** argument with a negative number as shown in example 4.22.

Example 4.22

```
dim x
for x = 10 to 1 step -1
  print x
next
```



*Any time you want to count backwards you must use the **step** argument. Supply it with a negative interval.*

You can also use a **goto** command to exit a **for/next** loop. A **gosub** command may be used to temporarily exit a **for/next** loop. Once the **return** is encountered it will re-enter the loop where it left off.

If / Then / Else Commands

The **if/then** command is where real programming can be done.

By combining them with the other looping commands and math expression some pretty advanced constructs can be created.

The expressions can be just about any variable/math combination. The condition can be any of the following

- = Equal
- < Less than
- > Greater than
- <> Not Equal to
- >= Greater than Equal to
- <= Less than Equal to

In Example 4.23 the state of port 0 is compared with the value of 1. Since we are using an equal sign as the condition operator, the condition will evaluate as true only when the state of port 0 is high.

To take it one step further we will add the **else** option.

In Example 4.24, by adding the **else** command we give the if statement a

Example 4.23

```
loop:
  if inp0 = 1 then
    print "Port high"
    goto loop
  endif
```



*If condition is true then everything following the **then** command is executed.*

place to go if the test condition is false.



The Athena **does not** allow multiple condition tests using the **and** and **or** commands.

Example 4.24

```
loop:
  if inp0 = 1 then
    print "Port high"
  else
    print "Port low"
  endif

  goto loop
```

If the port is high we will print "Port high"



If the port is low we will print "Port low"

Delay Commands


The Athena has 3 commands used to create various delays.

pauseus

The pauseus command allows you to delay for very short durations. The unit of resolution for this command is actually 3us. The command itself has a 50 us overhead. You can pass a value of 1-255 which will yield a delay of 53-815 us.

Example 4.25

```
output 0
loop:
high 0
pauseus 100
low 0
pauseus 100
goto loop
```



*This command will cause a 350us delay
overhead = 50
value = 100 = 300us
Total = 350us*

If you were to place an oscilloscope on port 0 the actual delay would be over 400us. Why? Each command has its own delay. The high and low commands have a 77us overhead. The goto has a 45us overhead.

pause


The **pause** command is used to create a longer delay than the **pauseus** command. The resolution is in 1ms units. The value passed is in the range of 1-255. This will yield you a 1-255ms delay. This command has a 137us overhead.

longpause

For longer delays you could use multiple pause commands as shown in example 4.26

Example 4.26

```
pause 250
pause 250
pause 250
pause 250
```



These 4 commands will cause a 1 second delay.



The AthenaHS runs 5 times faster than the Athena and therefore the overhead for the commands is 5 times less.

The actual delay times for the pauseus, pause and longpause are the same.

This can be a bit awkward and eats too much program memory. For the longer delays you can use the **longpause** command. This command lets you make multiple pause commands with the overhead of only 1 byte.

Example 4.27

longpause 250,4



This will cause a 1 second delay.

It will pause for 250ms 4 times.

The Code in Example 4.27 will do the same thing as the code in Example 4.26.

Branch Command

There are times when you may want to jump to multiple locations based on the value of a variable. Certainly you can use multiple if/then and goto commands as shown in Example 4.28

Example 4.28

```
if x = 0 then
  goto do0
endif
```

```
if x = 1 then
  goto do1
endif
```

```
do0:
  print "Its 0"
end
```

```
do1:
  print "its 1"
end
```

This can be a bit bulky and uses a lot of memory. The branch command is much better suited for this type of operation. The branch command can also be used with the lookup command to create a very efficient method of branching to various portions of code.

Example 4.29 shows the same operation as in Example 4.28 but the code is much more efficient.

Example 4.29

```
branch x,do0,do1
```



This does the same as Example 4.28

```
do0:
  print "Its 0"
  end
```

```
do1:
  print "its 1"
  end
```

end

The end command simply stops program execution in its tracks. The command can be placed anywhere. Once encountered the Athena will no longer process commands.

Accessing Registers

The Athena Language has several software and hardware registers that effect how the Athena operates.

These registers are all 8-bit and are accessed just like variables. You access them by name. All registers name are capitalized.

Example 4.30

```
PULSEINSCALE=21
```

```
print VERSION
```

Appendix A lists the important Athena registers.



In Example 4.29 if x is not 0 or 1 it will fall through. So in this case it will display "Its 0"

You can place a catch all print statement after the branch command as shown here.

```
branch x,do0,do1
```

```
print "Over 1"
end
```

```
do0:
  print "Its 0"
  end
```

```
do1:
  print "its 1"
  end
```

APFLAGS

This register contains the current flags used by the address protocol. (Athena485, Nemesis and Perseus only)

Bits:

- 0: Address protocol status: 0=off 1=on
- 1: RS485 control port: 0=Dont Use 1=Use
- 2: UART TX: 0=Shutdown 1=No Shutdown
- 3: RS485 Duplex: 0=Full 1=Half
- 4: Control Port always follows UART TX status

- 7: Address status. 0:Chip Not active 1:Chip Active

APID

This is the address of the chip when in address protocol mode. It is normally set by the apinit command. (Athena485, Nemesis and Perseus only)

ATODMODE

This Register allows you to set up the atod command so that it returns a 16bit value. (Nemesis and Perseus only)

The default value is 0. In this mode the atod command will return 8 bit value.

When ATODMODE is set to 128 the atod command will return a 16bit value with the results in registers ADRESH and ADRESL. The atod command also returns the high byte.

DISPLAYMASK

This Register allows you to set up how the display routines display decimal numbers. This applies to the print, debug, serout, lcdwrite commands.

mask - Turns on and off various display parameters.

bit 0

- 0: Leading Zeros
- 1: Leading Spaces

bit 1

- 0: Add leading characters
- 1: Don't add leading characters

bit 2

- 0: Display all characters
- 1: Display only 2 digits

HTIMEOUTL

HTIMEOUTH

HTIMEOUTU

These registers set the timeout set by the debugin command.

Default values are

HTIMEOUTL = 0

HTIMEOUTH = 0

HTIMEOUTU = 6

This is a count down timer so setting the total count lower will create a quicker timeout.

IDLECOUNT

When the getpacket and getvpacket commands are called and no data is received they increment this register. When this register reaches 256 it wraps to 0 and resets the index variable to 0.

You can use this register to determine if compound packets are out of sequence.

IRMODE

Sets the mode of operation for the irin and irout commands

- 0 - Standard Sony 7 bit command and 5 bit device. (default)
- 1 - Extended Kronos Robotics Mode 8 bit command and 8 bit device.

IROUTPERIOD

This register is used to set the modulation frequency. (Athena, AthenaHS, Athena485 and Nemesis only)

LOOKDOWNMODE

This register affects how the lookdown command operates.

- 0: The index returned in the resultvarb will be 0-n and the result variable will remain unchanged if searchvalue is not found.
- 1: The index returned in the resultvarb will be 1-n and the result variable will contain 0 if the searchvalue is not found.

Appendix A: Registers Cont.

P2COUNTER

This counter indicates the number of times the Port 2 IRQ fired. See the p2irq command. (Perseus only)

P7COUNTER

This counter indicates the number of times the Port 7 IRQ fired. See the p7irq command. (Athena, Athen485, AthenaHS and Nemesis)

PROCTYPE

This register will return the processor type.

Athena = 0
AthenaHS = 1
Athena485 = 3
Perseus = 4
Nemesis = 5

PULSEINTIMEOUT

When waiting for a pulse with the pulsein command this parameter will increment. When it overflows a timeout will occur. Valid values are 0-255. The higher the number the faster the timeout. Default = 32.

PULSINSCALE

This register sets the resolution of the pulsein command.

(Athena)

1: units = 4us
2: units = 8us
3: units = 16us (default)
4: units = 32us
5: units = 64us
6: units = 128us
7: units = 256us

(AthenaHS)

1: units = .8us
2: units = 1.6us
3: units = 3.2us
4: units = 6.4us
5: units = 12.8us (default)
6: units = 25.6us
7: units = 51.2us

RXHEAD

RXTAIL

These registers are used to keep track of the internal UART buffer. Valid values are 0-79. These registers will wrap.

SERTIMEOUTL

SERTIMEOUTH

These registers set the software serial command timeout.

They are simply a counter. The serial command will start looking for the start bit. It will decrement this counter while it checks. When the counter reaches zero it times out and jumps to your label.

SIGNALRES

This register sets the resolution of the signal command.

Bits 0-3 allow you to change the period resolution.

xxxx0000 - each unit = 6.05us with an overhead of 82us.
xxxx0001 - each unit = 12.05us with overhead of 103us.
xxxx0011 - each unit = 18.15us with overhead of 123us. (default)
xxxx0111 - each unit = 24.2us with overhead of 143us.
xxxx1111 - each unit = 30.35us with overhead of 162us.

Bits 4-7 allow you to change the cycles resolution

0000xxxx - Cycle units 1:1. Will cycle 1 x the amount given.
0001xxxx - Cycle units 2:1. Will cycle 2 x the amount given.
0011xxxx - Cycle units 4:1. Will cycle 4 x the amount given.
0111xxxx - Cycle units 8:1. Will cycle 8 x the amount given. (default)
1111xxxx - Cycle units 16:1 Will cycle 16 x the amount given.

STACK

This is a stack pointer to the gosub stack.

VERSION

This register contains the current version of the software

VPACKETMAXBYTE

This register will contain the number of bytes received by the getvpacket command.

(Athena485, Nemesis and Perseus only)

Blank Page

Appendix B: Chip Specifications

Athena / Athena485

Power Supply Requirements	.4v - 5.5v
Normal Operation No Load	.800ua
Sleep Mode	.Less than 1ua
Max Load on IO Port	.25ma
Max Load on All IO Ports	.200ma
Operating Temperature	-.40 to +85 centigrade
Clock Speed	.4Mhz
Program Memory	.256 bytes
Total Ram	.64 Bytes
Timers	.1
Command Speed	.15,000 per second
PC Connection Speed	.9600 Baud
IO Ports	.Up to 15 input or output
Software Baud Range	.1200 - 9600
UART Baud Range	.9600 - 19200
Program Writes	.1,000,000 Times

AthenaHS / NemesisHS

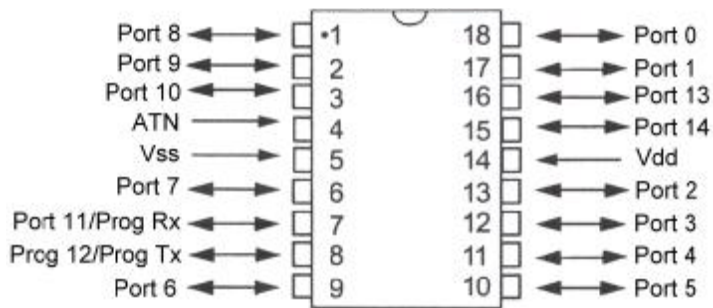
Power Supply Requirements	.4v - 5.5v
Normal Operation No Load	.3.7ma
Sleep Mode	.Less then 5ua
Max Load on IO Port	.25ma
Max Load on All IOPorts	.200ma
Operating Temperature	-.40 to +85 centigrade
Clock Speed	.20Mhz
Program Memory	.256 bytes, (2048 Bytes NemesisHS)
Total Ram	.64 Bytes, (160 NemesisHS)
Timers	.1, (2 NemesisHS)
Command Speed	.75,000 per second (Up to 2,000,000 for NemesisHS)
PC Connection Speed	.9600 Baud (NemesisHS Programs at 57600)
IO Ports	.Up to 13 input or output
Software Baud Range	.1200 - 57600
UART Baud Range	.9600 - 1250000
Program Writes	.1,000,000 Times, (100,000 for NemesisHS)

Perseus / Nemesis

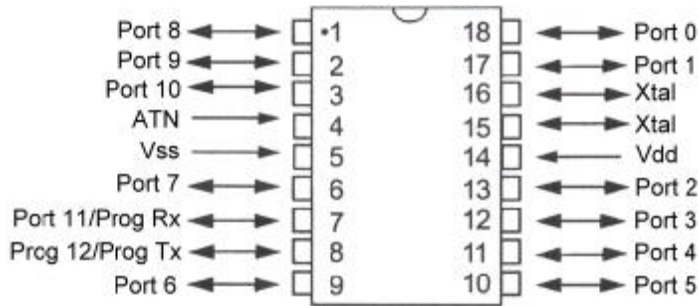
Power Supply Requirements	.4v - 5.5v
Normal Operation No Load	.28ua - 1.34ma
Sleep Mode	.Less then .2ua
Max Load on IO Port	.25ma
Max Load on All IOPorts	.200ma
Operating Temperature	-.40 to +85 centigrade
Clock Speed	.32Khz - 8Mhz
Program Memory	.256 bytes, (2048 Bytes Nemesis)
Total Ram	.64 Bytes, (160 Nemesis)
Timers	.2
Command Speed	.30,000 per second, (upto 800,000 for Nemesis)
PC Connection Speed	.9600 Baud, (Nemesis Programs at 38400)
IO Ports	.Up to 11 input or output, (15 for Nemesis)
Software Baud Range	.1200 - 19200
UART Baud Range	.1200 - 1250000, (1200 - 57600 for Nemesis)
Program Writes	.1,000,000 Times, (100,000 for Nemesis)

Appendix C: Athena / Nemesis / Perseus Chip Diagrams

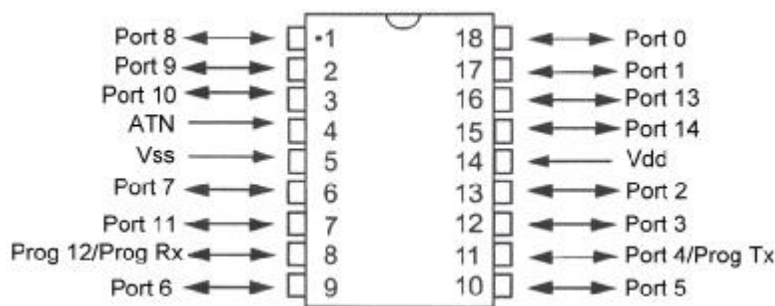
Athena / Athena485



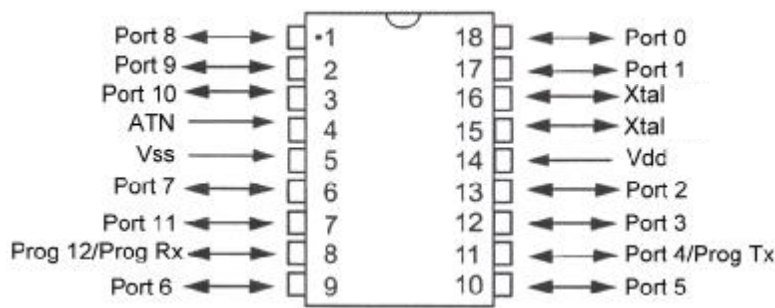
AthenaHS



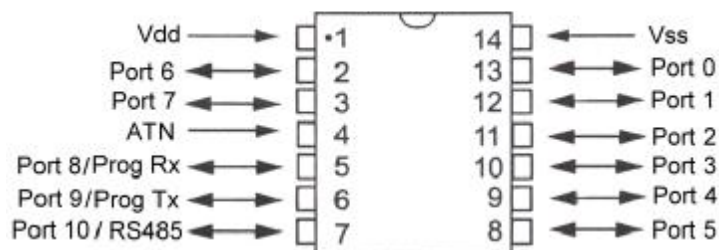
Nemesis



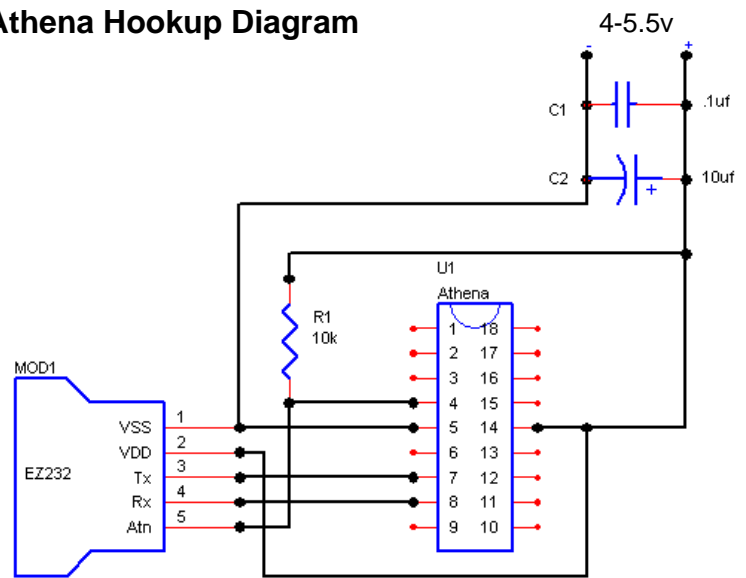
NemesisHS



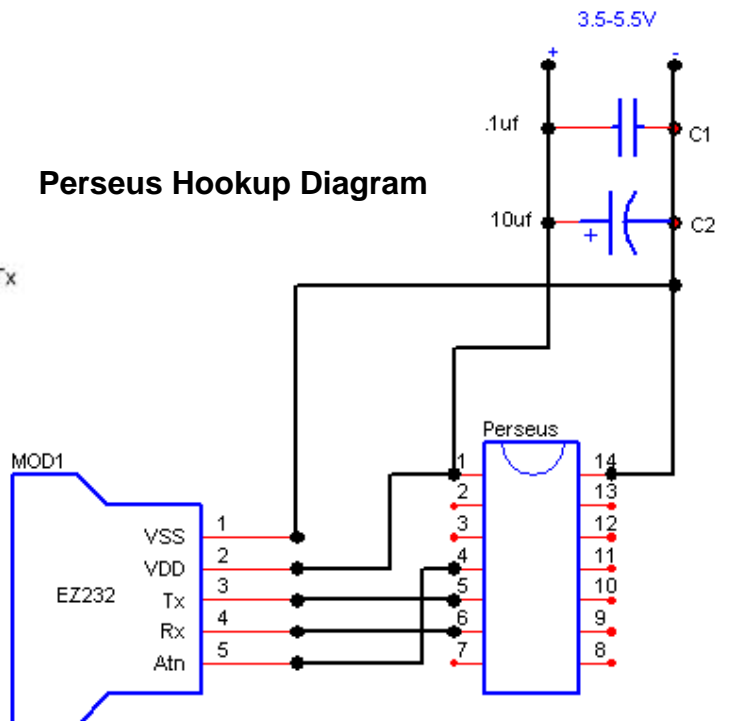
Perseus



Athena Hookup Diagram



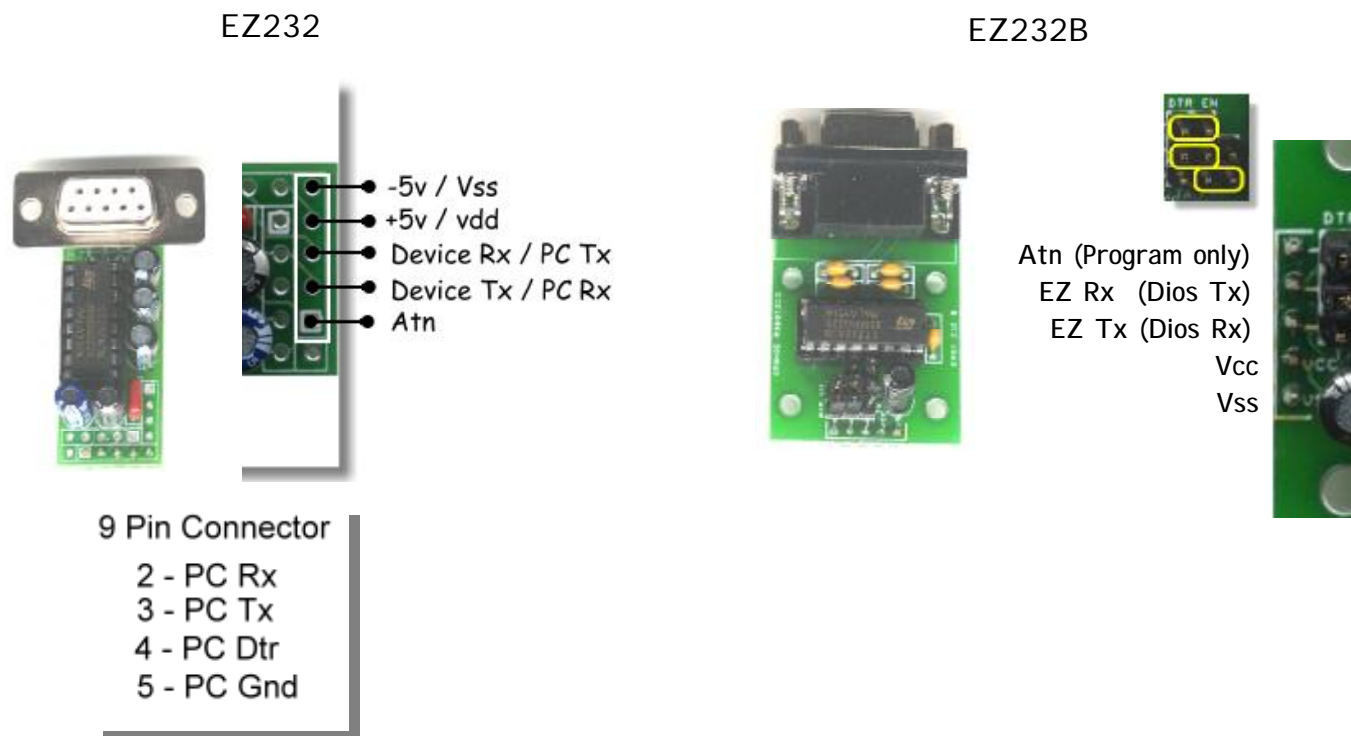
Perseus Hookup Diagram



Appendix D: EZ232 Connection

Connecting the Athena to the PC

The Athena is connected to the PC with an RS232 driver. There are 2 RS232 Drivers available from Kronos Robotics. If you wish to build your own see Appendix E.



Perseus Connections

Pin 4	Atn
Pin 8	Receive
Pin 9	Transmit
Pin 1	Vdd
Pin 14	Vss

Athena Connections

Pin 4	Atn
Pin 7	Receive
Pin 8	Transmit
Pin 14	Vdd
Pin 5	Vss

Nemesis Connections

Pin 4	Atn
Pin 8	Receive
Pin 11	Transmit
Pin 14	Vdd
Pin 5	Vss

Both the EZ232 and EZ232B drivers are available on the Kronos Robotics web site. See them at:

<http://www.kronosrobotics.com/xcart/customer/product.php?productid=16167>

and

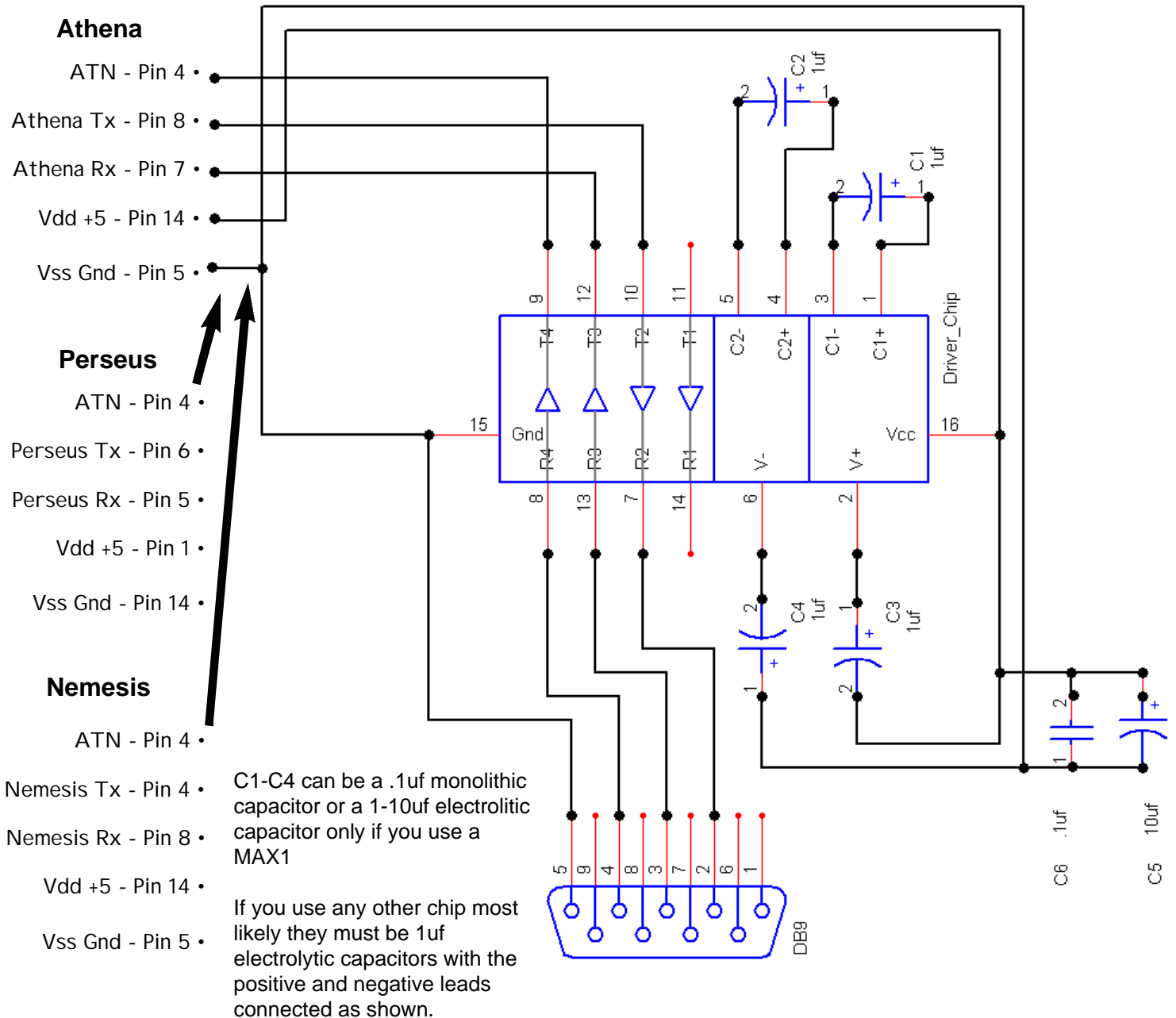
<http://www.kronosrobotics.com/xcart/customer/product.php?productid=16338>

Appendix E: Build your Own RS232 Driver

You will need a level converter chip. I recommend the Kronos Robotics MAX1 as it will work at speeds up to 115200 baud and will accept .1 uf or 1uf capacitors. The Kronos Robotics Part number is #16210

With the MAX1 you can use ether .1 - 10 uf capacitors for C1-C4.

While the Athena Class Chips can operate at voltage down to 4v the RS232 drivers can not. Most require at least 4.5v. Some can go as low as 3.5v.



Appendix F: Athena and Athena485 Breadboard Hookup

Athena BreadBoard Hookup

These instructions will work for both the Athena and Athena485 chips.

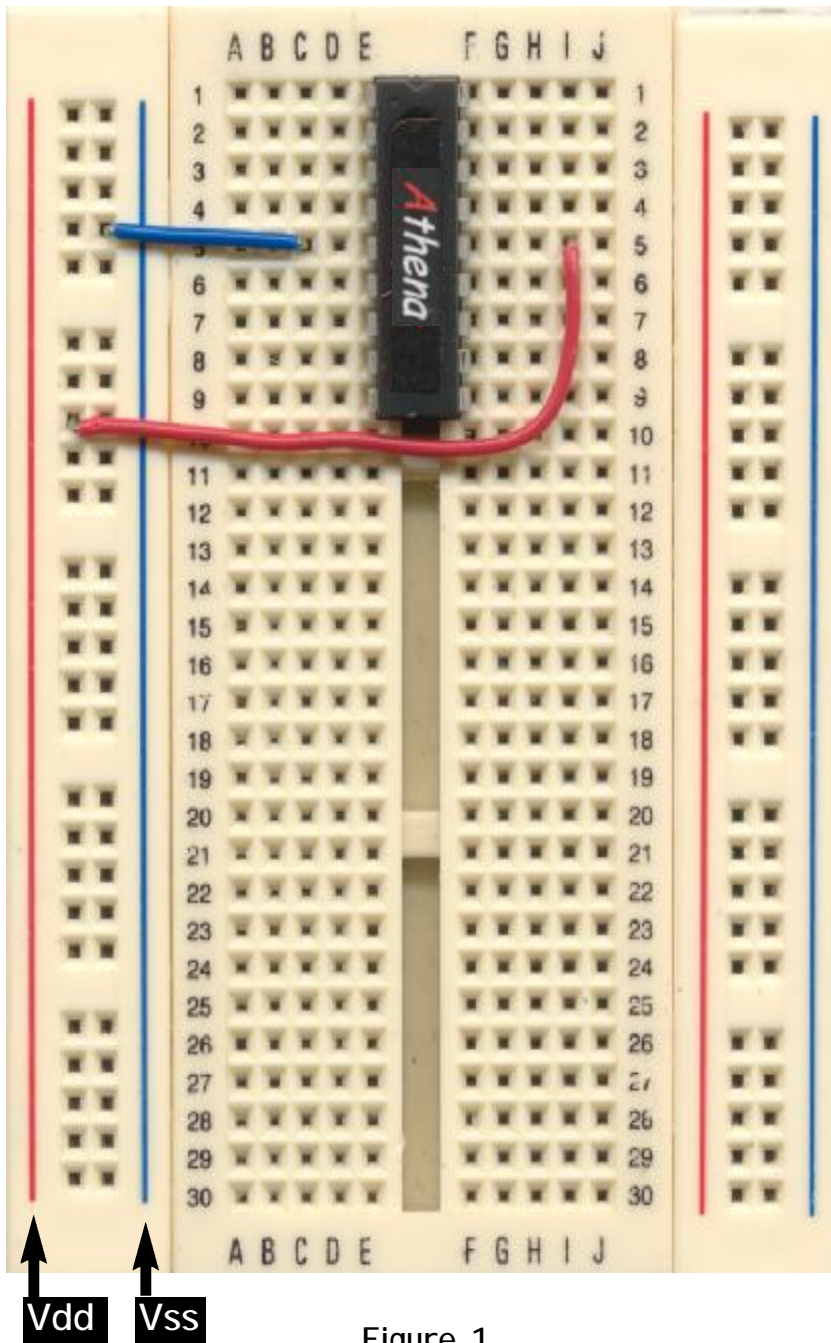


Figure 1

Step 1

The first step is to setup your power connections. Connect pin 14 to Vdd (4-5.5Vdc) and pin 5 to Vss (Gnd).

!!! Warning !!!

The small dot on the chip indicates Pin 1. Note that it is possible that the Label is not oriented as shown.

!!! Warning !!!

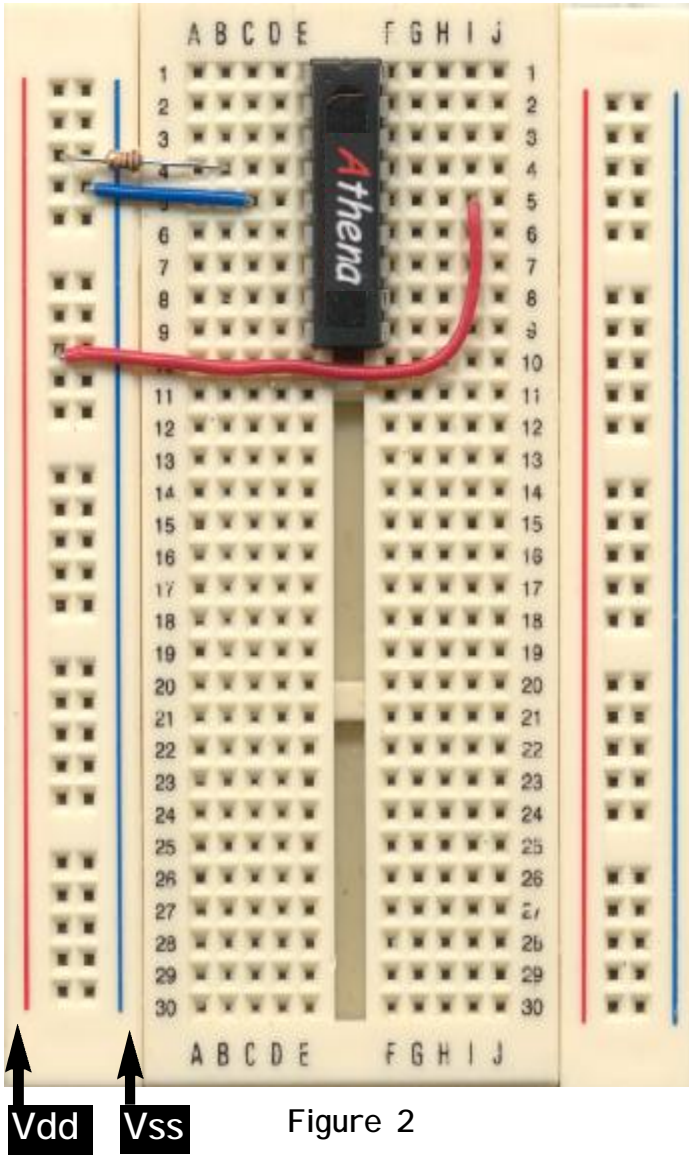


Figure 2

Step 2

Connect a 10K resistor to pin 4 and Vdd. This will hold the reset (Atn) pin high and keep the chip from resetting.

You can add a reset button by placing a button between pin 4 and Gnd.

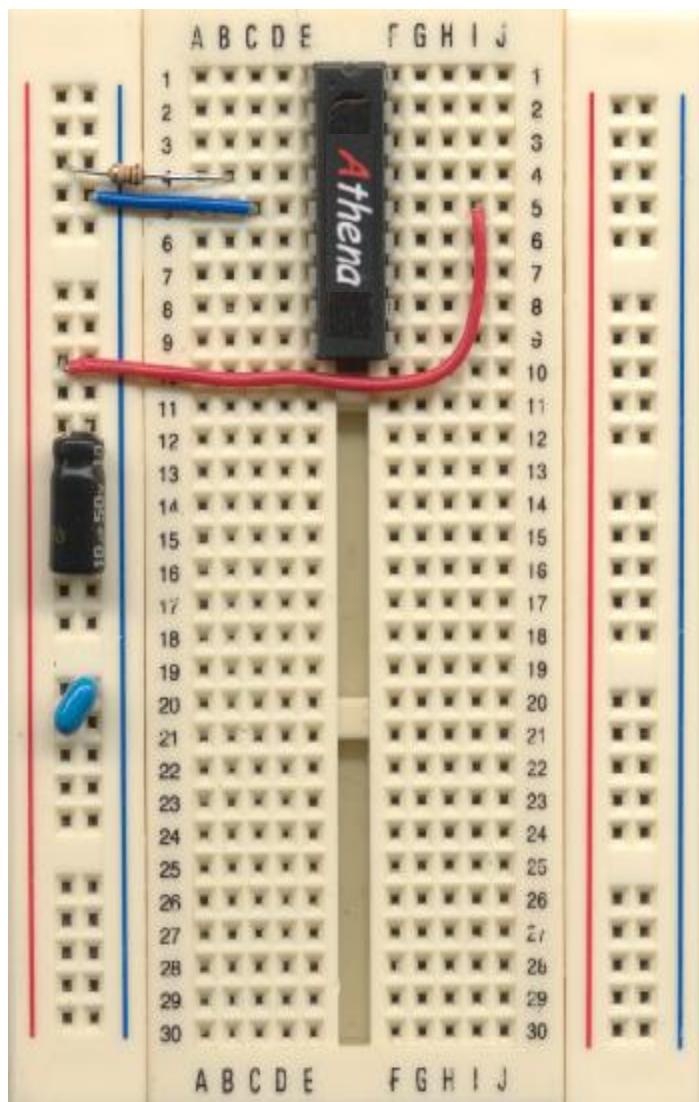


Figure 3

Step 3

Place a .1uf capacitor and a 10uf capacitor between Vdd and Vss.

Note:

The EZ232 Driver has both a 10uf and .1uf across Vdd and Vss and will provide the needed filtration in a pinch. (IE you wont need them)

However once you remove the EZ232 driver the chip could become unstable.

!!! Warning !!!

The small dot on the chip indicates Pin 1. Note that it is possible that the Label is not oriented as shown.

!!! Warning !!!

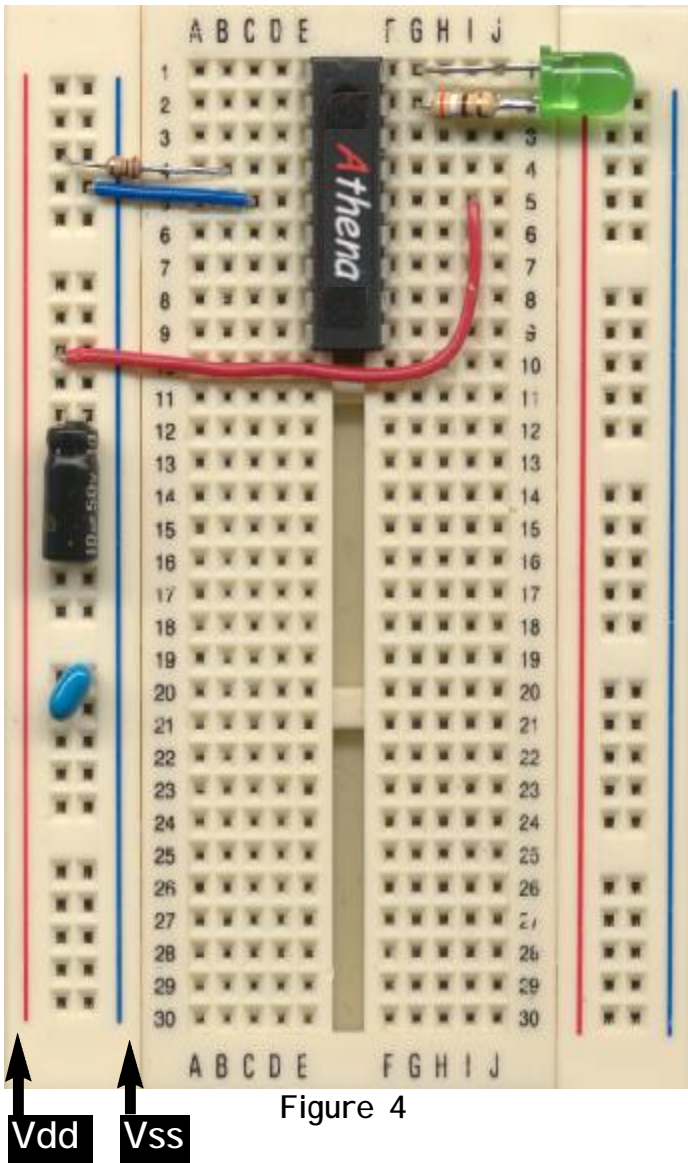


Figure 4

The Athena comes with a built-in test program. You should test your Athena for proper operation before proceeding.

Step 4

Connect an LED to pins 17 and 18 (IO ports 0 and 1). Note that the LED shown here has a 390 ohm resistor in series.

The polarity on the LED does not matter as the Athena will be switching back and forth.

Step 5

Apply 3.5-5 volts power to the breadboard bus. If you have done everything correctly the LED should blink.

!!! Warning !!!

The small dot on the chip indicates Pin 1. Note that it is possible that the Label is not oriented as shown.

!!! Warning !!!

Appendix F: Athena Breadboard Hookup

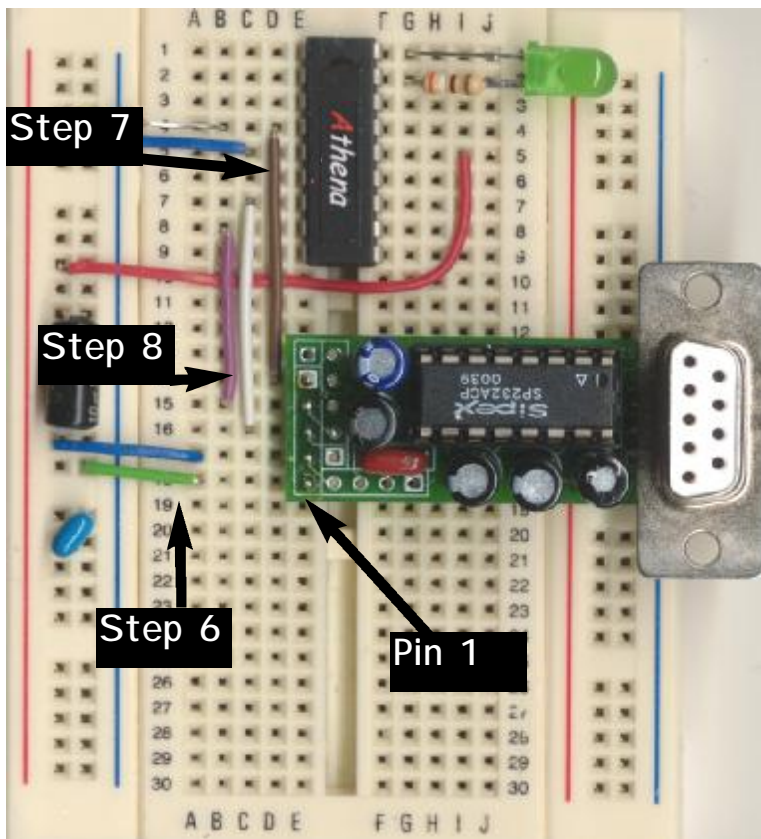


Figure 5

In order to program the Athena Chip you will need a RS232 driver. The Driver converts the RS232 (-7, +7v) signals to TTL (0-5v).

Step 6

Connect the Driver to Vss and Vdd as shown. This is pin 1 and 2 of the Easy RS232 Driver. (Note that Pin1 **is not** the pin with the square marker)

Step 7

Connect pin 5 (the one with the square marker) to pin 4 of the Athena chip.

Step 8

Connect pin 4 of the driver to pin 8 of the Athena chip.
Connect pin 3 of the driver to pin 7 of the Athena chip.

!!! Warning !!!

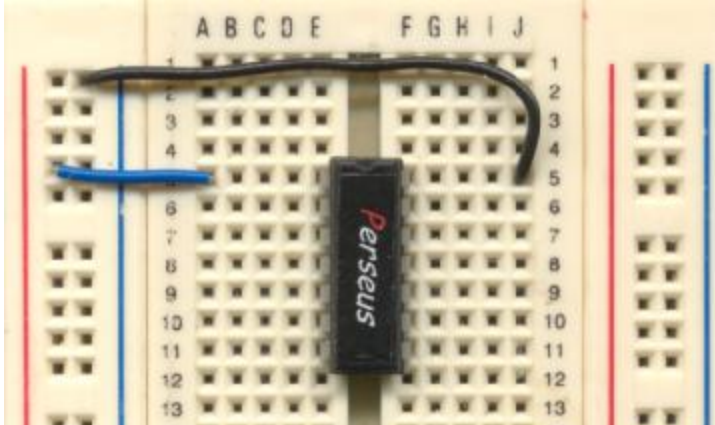
The small dot on the chip indicates Pin 1. Note that it is possible that the Label is not oriented as shown.

!!! Warning !!!

Appendix F: Perseus Breadboard Hookup

Perseus BreadBoard Hookup

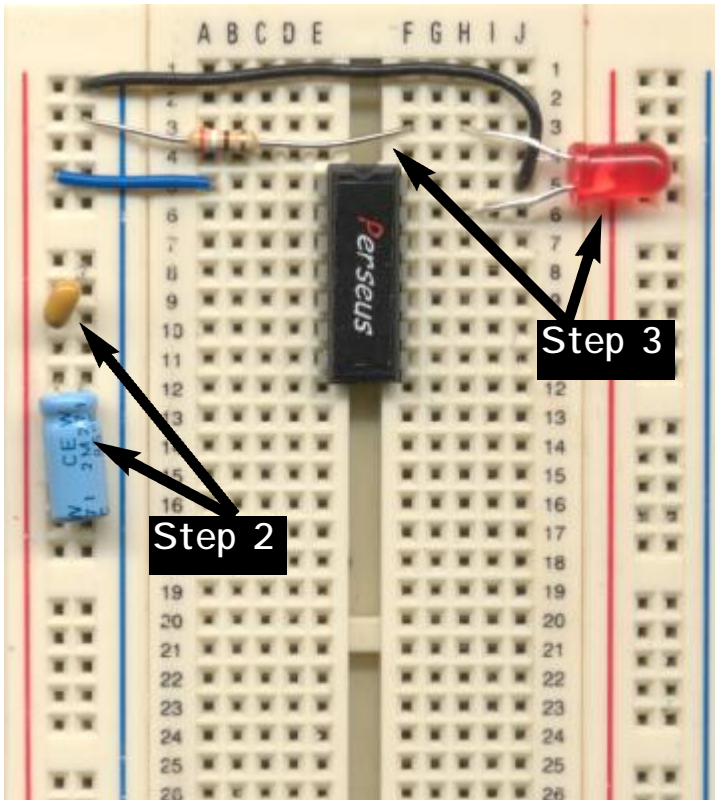
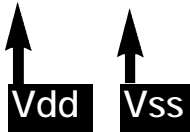
These instructions will work for the Perseus chips



Step 1

The first step is to setup your power connections. Connect pin 1 to Vdd (4-5.5Vdc) and pin 14 to Vss (Gnd).

Warning: There is a natural tendency to wire the positive voltage to pin 14. **Don't do this!!!**



Step 2

Place a .1uf capacitor and a 10uf capacitor between Vdd and Vss. Make sure the black band on the 10uf capacitor is connected to the Vss side of the power bus.

Note:
The EZ232 Driver has both a 10uf and .1uf across Vdd and Vss and will provide the needed filtration in a pinch. (IE you wont need them)

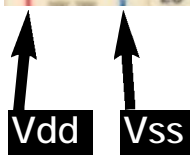
However once you remove the EZ232 driver the chip could become unstable.

Step 3

Connect the short lead on a LED to pins 13 (IO ports 0). Connect a 390 ohm resistor to the other lead of the LED and Vss.

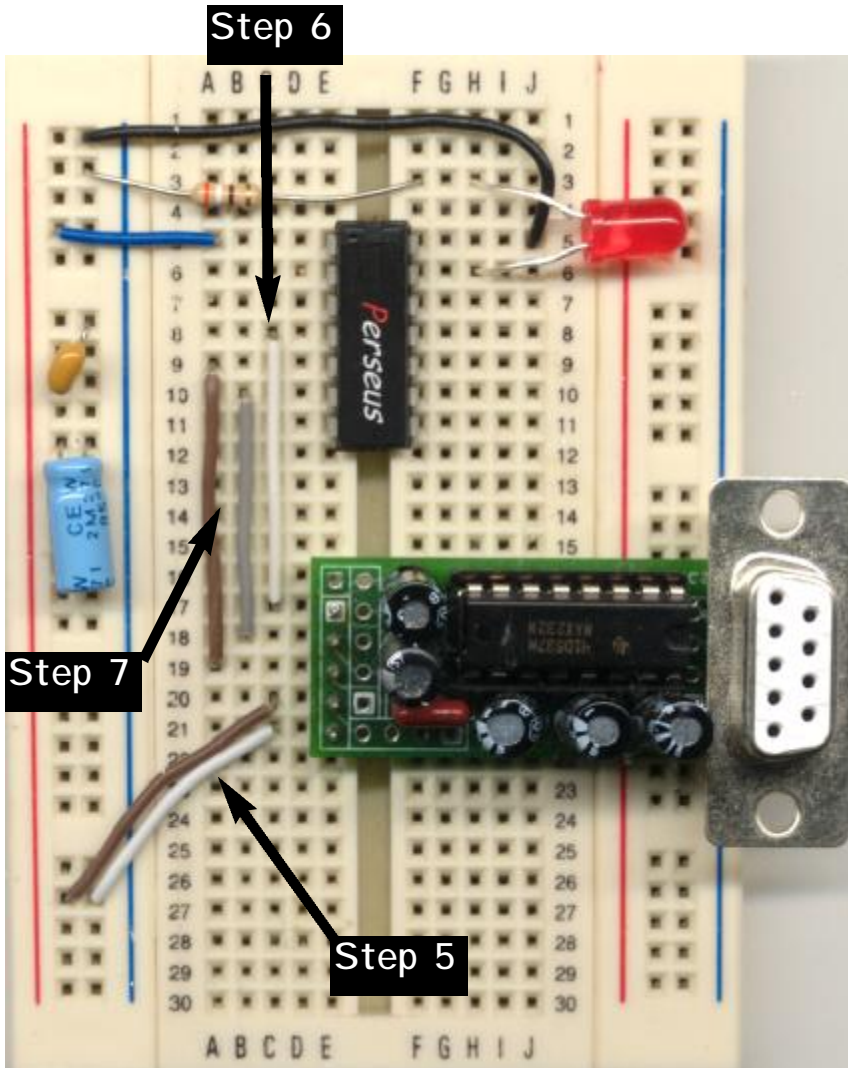
Step 4

Apply 3.5-5 volts power to the breadboard bus. If you have done everything correctly the LED should blink.



Appendix F: Perseus Breadboard Hookup

In order to program the Perseus Chip you will need a RS232 driver. The Driver converts the RS232 (-7, +7v) signals to TTL (0-5v).



Step 5

Connect the Driver to Vss and Vdd as shown. This is pin 1 and 2 of the Easy RS232 Driver. (Note that Pin1 **is not** the pin with the square marker)

Step 6

Connect pin 5 (the one with the square marker) to pin 4 of the Perseus chip.

Step 7

Connect pin 4 of the driver to pin 6 of the Perseus chip.

Connect pin 3 of the driver to pin 5 of the Perseus chip.

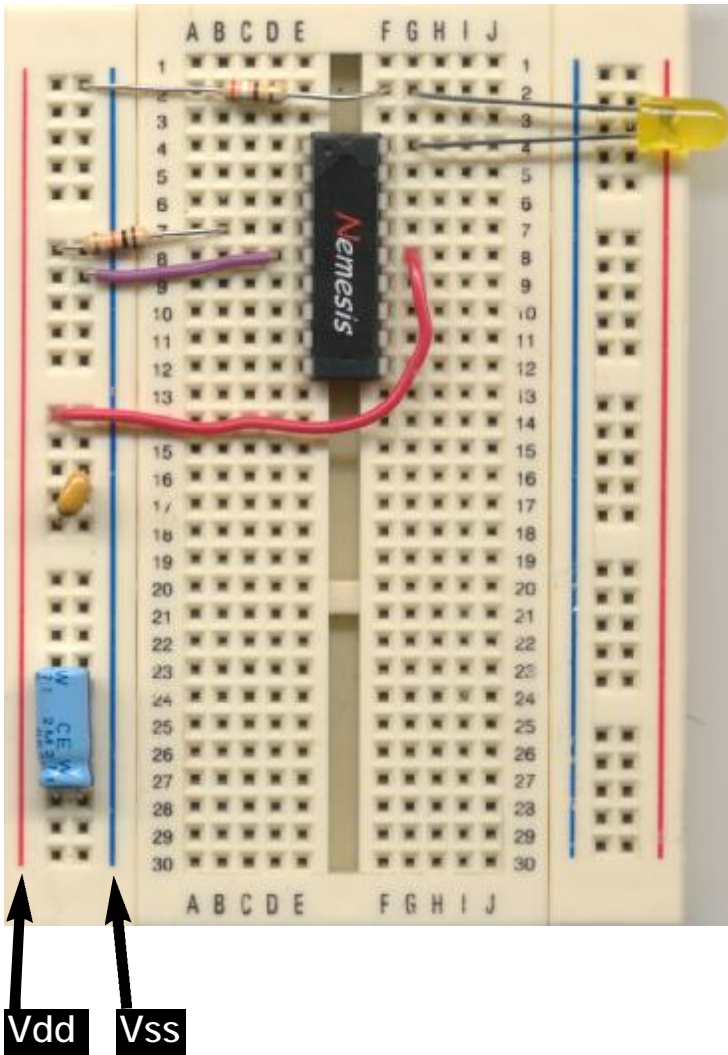
!!! Warning !!!

The small dot on the chip indicates Pin 1. Note that it is possible that the Label is not oriented as shown.

!!! Warning !!!

Nemesis BreadBoard Hookup

These instructions will work for the Nemesis chips



Step 1

The first step is to setup your power connections. Connect pin 5 to Vss (Gnd) and pin 14 to Vcc (4-5.5Vdc).

Step 2

Place a .1uf capacitor and a 10uf capacitor between Vdd and Vss. Make sure the black band on the 10uf capacitor is connected to the Vss side of the power bus.

Note:

The EZ232 Driver has both a 10uf and .1uf across Vdd and Vs and will provide the needed filtration in a pinch. (IE you wont need them)

However once you remove the EZ232 driver the chip could become unstable.

Step 3

Connect the short lead on a 390 Ohm resistor and then to Vss. Connect the other lead to port 0 (pin 18)

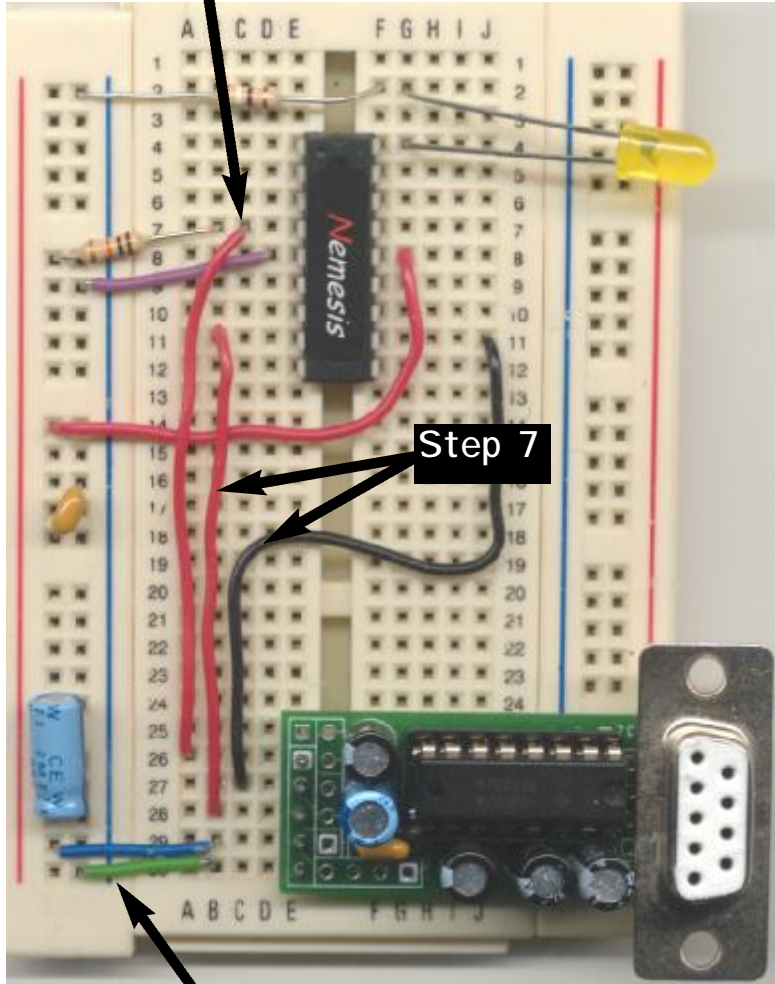
Step 4

Apply 3.5-5 volts power to the breadboard bus. If you have done everything correctly the LED should blink.

Appendix F: Nemesis Breadboard Hookup

In order to program the Nemesis Chip you will need a RS232 driver. The Driver converts the RS232 (-7, +7v) signals to TTL (0-5v).

Step 6



Step 5

Step 5

Connect the Driver to Vss and Vdd as shown. This is pin 1 and 2 of the Easy RS232 Driver. (Note that Pin1 **is not** the pin with the square marker)

Step 6

Connect pin 5 (the one with the square marker) to pin 4 of the Nemesis chip.

Step 7

Connect pin 4 of the driver to pin 11 of the Nemesis chip.

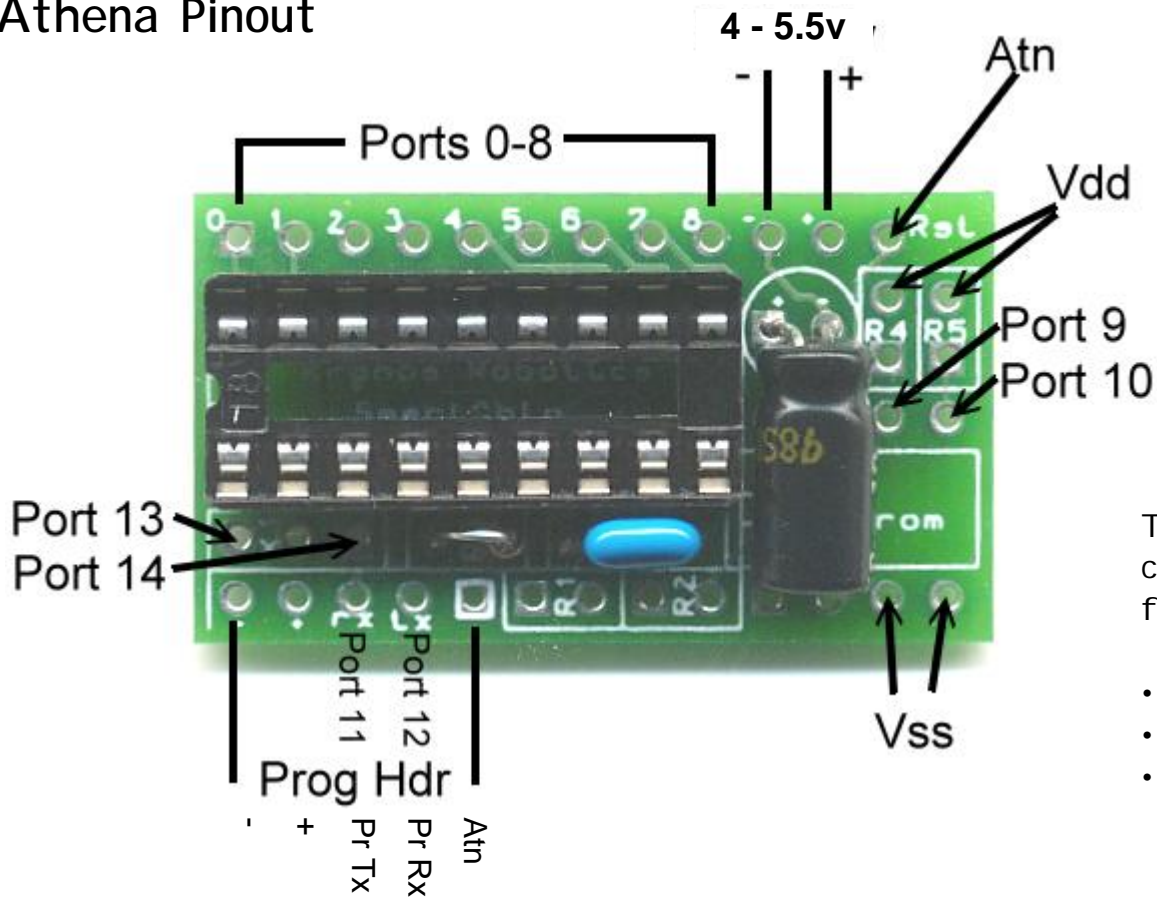
Connect pin 3 of the driver to pin 8 of the Nemesis chip.

!!! Warning !!!

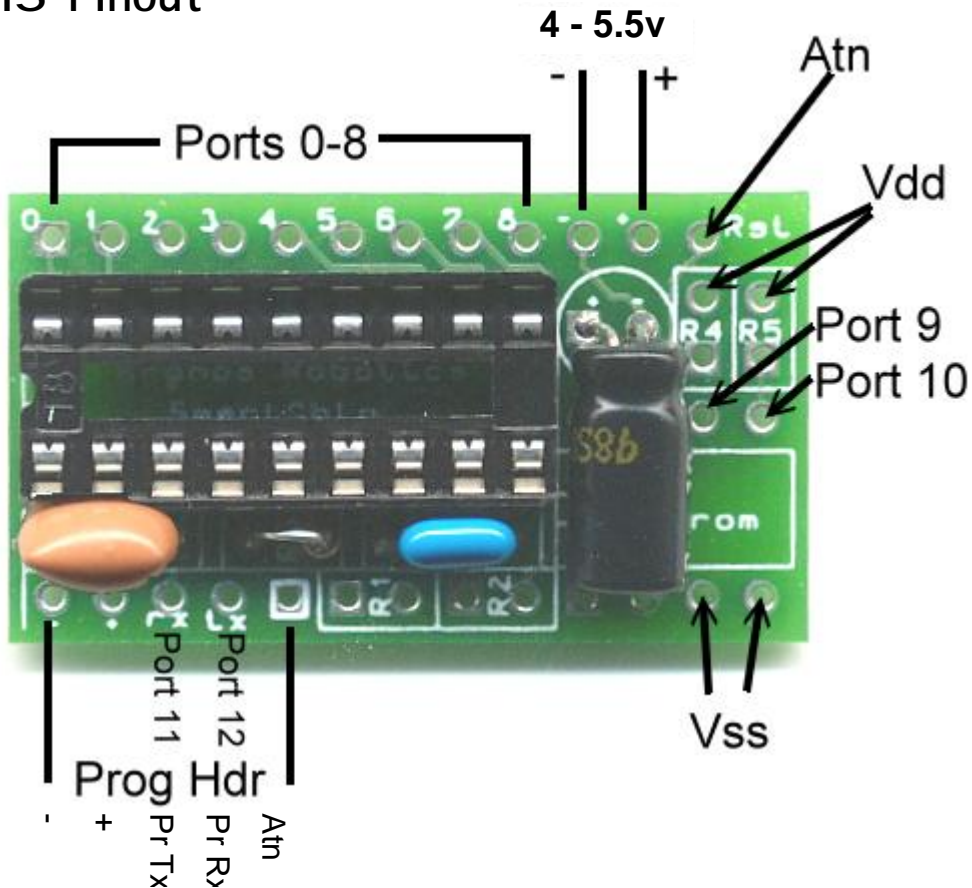
The small dot on the chip indicates Pin 1. Note that it is possible that the Label is not oriented as shown.

!!! Warning !!!

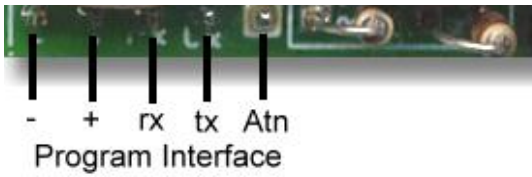
Athena Pinout



AthenaHS Pinout



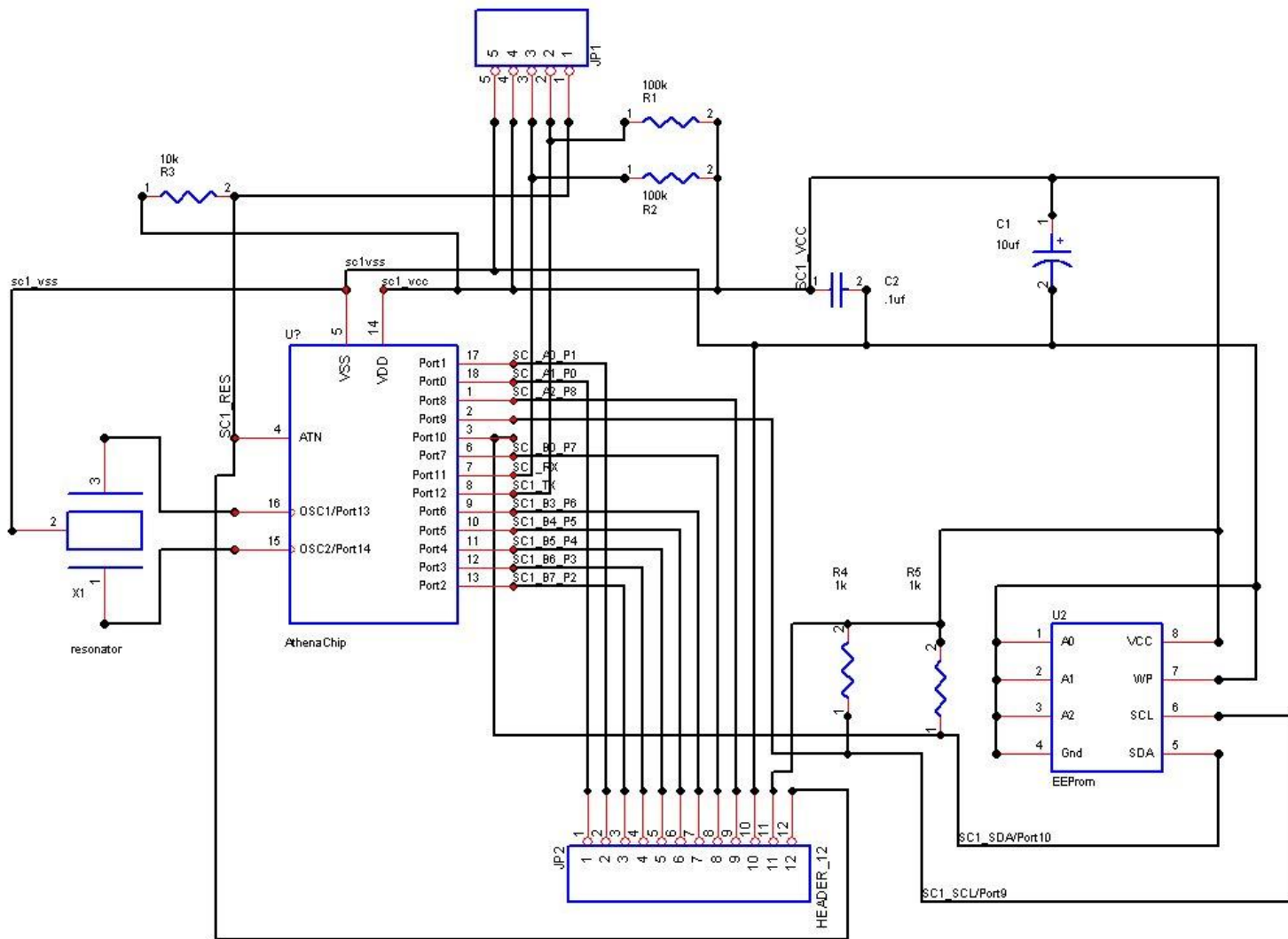
Appendix G: Athena Carrier 1 Hookup



The program interface (header) on the Carrier 1 was designed so that you could plug the board into a breadboard then plug the EZ232 driver directly in as shown.



Carrier 1 Schematic



Appendix H: Athena EDU/Carrier 3 Hookup

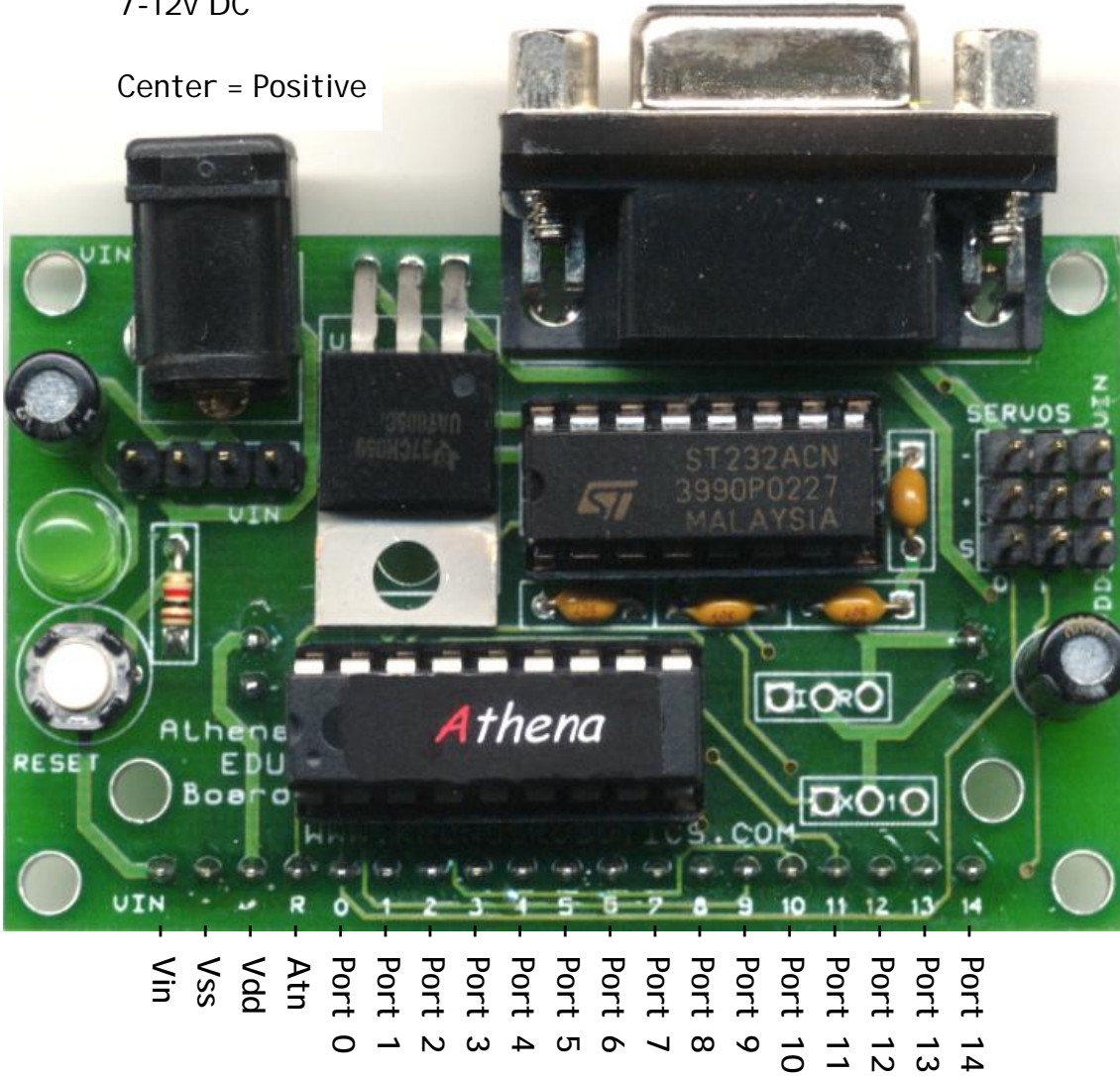
2.1 Coax Connector
7-12v DC

Center = Positive

PC Program Cable Connector

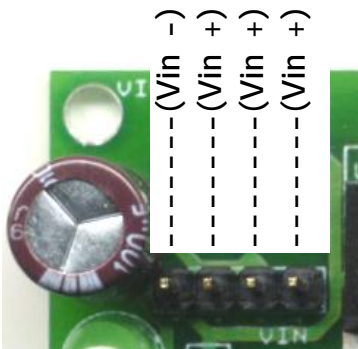
This carrier is compatible with the following chips:

- Athena
- AthenaHS
- Athena485



Port 14
Port 13
Port 12
Port 11
Port 10
Port 9
Port 8
Port 7
Port 6
Port 5
Port 4
Port 3
Port 2
Port 1
Port 0
Atn
Vdd
Vss
Vin

Battery Connector
7-12v DC

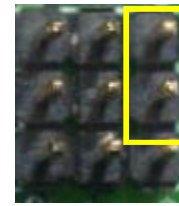


Servo Power -
Servo 1 -
Servo 0 -



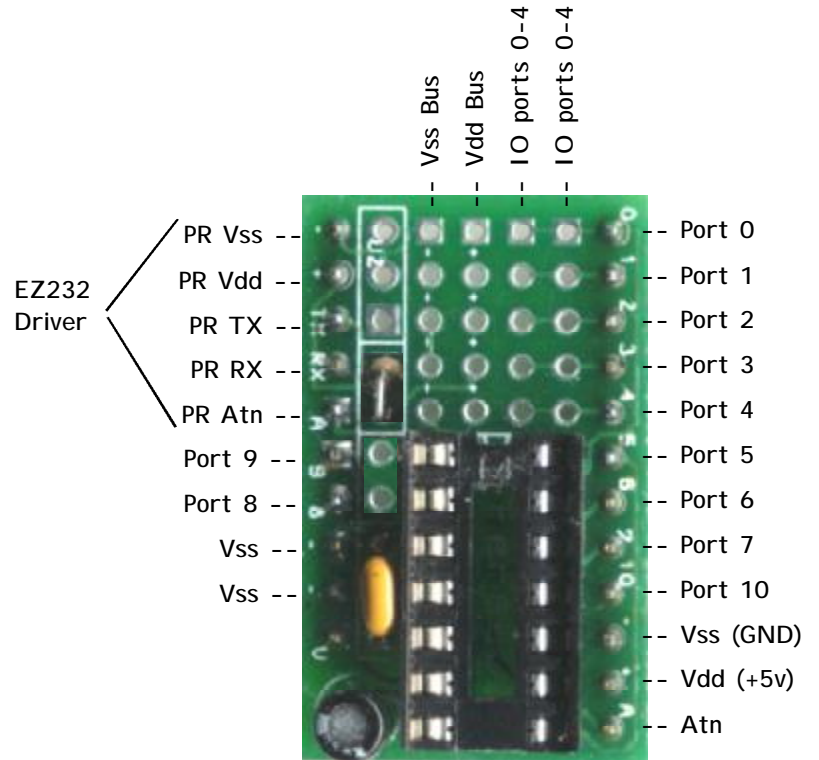
Servo Power from 5v Regulator.

Servo Power -
Servo 1 -
Servo 0 -

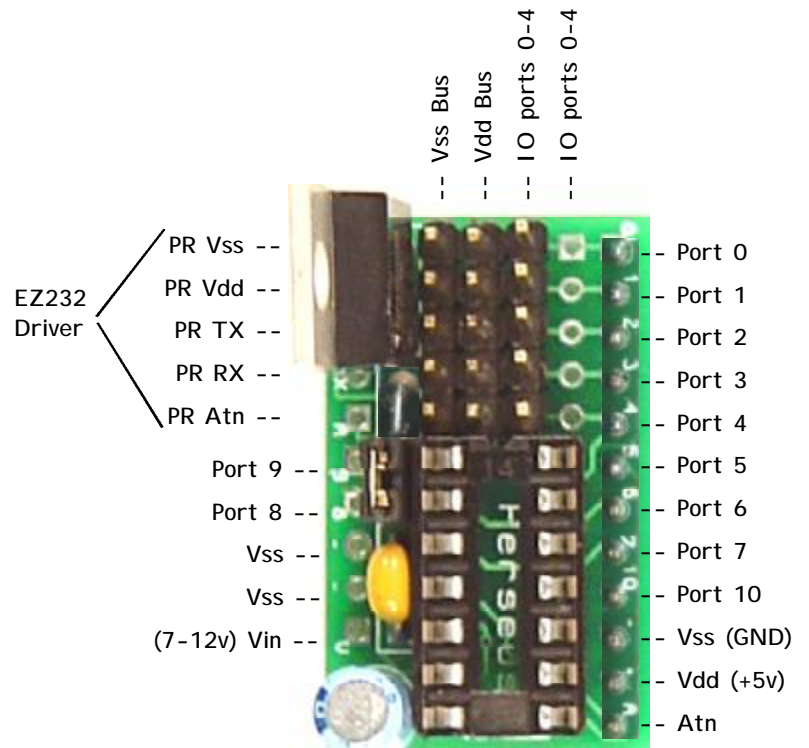


Servo Power from Vin. Use only if Vin is less than 7.5 volts.

Standard Hookup



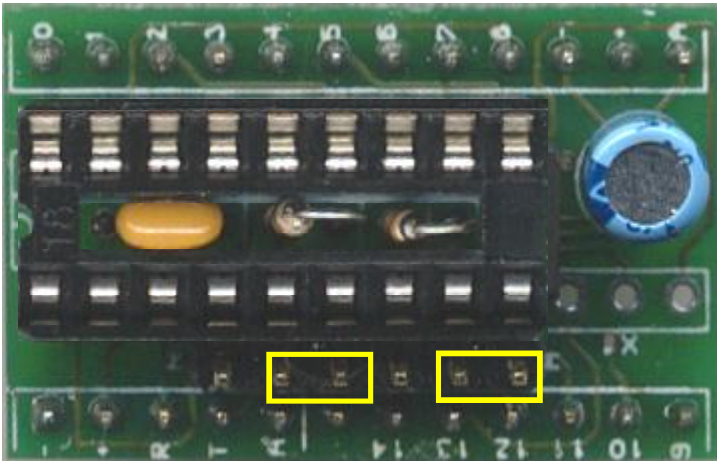
Option Pack Hookup



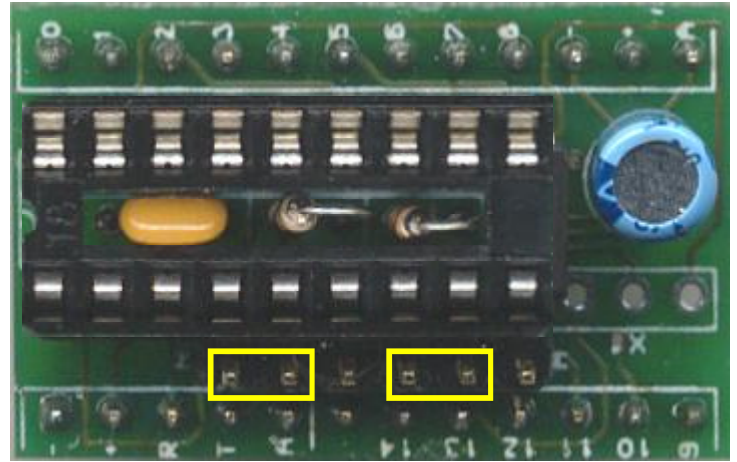
J1 Enables the 7805 regulator. For low power/low voltage operations remove.

Appendix J: Athena/Nemesis Carrier 1u Hookup

This carrier 1u has 2 jumpers that are used to select the type microcontroller you are using.



Athena
AthenaHS
Athena485



Nemesis
NemesisHS

Note:

If you are using this carrier for a NemesisHS or AthenaHS you will need to connect a 20Mhz Resonator to X1

