
BASIC MICRO
TECHNOLOGY AT WORK
BasicATOMPro
Syntax Manual
Unleash The Power Of The Basic Atom Pro
Revision 7.2.0.0

Warranty

Basic Micro warranties its products against defects in material and workmanship for a period of 90 days. If a defect is discovered, Basic Micro will at our discretion repair, replace, or refund the purchase price of the product in question. Contact us at support@basicmicro.com No returns will be accepted without the proper authorization.

Copyrights and Trademarks

Disclaimer

Basic Micro cannot be held responsible for any incidental, or consequential damages resulting from use of products manufactured or sold by Basic Micro or its distributors. No products from Basic Micro should be used in any medical devices and/or medical situations. No product should be used in a life support situation.

Contacts

Web: http://www.basicmicro.com

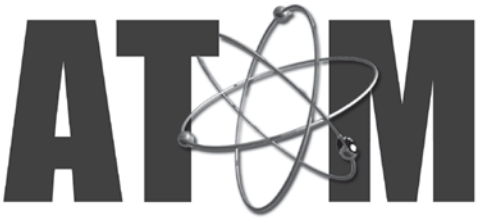
Discussion List

A web based discussion board is maintained at http://www.basicmicro.com

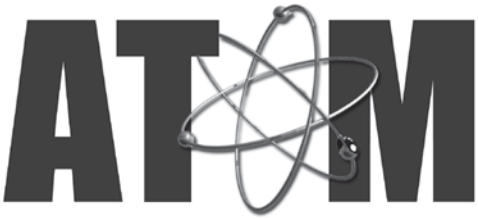
Updates

In our continuing effort to provide the best and most innovative products, software updates are made available in the download section of the Basic Micro website at http://www.basicmicro.com

Table of Contents



Contents


ATOM

Introduction

What is the BasicATOM-Pro ?

The BasicATOM-Pro is the next generation of the BasicATOM product line. It is a self contained microcontroller which is programmed using an advanced programming language modeled after BASIC, creating a cost effective and flexible way to program the BasicATOM-Pro hardware. The BasicATOM-Pro maintains the simplicity of Basic, but offers more power.

This Manual

This manual in general applies to the BasicATOM-Pro programming syntax. Certain information may only pertain to a given version of the product.

This manual documents the BasicATOM-Pro's programming language in depth . The main purpose of this manual is to teach the general syntax.

For more information about a particular device refer to its Data Sheet. All Data Sheets are available from the download section of the Basic Micro website. Http://www.basicmicro.com

We will continue to update and improve this manual. All updates will be made available for download from our web site at http://www.basicmicro.com.

On-line Discussion Forums

We maintain discussion forums at http://www.basicmicro.com in order to help you to connect with a wide range of related information and users. The discussion forums are free and will allow you to find information and help fast.

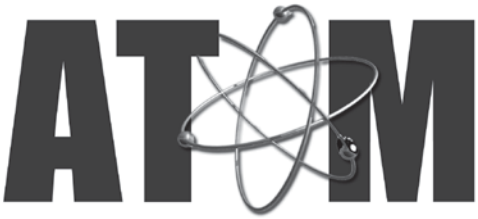
Updates

The BasicATOM-Pro software updates are available to new and current customers from the Basic Micro website download section.

Technical Support

Technical support is provided via the Basic Micro online Support system and the discussion forums at www.basicmicro.com. When technical support is required please fill out a Support Form in the Support section of our website. In order to assure a proper response please include a copy of the program you are having problems with, the hard-

ware you are using, BasicATOM-Pro software version number, prototyping board and so on. By including this information with your e-mail, you can help us answer your questions much faster. Additional technical support is often provided by our forum moderators in the discussion boards at our website(http://www.basicmicro.com).


ATOM

General Theory of Operation

The BasicATOM-Pro(here after refered to as the ATOM-Pro) is a tiny computer, or better known as a microcontroller. The ATOM-Pro was designed for use in a wide array of applications. The ATOM-Pro is built around the Hitachi H8/TINY family, which contains internal memory (2048 Bytes of RAM and 32K of FLASH program memory). Each ATOM has a built-in 5-volt regulator, a number of general-purpose I/O pins (TTL-level, 0-5 volts), commands for math and I/O pin operations and a serial port for in circuit programming.

The ATOM-Pro Language

The ATOM-Pro language is a simple, easy to learn platform based on BasicMicro's MBasic as well as some specialized instructions for the ATOM hardware.

How the ATOM-Pro Works

The ATOM-Pro's software brains are not permanently stored in the CPU. This has many advantages. By not storing the software brains on the ATOM-Pro, new commands and functionality can easily be added without the need for new hardware. Once a program is written the ATOM-Pro's software will compile what is needed to run correctly.

Since the software brain is not pre loaded, any program even the smallest will have a minimum size of about 500 bytes out of 32,000bytes regardless of the commands used. This method allows minimum command / function duplication. If you were to use the SERIN or SEROUT commands more than once in your program only minimal code would be added. The ATOM-Pro can use the same internal code to perform the task of sending or receiving serial data throughout the program. The same would apply to any other commands used in the program.

Small programs will increase in size rapidly until they reach about 3KB in size. The rate at which a programs size grows will slow as it grows simply because internal code will be used over again.

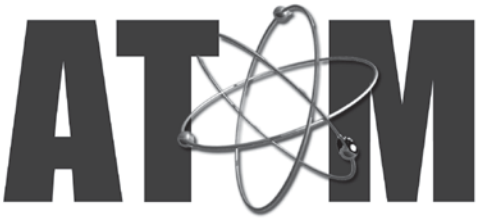
Hardware

There are several models of the ATOM-Pro available. For specific information refer to the data sheets regarding the version you are using. All data sheets can be downloaded from the downloads section of the Basic Micro website at http://www.basicmicro.com

The ATOM-pro is programmed at 115Kbps. This means that the ATOM-Pro interface software will not work on a computer that does not have serial ports capable of 115Kbps. Most computers that have been shipped since 1996 have serial ports capable of 115Kbps.

If you are using a laptop or new computer that does not have the traditional DB-9 serial connector the ATOM-Pro will program from a USB to serial adapter.

The ATOM-Pro Integrated Development Environment(IDE) software only works in a Windows environment. There is no DOS support and no plans for future DOS support.



The Basic's

Line Labels

In order to access different sections of code you must use line labels. Unlike the original Basic language, MBasic does not use line numbers.

example:

```
Loop: goto Loop          ;This line repeats infinitely
```

The above goto statement jumps to a line label LOOP, which is in front of the GOTO statement. The above line will repeat infinitely. Line labels can not be duplicated or used as variable names once defined as a label.

RAM and Program Memory

RAM, or random access memory, is where variable values, and system values are stored. RAM is also used to store the return location of GOSUB statements. The ATOM-Pro has about 2000 bytes of user RAM available.

Program Memory is the memory where your program will reside. The size of available program memory will limit the size of your program. The more complicated the program , the more memory you will use. Most ATOM-Pro modules have 32kbytes of program memory(Some have 56kbytes of memory).

Variables

Variables are used to store temporary information in the program. They are created using the VAR keyword. Variables can be BITs, NIBBLEs, BYTEs, WORDs and LONGs. Before you can use a variable it must be defined.

example:

Variable name:	Variable:	Size:
Temp	Var	Byte

The above states Temp is allocated a byte (8 bits) of ram

Variable names must start with a letter. They can contain letters, numbers and special characters. However they can not be the same name as ATOM-Pro reserved words or labels used in a program. The same variable name can not be defined twice. The ATOM-Pro does not distinguish between upper and lower case, so the name “TEMP” is equivalent to “temp”. The maximum character length can be up to 1024 characters.

Throughout this manual and when dealing bits, bytes, words ,longs and floats will be referred to often. The following is a quick break down of the values these different variable types can hold:

Type	Bit Size	Range
Bit	1	1 or 0
Nib	4	0 to 15
Byte	8	0 to 255
SByte	8	-127 to +128
Word	16	0 to 65535
SWord	16	-32767 to +32768
Long	32	0 to 4,294,967,295
SLong	32	-2147483647 to +2147483648
Float	32	+-2^-126 to 2^127

Some examples of defined variables:

```
DOG     Var     Bit     ;0 or 1
POST    Var     Nib     ;0 to 15
LOG     Var     Byte    ;0 to 255
STICK   Var     Word    ;0 to 65535
TREE    Var     Long    ;0 to 4,294,967,295
```

Arrays

As your programs begin to perform more complex tasks, there will be times when you want a variable to hold many values. An Array is a structure that can store multiple values of the same type.

example:

```
Temp        var     Word(5)
```

The number 5 in parenthesis shows the variable temp has 5 cells. Once the array has been defined, each cell can be accessed by its number:

```
Temp(0) = 10
Temp(1) = 25
Temp(2) = 45
Temp(3) = 55
Temp(4) = 65
```

The above will assign the value of 10 to the first cell in the 5 cell array, 25 to the second cell and so on. Using arrays can simplify your program as shown below:

```
Temp        Var     Byte(5)         ;variable temp now has 5 cells
Cntr        Var     Byte

For Cntr = 0 to 4               ;Set each cell to Cntr + 2
    Temp(Cntr) = Cntr + 2
Next
```

The above code example will load each array, 0 to 4 (5 cells) with the array number + 2. To do this manually:

```
Temp(0) = 2
Temp(1) = 3
Temp(2) = 4
Temp(3) = 5
Temp(4) = 6
```

Tables

Label TableType Data, Data, ...Data

Label is the name of the table used to call or access the table.
TableType is the size of the table data.

Tables Types:
ByteTable(8bit data)
WordTable(16bit data)
LongTable(32bit data)
FloatTable(floating point data)

Data: the constant value or constant expression to store in the table.

explanation:
Tables are used to store constant data(doesn't change after programming) which can be accessed like an array variable.

example:
FirstMenu ByteTable "Enter An Option",0

FirstMenu(0) equals "E"
FirstMenu(1) equals "n"
....

note:
The ending NULL(0) is usefull when using tables with STR modifiers.

Aliases

Aliases are alternate names for defined variables. As an example:

```
DOG  Var  Byte   ;DOG is assigned as an 8 bit variable (Byte)
CAT  Var  DOG    ;CAT now points to the variable DOG
```

In the above example if DOG were equal to 10, any time the variable CAT was accessed it would equal 10 since it points to the same RAM location. Aliases are a good idea when you want to use a temporary variable with a name that suits its function.

Variable Modifiers

Variable modifiers are used to access only parts of a variable.

In example 1 we show how to alias a variable(point a new variable at the data of a previously defined variable) using a variable modifier. In example 1 the “highbyte” modifier is used. This points the aliased variable, “Cat”, at the high byte(bits 8-15) of “Dog”.

example 1:

```
Dog   Var   Word
Cat   Var   Dog.HighByte
```

In example 2 we show how to access parts of variables on the fly. In this example “Cat” is being loaded with a value from “Dog”. Here we are geting the second byte of the two byte(word) variable, “Dog”, and storing it in “Cat” by using the “Byte1” modifier.

example 2:

```
Dog Var Word
Cat Var Byte

Cat = Dog.Byte1
```

The table below shows all the different variable modifiers that can be used:

Modifier	Create alias to
LOWBIT	bit 0 of variable
BIT0	bit 0 of variable
BIT1	bit 1 of variable
BIT2	bit 2 of variable
BIT3	bit 3 of variable
BIT4	bit 4 of variable
BIT5	bit 5 of variable
BIT6	bit 6 of variable
BIT7	bit 7 of variable
BIT8	bit 8 of variable
BIT9	bit 9 of variable
BIT10	bit 10 of variable
BIT11	bit 11 of variable
BIT12	bit 12 of variable
BIT13	bit 13 of variable
BIT14	bit 14 of variable
BIT15	bit 15 of variable
BIT16	bit 16 of variable
BIT17	bit 17 of variable
BIT18	bit 18 of variable
BIT19	bit 19 of variable
BIT20	bit 20 of variable
BIT21	bit 21 of variable
BIT22	bit 22 of variable
BIT23	bit 23 of variable
BIT24	bit 24 of variable
BIT25	bit 25 of variable
BIT26	bit 26 of variable
BIT27	bit 27 of variable
BIT28	bit 28 of variable
BIT29	bit 29 of variable
BIT30	bit 30 of variable
BIT31	bit 31 of variable

HIGHBIT	highest bit of variable

Modifier	Create alias to
LOWNIB	nibble 0 of variable
NIB0	nibble 0 of variable
NIB1	nibble 1 of variable
NIB2	nibble 2 of variable
NIB3	nibble 3 of variable
NIB4	nibble 4 of variable
NIB5	nibble 5 of variable
NIB6	nibble 6 of variable
NIB7	nibble 7 of variable
HIGHNIB	highest nibble of variable
LOWBYTE	byte 0 of variable
BYTE0	byte 0 of variable
BYTE1	byte 1 of variable
BYTE2	byte 2 of variable
BYTE3	byte 3 of variable
HIGHBYTE	highest byte of variable
LOWWORD	word 0 of variable
WORD0	word 0 of variable
WORD1	word 1 of variable
HIGHWORD	word 1 of variable

Note: Variable modifiers can also be used in code statements

example:

```
if myvar.bit0 = 1 then dosomething
```

Pin Variables

Pin variables are just like any other variables except that the states of the individule bits in the variables are the states/directions of the corresponding pin.

DIRE a 32bit variable accessing the directions of P0-P31.
DIRS a 16bit variable accessing the directions of P0-P15.
DIRES a 16bit variable accessing the directions of P16-P31.
DIRL an 8bit variable accessing the directions of P0-P7.
DIRH an 8bit variable accessing the directions of P8-P15.
DIREL an 8bit variable accessing the directions of P16-P23.
DIREH an 8bit variable accessing the directions of P24-P31.
DIRA a 4bit variable accessing the directions of P0-P3.
DIRB a 4bit variable accessing the directions of P4-P7.
DIRC a 4bit variable accessing the directions of P8-P11.
DIRD a 4bit variable accessing the directions of P12-P15.
DIREA a 4bit variable accessing the directions of P16-P19.
DIREB a 4bit variable accessing the directions of P20-P23.
DIREC a 4bit variable accessing the directions of P24-P27.
DIRED a 4bit variable accessing the directions of P28-P31.
DIR# (# is any number from 0 to 31) is a variable that accesses the direction of P0 - P31 individually.

INE a 32bit variable accessing the states of P0-P31.
INS a 16bit variable accessing the states of P0-P15.
INES a 16bit variable accessing the states of P16-P31.
INL an 8bit variable accessing the states of P0-P7.
INH an 8bit variable accessing the states of P8-P15.
INEL an 8bit variable accessing the states of P16-P23.
INEH an 8bit variable accessing the states of P24-P31.
INA a 4bit variable accessing the states of P0-P3.
INB a 4bit variable accessing the states of P4-P7.
INC a 4bit variable accessing the states of P8-P11.
IND a 4bit variable accessing the states of P12-P15.
INEA a 4bit variable accessing the states of P16-P19.
INEB a 4bit variable accessing the states of P20-P23.
INEC a 4bit variable accessing the states of P24-P27.
INED a 4bit variable accessing the states of P28-P31.
IN# (# is any number from 0 to 31) is a variable that accesses the state of P0 - P31 individually.

OUTE a 32bit variable accessing the states of P0-P31.
OUTS a 16bit variable accessing the states of P0-P15.
OUTES a 16bit variable accessing the states of P16-P31.
OUTL an 8bit variable accessing the states of P0-P7.

OUTH an 8bit variable accessing the states of P8-P15.
OUTEL an 8bit variable accessing the states of P16-P23.
OUTEH an 8bit variable accessing the states of P24-P31.
OUTA a 4bit variable accessing the states of P0-P3.
OUTB a 4bit variable accessing the states of P4-P7.
OUTC a 4bit variable accessing the states of P8-P11.
OUTD a 4bit variable accessing the states of P12-P15.
OUTEA a 4bit variable accessing the states of P16-P19.
OUTEB a 4bit variable accessing the states of P20-P23.
OUTEC a 4bit variable accessing the states of P24-P27.
OUTED a 4bit variable accessing the states of P28-P31.
OUT# (# is any number from 0 to 31) is a variable that accesses the state of P0 - P31 individually.

IN and OUT pin variables are interchangable. Either one can be used to read or write a pin state. The two different names are provided to make code more easy understood.

Constants

Constants are similar to variables except their values are set at compile time and can not be changed. When creating a program it can be beneficial to use constants for certain values that don't change.

example:

```
Meter       CON   1                   ;Meter = 1
Centimeter  CON   Meter * 100         ;Centimeter = 100
Millimeter  CON   Centimeter * 10     ;Millimeter = 1000
```

In the above example "centimeter" and "millimeter" values were derived from the constant "meter". There are a 100 centimeters in a meter and a 1000 millimeters in a meter.

Pin names are also constants so they can be used in the following way:

```
RedLed    Con P0
GreenLed  Con P1

Main
    High RedLed
    High GreenLed
Goto Main
```

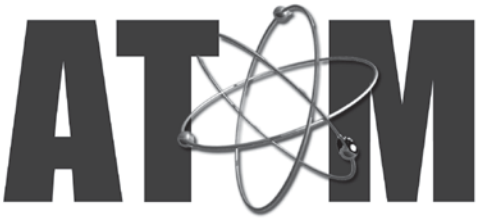
RedLed and GreenLed are now constants that point to pin 0 and pin 1. When writing complex programs it may be beneficial to use constants as shown above.

Pin constants

Pin constants are simple predefined constants for the different pin numbers on the ATOM-Pro. All ATOM-Pro modules/boards have 16 common I/O pin names. See specific ATOM-Pro module datasheets for each modules extended list of pin names.

P0 = 0
P1 = 1
P2 = 2
P3 = 3
P4 = 4
P5 = 5
P6 = 6
P7 = 7
P8 = 8
P9 = 9
P10 = 10
P11 = 11
P12 = 12
P13 = 13
P14 = 14
P15 = 15

Also each ATOM-Pro module has two specialized pins(S_IN and S_OUT) for programming and for use as serial input/output.


ATOM

Preprocessor

The ATOM-Pro compiler's preprocessor commands are used to include external files and to conditional compie sections of code.

Including files

By using the *#include* preprocessor directive the user can modularize their program. The #include directive acts like a compile time paste. The text in the file specified in the #include directive will be placed into the users program at the location of the directive at compile time.

For example, if you have a basic file called "myfuncs.bas" you can add the code from this file to your compiled program by using the #include directive like this:

```
main
    gosub myfunc1      ;myfunc1 is defined in myfuncs.bas
    goto main

#include "myfuncs.bas"
```

This assumes that the myfuncs.bas file is in the same directory as the main program. If the included file is not in the same directory you must include the full or partial path name.

For example, if "myfuncs.bas" is in a sub directory of the directory the main program is in you can:

```
#include "mysubdir\myfuncs.bas"
```

Or you can specify the full path:

```
#include "c:\mybasprogs\mysubdir\myfuncs.bas"
```

Conditional compiling

Conditional compiling is used when you don't always want some sections of your code to be compiled into the program. Using the conditional directives you can specify whether certain lines of code are compiled into your program or not based on whether something was previously defined.

#IF *constant expression*

The #IF directive is used to specify user code that will only be compiled in the program if the constant expression is true(ie non zero).

Example:

```
#IF mycon = 120
    ...code...
#ENDIF
```

#IFDEF *name*

The #IFDEF directive is a special case of the #IF directive. Its argument must be a name(ie label,variable or constant name). If the name was defined previously in the program the code inside the directive will be compiled.

Example:

```
mycon con 10
#IFDEF mycon
    ...code...
#ENDIF
```

#IFNDEF *name*

The #IFNDEF directive is a special case of the #IF directive. Its argument must be a name(ie label,variable or constant name). If the name was not defined previously in the program the code inside the directive will be compiled.

Example:

```
#IFNDEF mycon
    ...code...
#ENDIF
```

#ELSEIF *constant expression*

The #ELSEIF directive is used to allow multiple conditions easily

Example:

```
#IF mycon = 120
    ...code...
#ELSEIF mycon = 130
    ..code..
#ELSEIF mycon = 140
    ..code..
#ENDIF
```

#ELSEIFDEF *name*

The #ELSEIFDEF directive is used to allow multiple conditions easily

Example:

```
#IFDEF mycon1
    ...code...
#ELSEIFDEF mycon2
    ..code..
#ELSEIFDEF mycon3
    ..code..
#ENDIF
```

#ELSEIFNDEF *name*

The #ELSEIFNDEF directive is used to allow multiple conditions easily

Example:

```
#IFNDEF mycon1
    ...code...
#ELSEIFNDEF mycon2
    ..code..
#ELSEIFNDEF mycon3
    ..code..
#ENDIF
```

#ELSE

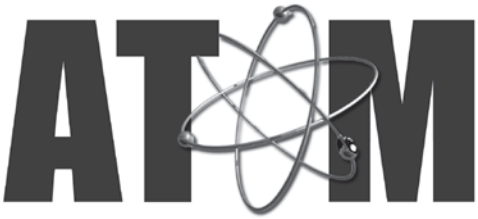
The #ELSE directive can be used with any #IF directive. When the #IF directive is false the code inside the #ELSE directive will be added to the compiled program instead.

Example:

```
#IF mycon = 10
    ...code...
#ELSE
    ..code..
#ENDIF
```

#ENDIF

All conditional compiling directives must end with an #ENDIF.


ATOM

Numerical Types

Numbers can be written in different ways. Binary numbers are written using only 0 and 1. Hexadecimal uses characters '0' to 'F'. Binary and hexadecimal numbers must have an indicator.

1234 or d'1234'	: Standard Decimal number
$1F2A or 0x1F2A	: Hexadecimal notation
%1001	: Binary notation

The character $(string) or "0x" indicates Hexadecimal and the percentage sign % indicates binary data. These special characters must be used in order to let the Atom know what numerical types they are.

Operator Precedence

The ATOM-Pro uses standard algebraic syntax. In the ATOM-Pro 2+2*5/10 = 3. This is because in the ATOM-Pro each math operator has a precedence. The multiply and divide operators have equal precedence. In the above calculation 2*5 will be calculated first (equaling 10), then the divide by 10 (equals 1), then the addition of 2 (equaling 3). You can use parenthesis to force specific orders (i.e.: ((2+2)*2) /2 would calculate the value the way the Basic Stamp does(ie left to right with no precedence).

Order:	Operation:
1st	NOT, ABS, SIN, COS, - (NEG), DCD, NCD, SQR, RANDOM, TOINT, TOFLOAT, BIN2BCD, BCD2BIN, ~(Binary NOT), !(Binary NOT), NOT(Logical NOT), FSQRT, FSIN, FCOS, FTAN, FASIN, FACOS, FATAN, FSINH, FCOSH, FTANH, FATANH, FLN, FEXP
2nd	Rev, Dig
3rd	MAX, MIN
4th	*, **, */, /, //
5th	+, -
6th	<<, >>
7nd	<, <=, =, >=, >, <>
8th	&, |, ^, &/, |/, ^/
9rd	And, Or, Xor

Math Functions

The following is a list of math functions the ATOM-Pro can perform.

UNARY Commands

-(NEG) expression	Negate value
ABS expression	Absolute value
SIN expression	sine of value(0-255)
COS expression	cosine of value(0-255)
DCD expression	2 to the nth power(n = value)
NCD expression	smallest power of 2 that is GREATER than value.
SQR expression	square root of value.
BIN2BCD expression	Integer to Packed BCD format
BCD2BIN expression	Packed BCD to integer.
RANDOM expression	Generate Rangom value with seed expression
NOT expression	Logical inverse

Floating Point UNARY Commands

TOINT expression	Converts a Floating Point value to an Integer value.
TOFLOAT expression	Converts an Integer value to a Float ing Point value.
FSQRT expression	Square Root
FSIN expression	Sin(range: PI/2 to -PI/2)
FCOS expression	Cos(range: PI/2 to -PI/2)
FTAN expression	Tan(range: PI/2 to -PI/2)
FASIN expression	ArcSin(range: 1 to -1)
FACOS expression	ArcCos(range: all values)
FATAN expression	ArcTan(range: 1 to -1)
FSINH expression	Hyperbolic Sin(range: 1.13 to -1.13)
FCOSH expression	Hyperbolic Cos(range: 1.13 to -1.13)
FTANH expression	Hyperbolic Tan(range: 1.13 to -1.13)
FATANH expression	Hyperbolic ArcTan (range: 1.13 to -1.13)
FLN expression	Natural Log(range: 9.58 to 0.1)
FEXP expression	Exponent(range: 1.13 to -1.13)

BINARY Commands

exp1 + exp2	Add exp1 to exp2
exp1 - exp2	Sub exp2 from exp1
exp1 * exp2	Mulitply exp1 by exp2
exp1 ** exp2	Get high 32bits of a multiply
exp1 */ exp2	Fractional Multiply
exp1 / exp2	Divide exp1 by exp2
exp1 // exp2	Mod exp1 by exp2
exp1 MAX exp2	smaller of the two expressions.
exp1 MIN exp2	larger of the two expressions.
exp1 DIG exp2	digit of exp1 at exp2 position.
exp1 REV exp2	reverses exp2 bits of exp1 starting with LSB

Bitwise Operators

Bitwise operators are commands that directly effect the bits of a value.

Bitwise operators

exp1 & exp2	And exp1 with exp2
exp1 | exp2	Or exp1 with exp2
exp1 ^ exp2	XOr exp1 with exp2
exp1 >> exp2	Shift exp1 right by exp2
exp1 << exp2	Shift exp1 left by exp2
~(NOT) expression	Invert exp1
!(NOT)	Invert exp1

Comparison Operators

Comparison operators are used when comparing two or more values. Examples are the IF...THEN and LOOKDOWN commands.

Compare Op.	Description
=	Equal
<>	Not Equal
<	LessThan
>	GreaterThan
<=	LessThan Equal
>=	GreaterThan or Equal

Logical Operators

Logical operators are slightly different in use than comparison operators. When an IF...THEN statement contains more than one comparison you must combine them using a logical operator. The example below illustrates this:

```
If (Variable < 100) AND (Variable > 10) Then Label
```

As you can see from the example if BOTH are true then the program jumps to the label.

Logical Op.	Description
AND	Logical AND
OR	Logical OR
XOR	Logical Exclusive OR
NOT	Logical NOT

Floating Point Format

The floating point math the ATOM-Pro uses differs some what from the normal IEEE 754 floating point standard.

IEEE format:

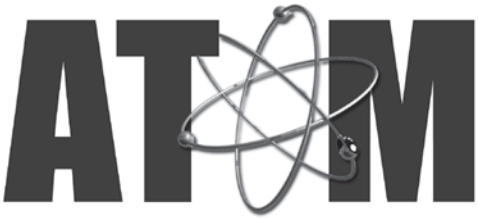
Bit 31 = Sign bit(S)
Bit 30-23 = Exponent(E)
Bit 22-0 = Mantissa(M)

ATOM-Pro format

Bit 31-24 = Exponent(E)
Bit 23 = Sign bit(S)
Bit 22-0 = Mantissa(M)

```
          32.31.30.29.28.27.26.25.24.23.22.21.20...0
IEEE    S E   E E   E E E   E E M M M M...M
ATOM  E E   E E   E E E   E S M M M M...M
```

Note: All variables that will contain a floating point number must be a FLOAT type variable.



Modifier usage

Command modifiers can be used to modify/enhance data in a command directly. Modifiers can be used with any commands that show {Modifier or Mods} in their syntax.

I/O Modifiers

dec	Decimal Value
hex	Hexadecimal Value
bin	Binary Value
str	Input or Output Array Variables

Signed I/O Modifiers

sdec	Decimal Value
shex	Hexadecimal Value
sbin	Binary Value

Indicated I/O Modifiers

ihex	Hexadecimal Value
ibin	Binary Value

Combination I/O Modifiers

ishex	Hexadecimal Value
isbin	Binary Value

Output Only Modifiers

rep	Output character *n* times
real	Output Floating point numbers

Input Only Modifiers

waitstr	Waits until values received match array
wait	Waits util values received match string of values
skip	Skip *n* values

HEX - DEC - BIN

desc(input):

Convert input ASCII characters to binary. Input must be in HEX,DEC or BIN format

desc(output):

Convert a binary value to ASCII characters in HEX,DEC or BIN format

syntax:

modifier{#1} arg{\#2}

#1: optional number that sets a maximum number of digits to pass

#2: optional value that sets a minimum number of digits to pass

example:

command args,[dec2 1234\2] ;output "34"

SDEC - SHEX - SBIN

desc(input):

Convert input ASCII characters to binary. Input must be in HEX,DEC or BIN format. "-" is a valid sign character.

desc(output):

Convert a binary value to ASCII characters in HEX,DEC or BIN format with sign("-") if negative

syntax:

modifier{#1} arg{\#2}

#1: optional number that sets a maximum number of digits to pass

#2: optional value that sets a minimum number of digits to pass

example:

command args,[sdec -1234] ;output "-1234"

IHEX - IBIN

desc(input):

Convert input ASCII characters to binary. Input must be in HEX,DEC or BIN format. Input characters are ignored until a valid indicator character is received

desc(output):

Convert a binary value to ASCII characters in HEX,DEC or BIN format. An indicator character is passed first.

indicator chars:

HEX: “$”

BIN: “%”

syntax:

modifier{#1} arg{\#2}

#1: optional number that sets a maximum number of digits to pass

#2: optional value that sets a minimum number of digits to pass

example:

command args,[ihex $ABCD] ;output “$ABCD”

ISHEX - ISBIN

desc(input):

Convert input ASCII characters to binary. Input must be in HEX,DEC or BIN format with optional sign character(“-”). Input characters are ignored until a valid indicator character is received

desc(output):

Convert a binary value to ASCII characters in HEX,DEC or BIN format. An indicator character is passed first. If the value is negative a sign character is pased next(“-”).

indicator chars:

HEX: “$”

BIN: “%”

syntax:

modifier{#1} arg{\#2}

#1: optional number that sets a maximum number of digits to pass

#2: optional value that sets a minimum number of digits to pass

example:

command args,[ihex -$ABCD] ;output “$-ABCD”

REAL
desc(output only):
Convert a Floatingpoint value to ASCII characters. Sign and decimal point are handled.

syntax:
modifier{#1} arg{\#2}
#1: optional number that sets the maximum digits to pass before the decimal point.(Default: 10)
#2: optional value that sets the maximum digits to display after the decimal point.(Default: 10)

example:
command args,[real 1.1234] ;output "1.1234000000"

REP
desc(output only):
Repeat a character *n times*

syntax:
modifier arg*n*

example:
command args,[rep "a"\20] ;output the letter "a" 20 times

STR
desc(input):
Receive a variable number of values and store in a variable array

desc(output):
Output the elements of a variable length array.

syntax:
str value{*length*{*eol*}}
length: optional value that sets the maximum number of values to pass
eol: optional value that sets the end of line(EOL) character to stop passing data on. *length* is required when using *eol*

example:
command args,[str myarray\10\"c"] ;output upto 10 ;values in myarray. ;Stop passing values ;on "c".

WAITSTR
desc(input only):
Receive value until a continous group matches string

syntax:
waitstr sting*length*{*eol*}
length: optional value that sets the maximum number of values to match
eol: optional value that sets the end of line(EOL) character to stop matching data on. *length* is required when using *eol*

example:
command args,[waitstr string\10\"c"]

WAIT
desc(input only):
Receive value until a continuous group matches constant string

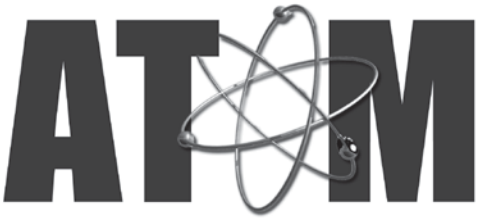
syntax:
wait("my constant string")

example:
command args,[wait("My string")]

SKIP
desc(input only):
Skip specified number of values

syntax:
skip *count*

example:
command args,[skip 10]


ATOM

ADIN

ADIN pin,variable
Convert Analog signal on *pin* and store value in *variable*.

Pin: a constant or variable that specifies the pin number. The specified pin number must be cabable of A/D conversion.

Variable: a word or long size variable

Explanation

The ADIN command is used to convert an analog voltage(0-5v) into a number from 0 to 1023. The value is stored in *variable*.

Branch

BRANCH index, [Label1,...LabelN]
Go to the Label specified by index.

Index is an expression that points to the label to jump to in the list of labels in the Branch command.

Label1,...LabelN a list of labels.

Explanation

The Branch command allows the program to jump to different locations based on a variable index. BRANCH is used to simplify code like this:

```
IF temp = 0 THEN dog          ;temp =0; go to label dog
IF temp = 1 THEN cat          ;temp =1: go to label cat
IF temp = 2 THEN mouse        ;temp =2: go to label mouse
```

into code like this:

```
BRANCH temp, [dog, cat, mouse]
```

If the index is greater than the number of Labels in the list then the command exits and program execution continues on the next line.

Button

BUTTON pin, pressedstate, repeatdelay, repeatrate, workbyte, logicstate, label

Reads the pin, debounces the button input, performs an auto-repeat if activated, and branches to a label if logical state is active. The Button may be activated in either a low state or a high state based on the logicalstate.

Pin is an expression of the pin number the button/switch is connected to. The pin will be made an input.

PressedState is an expression(0 or 1) which specifies the voltage when the button is pressed(Gnd or Vdd).

RepeatDelay is an expression(0–255) which sets the functions of the button command. If equal to 0 debounce and auto-repeat are disabled. If equal to 255, BUTTON executes a debounce(one loop), but auto-repeat is disabled. All other values(1-254) are the number of program loops the button command must execute before autorepeat begins.

RepeatRate is an expression(0–255) of the number of program loops BUTTON executes before each repeat

WorkByte is a work space variable used internally by the BUTTON command to store current loop counts for use when debouncing, delaying, and auto-repeating. This variable must be unique inside the program loop BUTTON is running in.

Logicalstate is an expression(0 or 1) which determines the logical state the button must be in for a branch to occure(pressed or not pressed).

Label is the label of the location in the user program to jump/branch to when the targetstate is triggered.

Explanation

The BUTTON command works much like a key on a PC keyboard. When a switch/button is closed or opened(depending on command arguments), the BUTTON command will jump/branch to the specified label. The BUTTON command also allows the user to specify the delay before auto-repeating and how fast to auto-repeat(if at all).

DEBOUNCE

When a switch or button contact is closed the mechanical connections may bounce. This can cause a period where the state of the switch or button can not acurately be determined. Debouncing the input rereads the input state one program loop after the first read of the pin the switch/button is connected to inorder to confirm the state of the input.

Clear

CLEAR
Clear user RAM.

Explanation

The Clear command will clear (Set to 0's) all of the user ram. User ram is set aside space for all the variables a user program will use.

Count

COUNT pin, period, variable

Count the number of cycles (0-1-0 or 1-0-1) on the specified pin during period number of milliseconds and store that number in variable.

Pin is an expression of the I/O pin number to use.
This pin will be placed into input mode.

Period is an expression(1 to 4294967296) of the time in millisec onds during which to count.

Variable is a variable where the count will be stored.

Explanation

COUNT checks the state of PIN in a tight loop and will count the low to high transitions. COUNT is ideal for figuring out frequency of certain waves or timings based on an incoming signal.

Debug

DEBUG [{Options} item, {{Options} item}]
Sends values of specified variables or constants to the debug watch window.

Options are DEC, HEX, BIN or REAL. These modifiers will convert Item to DEC = Decimal, HEX = Hexadecimal, BIN = Binary digits or REAL = Value.

Item can be a constant or variable. There is no limit to the amount of items used other than program memory.

Explanation

The DEBUG command will send any values stored in a given variable or constant to the debug watch window. The DEBUG command is also linked with the IDE's In Circuit Debugger (Refer to the *In Circuit Debugger* section of the BasicMicro IDE users guide). Variables used by themselves are automatically truncated to character size.

Debugin

DEBUGIN [(Options) item]
Receives byte values from the IDE DEBUG window and stores them in a specified variable on the Atom.

Options are DEC, HEX, BIN or REAL. These modifiers will convert Item to DEC = Decimal, HEX = Hexadecimal, BIN = Binary digits or REAL = Value.

Item can only be a byte variable. It stores the received byte value in the specified variable.

Explanation

The DEBUGIN command allows you to send data to your program on-the-fly from the IDE DEBUG Watch Window and stores the data in the specified variable. This can be useful for adjusting a program on the fly.

Disable

DISABLE {intname}
Disable the specified interrupt. If no interrupt is specified disable all interrupts.

> **IntName** is the name of the interrupt to disable. See the ONINTERRUPT directive for a list of interrupt names.

Explanation

DISABLE turns off the spcified interrupt by clearing its interrupt enable bit. Any other register settings remain the same. If no interrupt is specified the global interrupt enable bit is cleared.

Do...While

Do
....user code....
While expression

Repeat a group of commands while expression is true

Expression is any combination of variables, constants, mathmatical and/or logic operators

Explanation

Execute a group of commands while some expression is true. DO...WHILE will run at least once. A True value is any value other than zero(0).

DTMFout

DTMFOUT pin,{playtime,pausetime,}[,key...]
Generate dual-tone, multifrequency tones for DTMF devices. The tones a touch tone telephone make are examples of DTMF tones.

Pin is an experssion of the I/O pin number to use. This pin will be set to an output during tone generation. After tone generation is complete, the pin is left as an input.

Playtime is an optional expression(0 to 65535) of the time to play the tone in milliseconds. If playtime is not used DTMFout defaults to 200 ms on.

Pausetime is an optionalexpression(0 to 65535) of the length of silence after each tone. If pausetime is not used DTMFout de faults to 50 ms. If playtime is used then pausetime is required.

Key is a variable or constant specifying the DTMF key to send.

Key #	Telephone Key
0 to 9	Digits 0 - 9
10	Digit *
11	Digit #
12—15	Fourth column tones A through D

Explanation

DTMF tones are a simple form of analog/digital conversion used to communicate digital commands via an analog signal. These signals are ussually used to dial a telephone. For all intents the DTMFOUT command acts as a telephone keypad. DTMF tones are generated using pulse width modulation to digitally create the equivilent of two sine wave waveforms at different frequencies. Due to the PWM generation of the tones high-frequency noise will be present on the output. This noise must be filtered with a lowpass filter circuit.

DTMFout2

DTMFOUT2 pin1 \ pin2,{playtime,pausetime,}[,key...]
Uses two pins to generate dual-tone, producing a cleaner signal (i.e., telephone "touch" tones).

Pin1 / **Pin2** are expressions for the I/O pins to use. These pins will be set as outputs during tone generation. After tone genera tion is complete, the pins are set to inputs.

Playtime is an optional expression(0 to 65535) of the time to play the tone in milliseconds. If playtime is not used DTMFout defaults to 200 ms on.

Pausetime is an optionalexpression(0 to 65535) of the length of silence after each tone. If pausetime is not used DTMFout de faults to 50 ms. If playtime is used then pausetime is required.

Key is a variable or constant specifying the DTMF key to send.

Tone #	Telephone Key
0 to 9	Digits 0 - 9
10	Digit *
11	Digit #
12—15	Fourth column tones A through D

Explanation

The DTMFOUT2 command follows the same basic format as DTMFOUT (refer to DTMFOUT), except it generates the multifre-quency tones on two pins. These two pins can be tied together using a 390 ohms resistor. The tones generated by DTMFOUT2 are made of square wave frequencies. This produces clearer and louder tones but some DTMF decoders may have trouble with it

Enable

EnABLE {intname}
Enable the specified interrupt. If no interrupt is specified enable all interrupts.

> **IntName** is the name of the interrupt to disable. See the ONINTERRUPT directive for a list of interrupt names.

Explanation

ENABLE turns on the spcified interrupt by setting its interrupt enable bit. Any other register settings remain the same. If no interrupt is specified the global interrupt enable bit is set

Enablehserial

ENABLEHSERIAL
Enable the hardware serial system(SCI3)

Explanation

ENABLEHSERIAL is a compile time directive that tells the compiler to add support for the hardware serial system(SCI3)

Enablehserial2 (ATOM-Pro Plus only)

ENABLEHSERIAL2
Enable the hardware serial system(SCI3_2)

Explanation

ENABLEHSERIAL2 is a compile time directive that tells the compiler to add support for the hardware serial system(SCI3_2). This directive is only supported by the ATOM-Pro Plus processors.

Enablehservo

ENABLEHSERVO pinmask,minpulse,maxpulse,resfresh
Enable the hardware servo control system

Pinmask is a constant that specifies a mask for which I/O pins will be taken over by the HSERVO system.

Minpulse is a constant of the minimum pulse width in useconds of the servo control pulses

Maxpulse is a constant of the maximum pulse width in useconds of the servo control pulses

Refresh is a constant of the refresh rate in milliseconds.

Explanation

ENABLEHSERVO is a compile time directive that tells the compiler to add support for the hardware servo control system. The HSERVO system uses the TimerW(or TimerZ0 in ATOM-Pro Plus) to produce interrupt driven signals for up to 32 servos. The pinmask is a 32bit binary number where each bit specifies whether that particular pin is handled by the hardware servo system or by user code. A bit set as logical 1 enables that pin for the hardware servo system.

Example:

```
ENABLEHSERVO  %11110000,200,2200,20
```

In the example pins P4-P7 are handled by the hardware servo system. The minimum pulse width is 200us and the maximum pulse width is 2200us giving a total swing of 2000us on the servo. The refresh rate of the servos is every 20ms.

End

END
Ends the program.

Explanation

END will stop the program until reset. All I/O lines will remain at their last know state.

Exception

EXCEPTION label

Clears the return stack and jumps to label.

Label is any root lable. A root label is any label not defined in a gosub routine.

Explanation

Exception's puprose is to jump out of nested subroutines(ie GOSUBs). Using a goto to jump out of a GOSUB will leave a return value on the stack and eventually may cause the stack to overflow. EXCEPTION works just like a GOTO except it clears any return values from the stack.

For...Next

FOR variable = startVal to endVal {STEP stepVal} ... NEXT

Create a repeating loop that executes the program lines between FOR and NEXT, increment or decrement the variable according to stepVal, until the value of the variable passes the endVal.

Variable is where to store the current count.

StartVal is an expression of the initial value of the variable.

EndVal is an expression of the end value of the variable. When the value of the variable passes endVal execution stops and the program goes to the instruction after Next.

StepVal is an optional expressionof the amount variable increases or decreases with each trip through the FOR/ NEXT loop. Negative values for StepVal will decrement and Positive values will increment.

For counter = 20 to 1 step -1 ; this will decrement -1

For counter = 1 to 20 step 1 ; this will increment +1

Explanation

The FOR...NEXT loop will allow your program to execute a series of instructions for a specific number of repetitions. By default, the counter variable is incremented by 1 each time through the loop. It will continue to loop until the result of the counter is outside of the range set by StartValue and EndValue.

Note: The variable type must match the Start,End and Step value types. If Floating Point numbers are used for Start,End and Step, variable MUST be a FLOAT type variable.

Freqout

FREQOUT pin, duration, freq1{,freq2}
Generates one or two tones for a specified duration.

Pin is an expression of the I/O pin number to use. This pin will be put into output mode during generation of tones and left in that state after the instruction is completed.

Duration is an expression of the length in milliseconds of the tone(s).

Freq1 is an expression of the frequency(Hz, 0 to 32767) in hertz of the first tone.

Freq2 is an expression of the frequency(0 to 32767 Hz) in hertz of the optional second tone

Explanation

FREQOUT generates one or two sine waves using a pulse-width modulation algorithm. The FREQOUT command can be used to play tones through a speaker or audio amplifier. FREQOUT can also be used to play simple songs. A filtering circuit is required with most speakers.

GetHSERVO

GETHSERVO pin,position{,idle}
Get the current position of the specified servo. Optionally get whether the servo is idle or not.

Pin is an expression of the I/O pin number to check servo position on.

Position is a variable where GETHSERVO will store the current position of the specified servo

Idle is an optional variable where GETHSERVO will store the current state of the servo. A value of $FFFFFFFF(ie Non-Zero) means the servo is idle(at it's final position)

Explanation

Gethservo is used to determine what position a servo is at and whether it has finished moving to it's final position.

Gosub...Return

GOSUB Label
Store the address after GOSUB, then go to the point in the program specified by Label.

Label specifies the section of the program to jump to.

Explanation

GOSUB is a close relative of the GOTO command. The GOSUB command tells the program to go execute code at the beginning of the specified label. Unlike GOTO, GOSUB stores the location of the next line of code immediately following itself, when the program encounters a RETURN instruction in the subroutine, it then tells the program to return to the stored location.

When GOSUBs are used, a RETURN statement is necessary (at the end of the subroutine) to take the program back to the instruction after the most recent GOSUB.

Important Notes

Each GOSUB call uses 4 bytes of ram from the STACK to store the return address.

Goto

GOTO Label
Go to the point in the program specified by Label.

Label specifies the section of the program to jump to.

Explanation

The GOTO command makes a program jump to a specific label and execute the code that starts at that location. BASIC programs are read from left to right / top to bottom, just like in the English language. The GOTO command forces the program to jump to another section of code.

High

HIGH pin
Makes the specified pin an output and sets it to high
(+5 volts is High)

Pin is an expression of the I/O pin number to use.

Explanation

The HIGH command is used to set the designated pin to an output and to +5 volts. This allows your program to easily turn on an LED or other such devices.

HPWM

HPWM pin,period,duty
Output a pulse width modulated signal at the specified period and duty cycle

Pin is an expression of the I/O pin number to use.
This pin will be placed into output mode during pulse generation then switched to input mode when the instruction finishes. Only pins with the FTIOB/C or D option can use the HPWM command.

Period is an expression of the period of the pulse width signal in *us.*

Duty is an expression of the duty cyle of the pulse width signal in *microseconds(us)*.

Explanation

The HPWM command outputs a user specified Pulse signal. The period is the time in *us* of one pulse cycle. The duty is the time in *us* that the pulse signal is high.

HSERIN

HSERIN {timeout,tlabel,}[{mods} Var...VarN]
Read data from the hardware serial port

Timeout is an expression of the time in millisecons to wait for data to be recieved.

TLabel a label to jump to when HSERIN times out.

Mods are command modifiers which can be used to modify the variable directly.

Var...VARN is a variable or list of variables(comma delimited) where data will be stored.

Explanation

The HSERIN command is part of the hardware serial port system. In order for it to work properly an ENABLEHSERIAL compiler directive must be in the program and a SETHSERIAL command must setup the hardware serial port. HSERIN works almost identically to the SERIN command except it must use the hardware serial input pin(RXD) and the baudrate is set by the SETHSERIAL command.

HSERIN2 (ATOM-Pro Plus only)

HSERIN2 {timeout,tlabel,}[{mods} Var...VarN]
Read data from the hardware serial port

Timeout is an expression of the time in millisecons to wait for data to be recieved.

TLabel a label to jump to when HSERIN times out.

Mods are command modifiers which can be used to modify the variable directly.

Var...VARN is a variable or list of variables(comma delimited) where data will be stored.

Explanation

The HSERIN2 command is part of the hardware serial port system(BasicATOM-Pro Plus only). In order for it to work properly an ENABLEHSERIAL2 compiler directive must be in the program and a SETHSERIAL2 command must setup the hardware serial port. HSERIN2 works almost identically to the SERIN command except it must use the hardware serial input pin(RXD_2) and the baudrate is set by the SETHSERIAL2 command.

HSEROUT

HSEROUT [{mods} Exp...ExpN]
Read data from the hardware serial port

Mods are command modifiers which can be used to modify the variable directly.

Exp...ExpN is an expression or list of expressions(comma delimited) of data that will be sent.

Explanation

The HSEROUT command is part of the hardware serial port system(BasicATOM-Pro Plus only). In order for it to work properly an ENABLEHSERIAL directive must be in the program and a SETHSERIAL command must setup the hardware serial port. HSEROUT works almost identically to the SEROUT command except it must use the hardware serial output pin(TXD) and the baudrate is set by the SETHSERIAL command.

HSEROUT2 (ATOM-Pro Plus only)

HSEROUT2 [{mods} Exp...ExpN]
Read data from the hardware serial port

Mods are command modifiers which can be used to modify the variable directly.

Exp...ExpN is an expression or list of expressions(comma delimited) of data that will be sent.

Explanation

The HSEROUT2 command is part of the hardware serial port system(BasicATOM-Pro Plus only). In order for it to work properly an ENABLEHSERIAL2 directive must be in the program and a SETHSERIAL2 command must setup the hardware serial port. HSEROUT2 works almost identically to the SEROUT command except it must use the hardware serial output pin(TXD_2) and the baudrate is set by the SETHSERIAL2 command.

HSERVO

HSERVO [Pin\Pos\Spd....PinN\PosN\SpdN]
Read data from the hardware serial port

Pin...PinN are expressions of the pin numbers of the servos whose position and speed are to be set.

Pos...PosN are expressions of the positions to set the specified servos to.

Spd...SpdN are optional expressions of the speed to move each servo to its new position(defaults to 255 if not used).

Explanation

The HSERVO command is a back ground timer interrupt driven command. It allows you to set the position and speed to move to that new position of upto 32 servos at one time. Each severo set will start moving to its new position imediately after the HSERVO comand finishes. HSERVO requires an ENABLEHSERVO directive in your program.

Note: The hardware servo command can affect timing critical commands such as pause and serial commands(not HSERIAL commands). See the ENABLEHSERVO directive for details on calculating processor usage.

I2Cin

I2CIN DataPin, ClockPin,{ErrLabel,}Control,{Address,}
[{mods}Var,...VarN]
Receives data from an I2C device (EEPROM, External A/D Converter)

DataPin is an expression of the I/O pin number to use for data(SDA).

ClockPin is an expression of the pin number that the BasicATOM will use to clock the bus signal. (SCL)

ErrLabel is a label that the program will jump to if the I2CIN command fails (i.e.: device is not connected).

Control is an expression of the I2C device's control byte. The control byte consist of two parts. The first four bits are the device type (EEPROMs use %1010). The next three bits are the device ID. If the address lines of the serial EEPROM (i.e. : A0,A1, A2) are grounded then the next three bits of the control byte must be zero.(ie: %1010000X). The last bit is a flag used by the ATOM-Pro to determine the addressing format, 1 =16bit ad dressing, 0 = 8bit addressing for I2C communications.

Address is an optional expression of the starting address for reading from the device.

Mods are command modifiers which can be used to modify the variable data directly after being recieved.

Var is a variable where data being recieved from the device will be stored

VarN is a list of variables where the data being recieved from the device will be stored

Explanation

The I2CIN command allows your program to receive data from an I2C device.

I2Cout

I2COUT DataPin, ClockPin,{ErrLabel,}Control,{Address,} [{mods} Exp,...ExpN]
Sends data to an I2C device (EEPROM, External A/D Converter)

DataPin is an expression of the I/O pin number to use for data(SDA).

ClockPin is an expression of the pin number that the BasicATOM will use to clock the bus signal. (SCL)

ErrLabel is a label that the program will jump to if the I2CIN command fails (i.e.: device is not connected).

Control is an expression of the I2C device's control byte. The control byte consist of two parts. The first four bits are the device type (EEPROMs use %1010). The next three bits are the device ID. If the address lines of the serial EEPROM (i.e. : A0,A1, A2) are grounded then the next three bits of the control byte must be zero.(ie: %1010000X). The last bit is a flag used by the ATOM-Pro to determine the addressing format, 1 =16bit addressing, 0 = 8bit addressing for I2C communications.

Address is an optional expression of the starting address for writing to the device.

Mods are command modifiers which can be used to modify the variable data directly before being sent.

Exp is an expression of the data being sent.

ExpN is a list of expressions of the data being sent.

Explanation

The I2COUT command allows your program to write data to an I2C device.

If...Then...Elseif...Else...Endif

IF *Compare* THEN {Gosub} *Label*
Compare, if true(not 0) jump to label or:

IF *Compare* THEN

Statements...

ELSEIF *Compare*

Statements...

Else

Statements...

Endif

The IF...THEN...ELSEIF...ELSE...ENDIF evaluates one or more conditions and, if true, jumps to a label. If false then skip next function

Condition is a statement, such as "x = 7" that can be evaluated as true or false.

Gosub is optional. Using GOSUB allows your program to return to the next line of your program after running a subroutine.

Label is a label that specifies where to go in the event that the condition is true.

Explanation

The If...Then command is a decision maker of sorts. There are two ways in which If...Then can be used. The first tests a condition and, if that condition is true, jumps to a point in the program specified by an address label. The condition that IF...THEN tests is written as a mixture of comparison and logic operators. The comparison operators are:

= equal	< less than
<> not equa l	>= greater than or equal to
> greater than	<= less than or equal to

The second use of the If...Then can conditionally execute a group of statements following the THEN. The statements must be followed by Elseif or Else with an Endif.

Input

INPUT pin
Makes the specified pin an input

Pin is a variable or constant that specifies the I/O pin to use.

Explanation

There are several ways to make a pin an input. When a program begins, all of the pins should be inputs. Input instructions PULSIN, SERIN will automatically change the specified pin to input and leave it in that state.

Lcdinit

LCDINIT RegSel\CLK\DB7\DB6\DB5\DB4,RdWrPin
Initilize the LCD display.

RegSel can be a constant or variable specifying the pin for the "R/S" liine of the LCD.

CLK can be a constant or variable specifying the pin for the "E" line of the LCD.

DB7-DB4 can be constants or variables specifying the data lines of the LCD.

RdWrPin can be a constant or variable specifying the pin for the "R/W" line of the LCD.

Lcdread

LCDREAD RSel\CLK\DB7\D6\D5\D4,RdWrPin, Address, [{mods} Var]
Reads the RAM on a LCD module using the Hitachi 44780 controller or equivalent.

RSel is an expression of the pin number to use for the RegSel (R/S line) of the LCD

CLK is an expression of the pin number to use the clock(E) line of the LCD

D7-D4 are expressions of the 4 pin numbers used for the LCD data line

RdWrPin is an optional expression of the pin number used to connect to the RdWr line of the LCD

Address is an expression of the first address location of RAM you are trying to read. Address from 0 to 127 return the current character in the display memory. Address 128 and above return Character RAM values.

Mods are command modifiers which can be used to modify the variable directly.(See Command Modifiers)

Var is the variable where the value returned will be stored.

Note

When using the LCDREAD command you will need to first initialize the LCD screen. See LCDINIT.

Lcdwrite

LCDWRITE RSel\CLK\D7\D6\D5\D4,{RdWrPin,} [{mods} Exp]
Sends Text to an LCD module using an Hitachi 44780 controller or equivalent.

RSel is an expression of the pin number to be connected to the LCD RegSel(R/S) line.

CLK is an expression of the pin number ot be connected to the clock(E) line of the LCD.

D7-D4 are expressions of the 4 pin numbers to connect to the LCD data lines.

RdWrPin is an option expression of the pin number to connect to the RdWr pin of the LCD.

Mods are command modifiers which can be used to modify the expression directly.

Exp can be a constant or variable that is the data to be written.

Note

When using the LCDWRITE command you will need to first initialize the LCD screen. See LCDINIT.

LcdWrite Comand Table

There are several control commands that can be used with LCDWRITE such as CLEAR and HOME. Each additional control command used with LCDWRITE must be separated with a "," (comma) inside of the brackets "[...]". Below is a chart of all the available control commands for use with LCDWRITE.

Command Name		Description
$101	CLEAR	Clear Display
$102	HOME	Return Home
$104	INCCUR	Auto Increment Cursor(default)
$105	INCSCR	Auto Increment Display
$106	DECCUR	Auto Decrement Cursor
$107	DECSCR	Auto Decrement Display
$108	OFF	Display,Cursor, and Blink off
$10C	SCR	Display on,†Cursor and Blink off
$10D	SCRBLK	Display and Blink on, Cursor off
$10E	SCRCUR	Display and Cursor on, Blink off
$10F	SCRCURBLK	Display, Cursor, and Blink on
$110	CURLEFT	Move Cursor left
$114	CURRIGHT	Move cursor right
$118	SCRLEFT	Move Display left
$11C	SCRRIGHT	Move Display right
$120	ONELINE	Set display for 1 line LCDs
$128	TWOLINE	Set display for 2 line LCDs
$140	CGRAM | *address*	Set CGRAM *address* for R/W
$180	SCRRAM | *address*	Set Display ram *address* for R/W

Let

LET Var = {mods} expression
Assign a value to a variable

Var is the variable to store the data in.

Expression is any type of expression.

Explanation

LET is an optional command; as an example Temp=2 is the same as LET Temp=2. The LET command is often used to make programming code more human readable.

Enhancements

The LET command also supports loading a list of values into arrays of variables. For example:

```
myarray var byte(50)
temp var byte

temp = 10
myarray = 1,2,temp,4,5,6,7,8*temp,9,"Hello world"
```

As you can see from the example the list can include variables, expressions and strings.

Important Note

The LET command also supports modifiers. (See Command Modifiers)

Lookdown

LOOKDOWN value,{comparisonOp,}[value0,
value1,...valueN],resultVariable

Compare a value to a list of values according to the relationship specified by the comparison operator. Store the index number of the first value that makes the comparison true into resultVariable. If no value in the list makes the comparison true, resultVariable is unaffected.

Value is an expression to be compared to the values in the list.

ComparisonOp is optional and maybe one of the following:

= equal	< less than
<> not equal	>= greater than or equal to
> greater than	<= less than or equal to

If no comparison operator is specified, Then it defaults to equal (=).

Value0, value1... a list of expressions.

ResultVariable is a variable in which the index number will be stored if a true comparison is found.

Explanation

LOOKDOWN searches values in a list, and stores the item number of the first match in a variable. In other words, Lookdown compares the user value to values in a list, the first comparison that is true, will return the value of the index position the match was found. The index list starts at 0 not 1.

Lookup

LOOKUP index, [value0, value1,...valueN], resultVariable

Get the value specified by the index and store it in a variable. If the index exceeds the highest index value of the items in the list, variable is unaffected.

Index an expression of the item number of the value to be retrieved from the list of values.

Value0, value1... a list of expressions.

ResultVariable is a variable in which the retrieved value will be stored.

Explanation

Lookup could be considered the opposite of Lookdown. Lookup returns an item from a list based on the item's position in the list. Positions start at 0.

Low

LOW pin
Make the specified pin output a low signal.

Pin is an expression of the I/O pin number to use.

Explanation

The LOW command will make the specified pin low (0 Volts), which will also make the specified pin an output.

Nap

NAP period
Enter sleep mode for the specified period. Power consumption is reduced while sleeping.

Period is an expression of the time in multiples of 2 millisecond periods of sleep

Explanation

NAP is similar to Sleep, in that it runs the processor's internal sleep system, however unlike the Sleep command, nap has no special options. All sleeping time is 2ms * period.

OnInterrupt

ONINTERRUPT *intname,label*
Set a label to jump to when the specified interrupt occures

IntName is the name of the interrupt(See table below)

Label is the name of the label to jump to when the interrupt occures

Interrupts

IRQ0INT	Irq0 pin interrupt
IRQ1INT	Irq1 pin interrupt
IRQ2INT	Irq2 pin interrupt
IRQ3INT	Irq3 pin interrupt

WKPINT_0	WKP0 pin onchange interrupt
WKPINT_1	WKP1 pin onchange interrupt
WKPINT_2	WKP2 pin onchange interrupt
WKPINT_3	WKP3 pin onchange interrupt
WKPINT_4	WKP4 pin onchange interrupt
WKPINT_5	WKP5 pin onchange interrupt

TIMERVINT_OVF	TimerV overflow interrupt
TIMERVINT_CMEB	TimerV compare match A int
TIMERVINT_CMEA	TimerV compare match B int

SCI3INT_TDRE	Transmit Data Register Empty interrupt
SCI3INT_RDRF	Read Data Register Full interrupt
SCI3INT_TEND	Transmit End interrupt
SCI3INT_OER	Overflow Error interrupt
SCI3INT_FER	Frame Error interrupt
SCI3INT_PER	Parity Error interrupt

IICINT	I2C interrupt

ADINT	Analog conversion complete int

HSERIALINT_TDRE	Transmit Data Register Empty interrupt
HSERIALINT_RDRF	Read Data Register Full interrupt
HSERIALINT_TEND	Transmit End interrupt
HSERIALINT_OER	Overflow Error interrupt
HSERIALINT_FER	Frame Error interrupt
HSERIALINT_PER	Parity Error interrupt

HSERVOINT_IDLE	Any Servo Idle interrupt
HSERVOINT_IDLE0-31	# Servo Idle interrupt
HSERVOINT_USER	HServo User Interrupt
HSERVOINT	Hservo Interrupt

ATOM-Pro only(H8/3664/3694)

TIMERAINT	Overflow interrupt
TIMERWINT_OVF	Overflow interrupt
TIMERWINT_IMIEA	Capture/Compare Match A int
TIMERWINT_IMIEB	Capture/Compare Match B int
TIMERWINT_IMIEC	Capture/Compare Match C int
TIMERWINT_IMIED	Capture/Compare Match D int

ATOM-Pro Plus only(H8/3687)

RTCINT	Real time clock interrupt(ATOM-Pro Plus only)
TIMERZ0INT_OVF	Overflow interrupt
TIMERZ0INT_IMIEA	Capture/Compare Match A int
TIMERZ0INT_IMIEB	Capture/Compare Match B int
TIMERZ0INT_IMIEC	Capture/Compare Match C int
TIMERZ0INT_IMIED	Capture/Compare Match D int
TIMERZ1INT_UDF	Underflow interrupt
TIMERZ1INT_OVF	Overflow interrupt
TIMERZ1INT_IMIEA	Capture/Compare Match A int
TIMERZ1INT_IMIEB	Capture/Compare Match B int
TIMERZ1INT_IMIEC	Capture/Compare Match C int
TIMERZ1INT_IMIED	Capture/Compare Match D int
TIMERB1INT	Overflow interrupt
SCI3_2INT_TDRE	Transmit Data Register Empty interrupt
SCI3_2INT_RDRF	Read Data Register Full interrupt
SCI3_2INT_TEND	Transmit End interrupt
SCI3_2INT_OER	Overflow Error interrupt
SCI3_2INT_FER	Frame Error interrupt
SCI3_2INT_PER	Parity Error interrupt
HSERIAL2INT_TDRE	Transmit Data Register Empty interrupt
HSERIAL2INT_RDRF	Read Data Register Full interrupt
HSERIAL2INT_TEND	Transmit End interrupt
HSERIAL2INT_OER	Overflow Error interrupt
HSERIAL2INT_FER	Frame Error interrupt
HSERIAL2INT_PER	Parity Error interrupt

Explanation

The OnInterrupt directive is a compile time action. It sets, at compile time, the label that the specified interrupt will jump to when that interrupt occurs. The OnInterrupt command does NOT enable an interrupt or setup any registers specific to the specified interrupt.

All interrupts except for SCI3(and SCI3_2), IICINT and ADINT clear their interrupt flags when they jump to the label specified in the ONINTERRUPT directive.

See Enable,Disable

Output

OUTPUT pin
Makes the specified pin an output

Pin is an expression of the I/O pin number to use.

Explanation

The OUTPUT command allows your program to directly affect the direction of the specified pin.

OWIN

OWIN Pin,Mode,{NCLabel,} [{Mods} Var]
Protocol used to communicate to 1-wire devices.

Pin is an expression of the I/O pin number to use for the One wire command.

Mode is an expression indicating the mode of data transfer. Mode controls placement of reset pulses, detection of presence pulses, byte / bit input and normal / high speed transmission. The proper value for Mode will depend on the 1-wire device used. Consult the device data sheet to determine the correct Mode. See chart below:

Mode	Setting
0	No Reset, Byte mode, Low speed
1	Reset before data, Byte mode, Low speed
2	Reset after data, Byte mode, Low speed
3	Reset before and after data, Byte mode, Low speed
4	No Reset, Bit mode, Low speed
5	Reset before data, Bit mode, Low speed

NCLabel is a label the program can jump to if a connection failure occurs with the OWIN command (ie. No chip present).

Mods are command modifiers which can be used to modify the variable directly.

Var is the variable or variable array where the value(s) returned will be stored.

Explanation

The 1-wire protocol was developed by Dallas Semiconductor as an asynchronous serial communication format. It uses one I/O pin as a common bi-direction serial data bus..

The 1-Wire protocol synchronizes the slave devices to the master. The master initiates and controls all activities on the 1-Wire bus. 1-Wire uses CMOS/TTL logic levels. A resistor connects the data line of the 1-Wire bus to the 5V supply of the bus master.

OWOUT

OWOUT Pin,Mode,{NCLabel,} [{Mods} Exp]
Protocol used to communicate to 1-wire devices.

Pin is an expression of the I/O pin number used for the One wire command.

Mode is an expression of the mode of data transfer. Mode controls placement of reset pulses, detection of presence pulses, byte / bit input and normal / high speed. The proper value for Mode will depend on the 1-wire device used. Consult the device data sheet to determine the correct Mode. See chart below:

Mode	Setting
0	No Reset, Byte mode, Low speed
1	Reset before data, Byte mode, Low speed
2	Reset after data, Byte mode, Low speed
3	Reset before and after data, Byte mode, Low speed
4	No Reset, Bit mode, Low speed
5	Reset before data, Bit mode, Low speed

NCLabel is a label the program can jump to if a connection failure occurs with the OWOUT command (ie. No chip present).

Mods are command modifiers which can be used to modify the variable directly.

Exp is an expression of the data to be sent.

Explanation

The 1-wire protocol was developed by Dallas Semiconductor as an asynchronous serial communication format. It uses one I/O pin as a common bi-direction serial data bus. The OWout and OWin commands are tightly integrated. In most cases you will need both to talk to any 1-wire part.

Pause

PAUSE milliseconds
Wait(stop) for the specified number of milliseconds.

Milliseconds is an expression of the length of the pause in ms. Milliseconds may be any length up to a 32 bit number

Explanation

The PAUSE command delays the execution of the program for the specified number of milliseconds.

Pauseclk

PAUSECLK cycles
Pause the program (do nothing) for the specified number clock cycles.

Cycles is an expression of number of clock cycles to pause.

Explanation

The PAUSECLK command delays the execution of the program for the specified number of clock cycles. Since each oscillator has a variances of 0.05% you will need to determine your own timings. This is only necessary if precision timing is required.

Pauseus

PAUSEUS halfmicroseconds
Pause the program (do nothing) for the specified number of micro seconds.

Halfmicroseconds is an expression of the number of .5us increments to pause

Explanation

The PAUSEUS command delays the execution of the program for the specified number of halfmicroseconds.

PEEK...POKE

PEEK address, variable
POKE address, expression

Read/Write specified RAM location

Address is an expression of the address in memory to read/write to.

Variable is the variable where the results will be stored or where where the value to be written is stored..

Expression is an expression

Explanation

PEEK and POKE are considered advance commands and should only be used by experienced users. The explanation of these commands are kept short intentionally. Use of the PEEK command allows a specific address to be read and store its value in the assigned *variable .* The PEEK and POKE commands allow direct access to all of the registers of the ATOMs H8/3664 processor.

Note The address value actually points to the address+ 0xF780. Ram begins in the H8/3664 at 0xF780. The PEEK and POKE commands automatically offset this address.

Pulsin

PULSIN pin, state, {TimeoutLabel,Timeout,} Var
Measure the width of a pulse.

Pin is an expression of the I/O pin number to use. This pin will be placed into input mode during pulse measurement and left in that state after the instruction finishes.

State is an expression(0 or 1) of the trigger state. 0 specifies a (0-to-1) transition. 1 specifies a 1-to-0 transition (0).

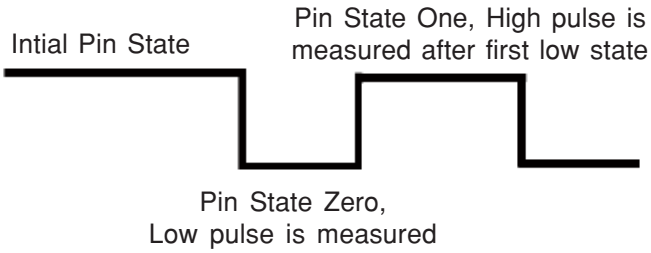
TimeoutLabel is an optional label that specifies where to go if a time out occurs. The default time out value is 65,535 microseconds.

Timeout is an expression of the amount of time to wait(in us) before timing out. Timeout must be used with TimeoutLabel

Var is a variable in which the pulse duration will be stored.

Explanation

PULSIN will measure the pulse width on a specified pin. If the state is zero, the width of a low pulse is measured. If the state is one, the width of a high pulse is measured. The measured width is then placed in Var. If the pulse edge never happens or the width pulse is too great to measure Var will default to 0. Pulsin will timeout after 65,535 microseconds if the optional timeout label is not used.



PULSIN will return the pulse width in µs.

Pulsout

PULSOUT pin, time
Output a pulse.

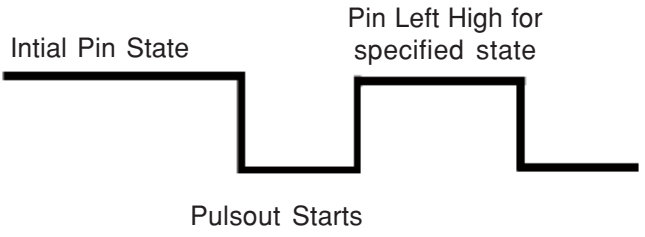
Pin is an expression of the pin number to use. This pin will be placed into output mode immediately before the pulse and left in that state after the instruction finishes.

Time is an expression of the duration of the pulse in .5µs increments.

Explanation

PULSOUT will generate a pulse on the specified pin for the given period. The pulse is generated by toggling the pin state twice. The initial state of the pin will determine the polarity of the pulse. The pin specified to generate the pulse is automatically made an output.

PULSOUT will generate a pulse with a period in 1 µs increments. The minimum pulse width is 4 µs. You can not go below this value.



Push...Pop

PUSH value
POP variable

Value is any value upto 32bits long

Variable can be any sized variable.

PUSH and POP are used to store a value on the stack and retrieve a value from the stack.

Explanation

PUSH and POP are convientient ways of saving and storing temporary values without using extra defined variables. Push and Pop are safe to use between gosubs.

Pwm

PWM pin, period, duty, cycles
Convert a digital value to analog output via pulse-width modulation.

Pin is an expression of the pin to use.
This pin will be placed into output mode during pulse generation.

Period is an expression of the period of the pulse width signal in *us.*

Duty is an expression of the duty of the pulse width signal in *us*.

Cycles is an expression of the number of pulses to output..

Explanation

The PWM command outputs a user specified Pulse signal. The period is the time in *us* of one pulse cycle. The duty is the time in *us* that the pulse signal is high. The PWM command is software based so it has the same limitations as any other software command. If you need to output a PWM signal constantly and still be able to run other commands see the HPWM command.

RCtime

RCTIME pin, state, {TimeoutLabel,TimeoutMultiple,}, resultVariable
Count time while pin remains in state—usually to measure the charge or discharge time of a resistor and capacitor circuit. (RC)

Pin is am expression of the I/O pin number to use.
This pin will be placed into input mode and left in that state when the instruction finishes.

State is an expression(1 or 0) of the state that will end the RCTIME period.

TimeoutLabel is an optional label that specifies where to go if a time out occurs. The default time out value is 65,535 microseconds.

Timeout is an expression of the amount of time to wait before timing out.

ResultVariable is a variable in which the time measurement will be stored.

Explanation

RCTIME can be used to measure the charge / discharge time of a resistor and capacitor circuit (RC). RCTIME can also be used as a fast stopwatch for recording events of very short duration. This allows measuring resistance or capacitance using R or C sensors (i.e. thermistors or capacitive humidity sensors); or respond to user input through a potentiometer. (Typically 5k to 50k pot.)

Read

READ address,variable
Read a value from the builin eeprom(Only available on some AtomPro modules)

Address is an expression of the address in the eeprom to read.

Variable any variable type may be used. However, thee eeprom will always return an 8bit value which may be truncated if you use a smaller variable type.

Explanation

READ can be used to get previously stored 8bit values from the onboard EEPROM of supported ATOM-Pro modules. The ATOM-Pro28 and 40 pin modules have 4kbyte eeproms The ATOM-Pro 24 pin module does not have an onboard eeprom. An external I2C eeprom may be connected to p10(SCL)/p11(SDA) to use the READ/WRITE commands. The ATOM-Pro ARC and Mini-ARC can have an external I2C eeprom added to their 8pin I2C socket to support the READ/WRITE commands.

ReadDM

READDM address,[{Mods}variable]
Read a value from the builin eeprom(Only available on some AtomPro modules)

Address is an expression of the addres to start reading at.

Variable any variable type may be used. However, the eeprom will always return an 8bit value which may be truncated if you use a smaller variable type.

Explanation

READDM can be used to get previously stored 8bit values from the onboard EEPROM of supported AtomPro modules. ReadDM supports command modifiers. The ATOM-Pro28 and 40 pin modules have 4kbyte eeproms The ATOM-Pro 24 pin module does not have an onboard eeprom. An external I2C eeprom may be connected to p10(SCL)/p11(SDA) to use the READ/WRITE commands. The ATOM-Pro ARC and Mini-ARC can have an external I2C eeprom added to their 8pin I2C socket to support the READ/WRITE commands.

Repeat...Until

REPEAT
....code....
UNTIL expression

Expression is an expression

Explanation

Repeat a group of commands until some expression is true. True being any value other than 0.

Resume

RESUME
Specialized RETURN command for interrupts

no arguments

Explanation

When exiting from an interrupt the resume command must be used.

Reverse

REVERSE pin
Reverse the data direction of the specified pin.

Pin is an expression of the I/O pin number to use. This pin will be placed into the opposite of its current input/output (I/O) mode.

Explanation

Reverse is a convenient way to switch the I/O direction of a pin. If a pin is set as an input, the REVERSE command, will change it to an output.

Serdetect

Serdetect pin,mode,var
Detect incoming baud rate. Used for auto detecting baud rates.

Pin is an expression of the I/O pin number that will be used to receive the sync character.

Mode is the settings for Bits 13,14 and 15 of the BAUDMODE. Bit 13 is a flag that controls the number of data bits and parity (0=8 bits and no parity, 1=7 bits and even parity). Bit 14 controls polarity(0=noninverted, 1=inverted). Bit 15 is not used by SERIN. Constants from the below table can be used for Mode:

IMODE	= Inverted
NMODE	= Non Inverted
IEMODE	= Inverted, Even Parity
NEMODE	= Non Inverted, Even Parity
IOMODE	= Inverted, Open Drain
NOMODE	= Non Inverted, Open Drain
IEOMODE	= Inverted, Even Parity, Open Drain
NEOMODE	= Non Inverted, Even Parity, Open Drain

Var is a word sized variable that will hold the calculated baudmode value which can be used by serin and serout.

Explanation

Serdect is used to auto detect an incoming baud rate. This is ideal for applications or products that can be used at multiple baud rates and be software switched. Serdetect can take the place of hard wired jumpers or switches for changing baud rates.

In order for serdetect to calculate the bitrate a character must be received. For inverted mode the binary value of the character to send for calculating bitrate must be %XXXXXX01. For non inverted modes the character must be %XXXXX101.(X = don't care)

Serin

SERIN recieve{\flow},baudmode,{plabel,}{timeout,tlabel,}[inputData]
Receive asynchronous (e.g., RS-232) data.

Recieve is an expression of the I/O pin number to recieve data through.

Flow is an optional expression of the I/O pin to be used for flow control. This pin will switch to output mode and remain in that state after the end of the instruction.

Baudmode is a 16-bit expression of the serial timing and configuration. The lower 13 bits are the bit period. Bit 13 ($2000 hex) is a flag that controls the number of data bits and parity (0=8 bits and no parity, 1=7 bits and even parity). Bit 14 ($4000 hex) controls polarity (0=noninverted, 1=inverted). Bit 15 ($8000 hex) is not used by SERIN.

Plabel is an optional label where the program will jump to in the event of a parity error.. This argument may only be provided if baud mode indicates 7 bits, and even parity, otherwise the label is ignored.

Timeout is an optional expression that tells SERIN how long, in milliseconds, to wait for incoming data. If data does not arrive in time, the program will jump to the address specified by Tlabel.

Tlabel is an optional label which must be provided along with Timeout, indicating where the program should go in the event that data does not arrive within the period specified by Timeout.

InputData is a list of variables and modifiers that tells SERIN what to do with incoming data. SERIN can store data in a variable or array; interpret numeric text (decimal, binary, or hex), and store the corresponding value in a variable; wait for a fixed or variable sequence of bytes; or ignore a specified number of bytes. These actions can be combined in any order in the inputData list.

SERIN Modes

Inverted?	*Parity?*	*Baud Rate*	*Constant*
No	No	300	N300
Yes	No	300	I300
No	Yes	300	NE300
Yes	Yes	300	IE300
No	No	1200	N1200
Yes	No	1200	I1200
No	Yes	1200	NE1200
Yes	Yes	1200	IE1200
No	No	...	*N...
Yes	No	...	*I...
No	Yes	...	*NE...
Yes	Yes	...	*IE...

* 2400, 4800, 9600, 14400, 19200, 28800, 33600, 38400, 57600

Explanation

One of the most used forms of communication between electronic devices is serial communication. The two types of serial communication are asynchronous and synchronous. The SERIN and SEROUT commands use an asynchronous method to receive and send serial data. The term asynchronous means “no clock.” Data is transmitted and received without the use of a separate “clock” wire. The PC’s serial ports (COM ports, RS-232) use asynchronous serial communication.

Improtant Note

1. There are several modifiers for use with the SERIN / SEROUT commands. Refer to Command Modifier section of this manual.
2. There are many predefined serial baudmodes. See reserved words for whole list.

Serout

SEROUT transmit,baudmode,{pace,}[outputData]
SEROUT transmit\flow,baudmode,{timeout,tlabel,}[outputData]
Transmit asynchronous (e.g., RS-232) data.

Transmit is an expression of the I/O pin number to send data through.

Baudmode is a 16-bit expression of the serial timing and configuration. The lower 13 bits are the bit period. Bit 13 ($2000 hex) is a flag that controls the number of data bits and parity (0=8 bits and no parity, 1=7 bits and even parity). Bit 14 ($4000 hex) controls polarity (0=noninverted, 1=inverted). Bit 15 ($8000 hex) determines whether the pin is driven to both states (0/1) or to one state and open in the other (0=both driven, 1=open).

Pace is an optional expression of the time in milliseconds it should pause between transmitting bytes.

OutputData is a list of expressions of the outgoing data. SEROUT can transmit individual or repeating bytes; convert values into decimal, hex or binary text representations; or trans mit strings of bytes from variable arrays.

Flow is an optional expression of the I/O pin to use for flow control(byte-by-byte handshaking). This pin will switch to input mode and remain in that state after the instruction is completed.

Timeout is an optional expression of how long in milliseconds to wait for permission to send. If permission does not arrive in time, the program will continue at tlabel. Flow control must be used with Timeout.

Tlabel is an optional label used with flow control and timeout. Tlabel indicates where the program should go in the event that permission to transmit data is not granted within the period specified by the Timeout command. Tlabel must be used with Timeout and flowcontrol.

SEROUT Modes

Driven?	*Inverted?*	*Parity?*	*Baud Rate*	*Constant*
Yes	No	No	300	N300
Yes	Yes	No	300	I300
Yes	No	Yes	300	NE300
Yes	Yes	Yes	300	IE300
No	No	No	300	NO300
No	Yes	No	300	IO300
No	No	Yes	300	NEO300
No	Yes	Yes	300	IEO300
Yes	No	No	...	N...
Yes	Yes	No	...	I...
Yes	No	Yes	...	NE...
Yes	Yes	Yes	...	IE...
No	No	No	...	NO...
No	Yes	No	...	IO...
No	No	Yes	...	NEO...
No	Yes	Yes	...	IEO...

* 1200, 2400, 4800, 9600, 14400, 19200, 28800, 33600, 38400, 57600

Table 2-2 lists the predefined Baudmode constants available in MBasic. As you can see from the table there are several different baudmodes for each actual baud rate. The following describes each baudmode modifier:

N Normal (not inverted) signal
I Inverted signal
E Even Parity(otherwise no parity)
O Open drain(otherwise both high and low are driven)

Table 2-3 lists the command modifiers for the Output data

Explanation

One of the most used forms of communication between electronic devices is serial communication. The two types of serial communication are asynchronous and synchronous. The SERIN and SEROUT commands use an asynchronous method to receive and send serial data. The term asynchronous means “no clock.” Data is transmitted and received without the use of a separate “clock” wire. The PC’s serial ports (COM ports, RS-232) use asynchronous serial communication.

Servo

SERVO pin, rotation{, repeat}

Pin is an expression of the the pin number to control the servo

Rotation is an expression of the position you want the servo to rotate to. A value from -2400 to + 2400 is used with 0 being center. The maximum +2400 and minimum -2400 will vary based on the servo being used. Take caution not to exceed these values.

Repeat (optional)Specifies the number of internal cycles the command runs(defaults to 20).

Explanation

The SERVO command automatically handles all servo pulse singal calculations for you. SERVO is a foreground task.

Multiple Servo

MSERVO pin,
servo1{\servo2{\servo3{\servo4{\servo5{\servo6{\servo7{\servo8
{\servo9{\servo10{\servo11{\servo12}}}}}}}}}}{, repeat}

Pin is an expression of the pin number controlling servo1. All other servos are controlled by the next pin(ie servo2,pin+1,servo3,pin+2 etc...)

Servo# is an expression of the position you want the servo to rotate to. A value from -2400 to +2400 is used with 0 being center. The maximum +2400 and minimum -2400 will vary based on the servo being used. Take caution not to exceed these values.

Repeat (optional)Specifies the number of internal cycles the command runs(defaults to 20).

Explanation

The MSERVO command automatically handles all servo pulse singal calculations for you. MSERVO is a foreground task.

SetHserial

SETHSERIAL baudrate, databits, parity, stopbits

Baudrate is any one of the constants below.

H300	H26400
H600	H28800
H1200	H31200
H2400	H33600
H4800	H36000
H7200	H38400
H9600	H57600
H12000	H62500
H14400	H125000
H16800	H250000
H19200	H312500
H21600	H500000
H24000	

Databits can be either of the constants below.

H8DATABITS
H7DATABITS

Parity can be any one of the constants below.

HNOPARITY
HEVENPARITY
HODDPARITY

Stopbits can be either of the constants below.

H1STOPBITS
H2STOPBITS

Explanation

The SETHSERIAL command sets the HSERIAL system baudrate and modes. This command must be used before using a HSERIN/ HSEROUT command. The ENABLEHSERIAL directive must be in your program for this command to function properly.

SetHserial2 (ATOM-Pro Plus only)

SETHSERIAL2 baudrate, databits, parity, stopbits

Baudrate is any one of the constants below.

H300	H26400
H600	H28800
H1200	H31200
H2400	H33600
H4800	H36000
H7200	H38400
H9600	H57600
H12000	H62500
H14400	H125000
H16800	H250000
H19200	H312500
H21600	H500000
H24000	

Databits can be either of the constants below.

H8DATABITS
H7DATABITS

Parity can be any one of the constants below.

HNOPARITY
HEVENPARITY
HODDPARITY

Stopbits can be either of the constants below.

H1STOPBITS
H2STOPBITS

Explanation

The SETHSERIAL2 command sets the HSERIAL2 system baudrate and modes. This command must be used before using a HSERIN2/ HSEROUT2 command. The ENABLEHSERIAL2 directive must be in your program for this command to function properly.

Shiftin

SHIFTIN Data,Clock,Mode,[result{\bits}{,result{\bits}...}]
Shift data in from a synchronous-serial device.

Data is an expression of the I/O pin connected to the synchronous-serial device's output pin. The pin's I/O direction will be changed to an input and will remain in that state after the instruction is completed.

Clock is an expression of the I/O pin connected to the synchronous-serial device's clock input. The pin's I/O direction will be changed to an output and will remain in that state after the instruction is completed.

Mode is a value (0—7) or one of 8 predefined symbols that sets the order in which data bits are to be arranged and the relationship of clock pulses to valid data and the speed of transmition. Here are the symbols,values, and their meanings:

MSBPRE	**0** Data msb-first; sample bits before clock
LSBPRE	**1** Data lsb-first; sample bits before clock
MSBPOST	**2** Data msb-first; sample bits after clock
LSBPOST	**3** Data lsb-first; sample bits after clock
FASTMSBPRE	**4** Data msb-first; sample bits before clock
FASTLSBPRE	**5** Data lsb-first; sample bits before clock
FASTMSBPOST	**6** Data msb-first; sample bits after clock
FASTLSBPOST	**7** Data lsb-first; sample bits after clock

(Msb is most-significant bit; the highest or left most bit of a nibble, byte, word or long. Lsb is the least-significant bit; the lowest or right most bit of a nibble, byte, word or long.)

(Fast mode runs SHIFTIN at the fastest possible rate. Normal mode limits the speed to 100kbps)

Result is a variable where incoming data will be stored.

Bits is an optional entry setting how many bits (1—32) are to be read by SHIFTIN. Defaults to 8 bits.

Explanation

Synchronous serial communications, unlike ansyncronous(i.e. SERIN and SEROUT), is clocked by a master(The ATOM) and data bits are read for each clock pulse. This form of communications is commonly used by many peripherals(ADCs, DACs, clocks, memory devices, etc). Trade names for synchronous-serial protocols include SPI and Microwire.

Shiftout

SHIFTOUT Data,Cpin,Mode,[value{\bits}{,value{\bits}...}]
Shift data out to a synchronous-serial device.

Data is an expression of the I/O pin connected to the synchronous-serial device's input pin. The pin's I/O direction will be changed to an output and will remain in that state after the instruction is completed.

Clock is an expression of the I/O pin connected to the synchronous-serial device's clock input. The pin's I/O direction will be changed to an output and will remain in that state after the instruction is completed.

Mode is a value (0—7) or one of 8 predefined symbols that sets the order in which data bits are to be arranged and the relationship of clock pulses to valid data and the speed of transmition. Here are the symbols,values, and their meanings:

MSBPRE	**0**	Data msb-first; sample bits before clock
LSBPRE	**1**	Data lsb-first; sample bits before clock
MSBPOST	**2**	Data msb-first; sample bits after clock
LSBPOST	**3**	Data lsb-first; sample bits after clock
FASTMSBPRE	**4**	Data msb-first; sample bits before clock
FASTLSBPRE	**5**	Data lsb-first; sample bits before clock
FASTMSBPOST	**6**	Data msb-first; sample bits after clock
FASTLSBPOST	**7**	Data lsb-first; sample bits after clock

Backwards Compatibility:

LSBFIRST	**1**	Data shifted out lsb-first.
MSBFIRST	**0**	Data shifted out msb-first.

Value is an expression of the data to be sent.

Bits is an optional entry setting how many bits (1—32) are to be written by SHIFTOUT. Defaults to 8 bits.

Explanation

Synchronous serial communications, unlike ansyncronous(i.e. SERIN and SEROUT), is clocked by a master(The ATOM) and data bits are read for each clock pulse. This form of communications is commonly used by many peripherals(ADCs, DACs, clocks, memory devices, etc). Trade names for synchronous-serial protocols include SPI and Microwire.

Sleep

Sleep time{,mode}
Sleep the specified time. Optionally change processor speeds or enter standby.

Time is an expression of the time to sleep in approx 2ms increments.

Mode is an optional expression of the mode to enter

Standby mode puts the processor to sleep and shuts off the oscillator. An external interrupt or a reset must be used to wake up the processor:

SLEEPSTANDBY	Enter standby. Wake on external int

The following modes cause the clock multiplier to be set and puts the processor to sleep for the time specified in the Time argument:

SLEEPACTIVE	Normal sleep
SLEEPACTIVE_8	1/8 system clock sleep
SLEEPACTIVE_16	1/16 system clock sleep
SLEEPACTIVE_32	1/32 system clock sleep
SLEEPACTIVE_64	1/64 system clock sleep

The following modes cause a direct transfer to another clock speed divisor: The processor is NOT put to sleep and the Time argument is ignored:

DIRECTACTIVE	Normal system clock.
DIRECTACTIVE_8	1/8 system clock.
DIRECTACTIVE_16	1/16 system clock.
DIRECTACTIVE_32	1/32 system clock.
DIRECTACTIVE_64	1/64 system clock.

Same as above modes except these modes reset all system registers, TimerV, SCI3 and the AD hardware:

DIRECTACTIVERES	Normal system clock
DIRECTACTIVERES_8	1/8 system clock
DIRECTACTIVERES_16	1/16 system clock
DIRECTACTIVERES_32	1/32 system clock
DIRECTACTIVERES_64	1/64 system clock

Explanation

The Sleep command allows you to run the ATOM-Pro at a lower power drain and/or place it into a sleeping/standby state.

Sound

Sound pin,[duration1\note1,...durationN\noteN]
Generate specific note from one pin.

Pin is an expression of the I/O pin to use. This pin will be set to an output during tone generation and left in that state after the instruction is completed.

Duration is an expression of the length in milliseconds of the tone(s).

Note is an expression of the frequency in hertz (Hz, 0 to 32767) of the first tone.

Explanation

The sound command generates a pulse at the specified frequency. The sound command can be used to play tones through a speaker or audio amplifier. Sound can also be used to play simple songs.

Sound2

Sound2
pin1\pin2,[duration1\note1\note2_1,...durationN\noteN\note2_N]
Generate specific notes one on each of the two defined pins.

Pin1 \ **Pin2** are expressions of the I/O pins to use. These pins will be set to an output during tone generation and left in that state after the instruction is completed. The two specified pins can be tied together via resistors(390ohm min) to create a single output signal.

Duration(N) is an expression of the length in milliseconds of the tone(s).

Note1_(N) is an expression of the frequency in hertz (Hz, 0 to 32767) of the first tone.

Note2_(N) is an expression of the frequency in hertz (Hz, 0 to 32767) of the second tone.

Explanation

Sound2 generates two pulses at the specified frequency one on each pin specified. The sound2 command can be used to play tones through a speaker or audio amplifier. Sound2 can also be used to play more complicated songs. By generating two frequencies on separate pins, a more defined sound can be produced.

Spmotor

SPMOTOR pin, delay, step

Pin is an expression of the first pin of 4 control pins required. If P0 was used, the control pins would then be P0, P1, P2, P3.

Delay is an expression of the delay time in milliseconds. Delay controls the speed at which the stepper motor will rotate. The delay will also vary from stepper motor to stepper motor.

Step is an expression of the number of steps and the direction. The direction is determined by the sign of Step. Positive values being clockwise and negative numbers being counter clockwise.

Explanation

Stepper motors are precision motors which have an absolute amount of travel per step. This is ideal in situation where precise positioning is necessary. Stepper motors are commonly found in XY positioning tables. Steppers motors can be purchased from several sources. Chances are you may have a few laying around. They are commonly salvaged from old disk drives and laser printers.

There are two types of stepper motors. Unipolar and Bipolar. Unipolar means one pole. This is usually a common ground between 4 coils. Unipolar stepper motors are easier controlled with minimal circuitry. Bipolar motors indicate two poles. Bipolar motors require additional circuity in order to drive them. The SPMOTOR command does not support Bipolar motors. In most cases you can easily distinguish between the two types. Unipolar stepper motors have 5 wires. Bipolar motors usually have 4.

The use of the SPMOTOR command requires a simple circuit using a darlington array (ULN2803A) to sink the load from the stepper motor. Some small low power stepper motors can be driven from the microcontroller directly. However this is not recommended. Other circuits can be used to sink the load from the stepper motor. The ULN2803A is the most commonly used.

Stop

STOP
Stops program execution.

Explanation

STOP prevents the program from executing any further instructions until it is reset. The STOP command is identical to END.

Swap

SWAP variable,variable

Variable are the variables to be swapped

Explanation

Swap any two variable's values with each other.

Toggle

TOGGLE pin
Invert the state of a pin.

Pin is an expression of the pin number to use.

Explanation

TOGGLE inverts the state of an I/O pin, changing 0 to 1 and 1 to 0. The pin is automatically made an output.

While...Wend

```
While expression
    ...code...
Wend
```

Expression is an expression

Explanation

While some expression is true run code. True being any value other than 0.

Write

WRITE address,expression
Write values to the onboard EEPROM of supported AtomPro modules

Address is an expression of the address to write

Expression can be any variable or constant or combination.

Explanation

WRITE is used to store 8bit values in supported AtomPro modules EEPROM. The ATOM-Pro28 and 40 pin modules have 4kbyte eeproms The ATOM-Pro 24 pin module does not have an onboard eeprom. An external I2C eeprom may be connected to p10(SCL)/p11(SDA) to use the READ/WRITE commands. The ATOM-Pro ARC and Mini-ARC can have an external I2C eeprom added to their 8pin I2C socket to support the READ/WRITE commands.

WriteDM

WRITEDM address,[{Mods}expression]
Write values to the onboard EEPROM of supported AtomPro modules

Address is an expression of the address to begin writing at

Expression can be any variable or constant or combination.

Explanation

WRITEDM is used to store 8bit values in supported AtomPro modules EEPROM. The ATOM-Pro28 and 40 pin modules have 4kbyte eeproms The ATOM-Pro 24 pin module does not have an onboard eeprom. An external I2C eeprom may be connected to p10(SCL)/ p11(SDA) to use the READ/WRITE commands. The ATOM-Pro ARC and Mini-ARC can have an external I2C eeprom added to their 8pin I2C socket to support the READ/WRITE commands.

Reserved Words

Reserved Words

There are many reserved words which can not be used as labels, constants or variables. All command/directive names are reserved words. All words begining with numbers are reserved. All words begining with "_" are reserved. The table below lists all other reserved types and words.

P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P15
P16
P17
P18
P19
P20
P21
P22
P23
P24
P25
P26
P27
P28
P29
P30
P31
P32
P33
P34
P35
P36
P37
P38
P39
P40
P41
P42
P43
P44
P45
P46
P47
P48
P49
S_IN
S_OUT
NMODE
IMODE
NEMODE
IEMODE
NOMODE
IOMODE
NEOMODE
IEOMODE
N300
I300
NE300
IE300
NO300
IO300
NEO300
IEO300
N600
I600
NE600
IE600
NO600
IO600
NEO600
IEO600
N1200
I1200
NE1200
IE1200

NO1200
IO1200
NEO1200
IEO1200
N2400
I2400
NE2400
IE2400
NO2400
IO2400
NEO2400
IEO2400
N4800
I4800
NE4800
IE4800
NO4800
IO4800
NEO4800
IEO4800
N7200
I7200
NE7200
IE7200
NO7200
IO7200
NEO7200
IEO7200
N9600
I9600
NE9600
IE9600
NO9600
IO9600
NEO9600
IEO9600
N12000
I12000
NE12000
IE12000
NO12000
IO12000
NEO12000
IEO12000
N14400
I14400
NE14400
IE14400
NO14400
IO14400
NEO14400
IEO14400
N16800
I16800
NE16800
IE16800
NO16800
IO16800
NEO16800
IEO16800
N19200
I19200
NE19200
IE19200
NO19200
IO19200
NEO19200
IEO19200
N21600
I21600
NE21600
IE21600
NO21600
IO21600
NEO21600
IEO21600
N24000
I24000
NE24000
IE24000
NO24000
IO24000
NEO24000
IEO24000
N26400
I26400
NE26400
IE26400
NO26400
IO26400
NEO26400
IEO26400
N28800
I28800
NE28800
IE28800

NO28800
IO28800
NEO28800
IEO28800
N31200
I31200
NE31200
IE31200
NO31200
IO31200
NEO31200
IEO31200
N33600
I33600
NE33600
IE33600
NO33600
IO33600
NEO33600
IEO33600
N36000
I36000
NE36000
IE36000
NO36000
IO36000
NEO36000
IEO36000
N38400
I38400
NE38400
IE38400
NO38400
IO38400
NEO38400
IEO38400
N57600
I57600
NE57600
IE57600
NO57600
IO57600
NEO57600
IEO57600
N115200
I115200
NE115200
IE115200
NO115200
IO115200
NEO115200
IEO115200
N230400
I230400
NE230400
IE230400
NO230400
IO230400
NEO230400
IEO230400
N460800
I460800
NE460800
IE460800
NO460800
IO460800
NEO460800
IEO460800
H300
H600
H1200
H2400
H4800
H7200
H9600
H12000
H14400
H16800
H19200
H21600
H24000
H26400
H28800
H31200
H33600
H36000
H38400
H57600
H62500
H115200
H125000
H250000
H312500
H500000
HNOPARITY
HEVENPARITY

HODDPARITY
H8DATABITS
H7DATABITS
H1STOPBITS
H2STOPBITS
MSBPRE
LSBPRE
MSBPOST
LSBPOST
FASTMSBPRE
FASTLSBPRE
FASTMSBPOST
FASTLSBPOST
MSBFIRST
LSBFIRST
X_A
X_B
X_C
X_D
X_E
X_F
X_G
X_H
X_I
X_J
X_K
X_L
X_M
X_N
X_O
X_P
X_1
X_2
X_3
X_4
X_5
X_6
X_7
X_8
X_9
X_10
X_11
X_12
X_13
X_14
X_15
X_16
X_Units_On
X_Lights_On
X_On
X_Off
X_Dim
X_Bright
X_Lights_Off
X_Hail
X_Status_On
X_Status_Off
X_Status_Request
CLEAR
HOME
INCCUR
INCSCR
DECCUR
DECSCR
OFF
SCR
SCRBLK
SCRCUR
SCRCURBLK
CURLEFT
CURRIGHT
SCRLEFT
SCRRIGHT
ONELINE
TWOLINE
CGRAM
SCRRAM
TRAP0INT
TRAP1INT
TRAP2INT
TRAP3INT
BREAKINT
DIRECTINT
IRQ0INT
IRQ1INT
IRQ2INT
IRQ3INT
WKPINT_0
WKPINT_1
WKPINT_2
WKPINT_3
WKPINT_4
WKPINT_5
RTCINT (ATOMPro Plus only)
TIMERAINT (ATOMPro only)

TIMERWINT_OVF (ATOMPro only)
TIMERWINT_IMIEA (ATOMPro only)
TIMERWINT_IMIEB (ATOMPro only)
TIMERWINT_IMIEC (ATOMPro only)
TIMERWINT_IMIED (ATOMPro only)
TIMERVINT_OVF
TIMERVINT_CMEB
TIMERVINT_CMEA
SCI3INT_TDRE
SCI3INT_RDRF
SCI3INT_TEND
SCI3INT_OER
SCI3INT_FER
SCI3INT_PER
IICINT
ADINT
TIMERZ0INT_OVF (ATOMPro Plus only)
TIMERZ0INT_IMIEA (ATOMPro Plus only)
TIMERZ0INT_IMIEB (ATOMPro Plus only)
TIMERZ0INT_IMIEC (ATOMPro Plus only)
TIMERZ0INT_IMIED (ATOMPro Plus only)
TIMERZ1INT_UDF (ATOMPro Plus only)
TIMERZ1INT_OVF (ATOMPro Plus only)
TIMERZ1INT_IMIEA (ATOMPro Plus only)
TIMERZ1INT_IMIEB (ATOMPro Plus only)
TIMERZ1INT_IMIEC (ATOMPro Plus only)
TIMERZ1INT_IMIED (ATOMPro Plus only)
TIMERB1INT (ATOMPro Plus only)
SCI3_2INT_TDRE (ATOMPro Plus only)
SCI3_2INT_RDRF (ATOMPro Plus only)
SCI3_2INT_TEND (ATOMPro Plus only)
SCI3_2INT_OER (ATOMPro Plus only)
SCI3_2INT_FER (ATOMPro Plus only)
SCI3_2INT_PER (ATOMPro Plus only)
HSERIALINT_TDRE (ATOMPro Plus only)
HSERIALINT_RDRF (ATOMPro Plus only)
HSERIALINT_TEND (ATOMPro Plus only)
HSERIALINT_OER (ATOMPro Plus only)
HSERIALINT_FER (ATOMPro Plus only)
HSERIALINT_PER (ATOMPro Plus only)
HSERIAL2INT_TDRE (ATOMPro Plus only)
HSERIAL2INT_RDRF (ATOMPro Plus only)
HSERIAL2INT_TEND (ATOMPro Plus only)
HSERIAL2INT_OER (ATOMPro Plus only)
HSERIAL2INT_FER (ATOMPro Plus only)
HSERIAL2INT_PER (ATOMPro Plus only)
HSERVOINT_IDLE
HSERVOINT_IDLE0

HSERVOINT_IDLE1
HSERVOINT_IDLE2
HSERVOINT_IDLE3
HSERVOINT_IDLE4
HSERVOINT_IDLE5
HSERVOINT_IDLE6
HSERVOINT_IDLE7
HSERVOINT_IDLE8
HSERVOINT_IDLE9
HSERVOINT_IDLE10
HSERVOINT_IDLE11
HSERVOINT_IDLE12
HSERVOINT_IDLE13
HSERVOINT_IDLE14
HSERVOINT_IDLE15
HSERVOINT_USER
HSERVOINT
SLEEPACTIVE
SLEEPACTIVE_8
SLEEPACTIVE_16
SLEEPACTIVE_32
SLEEPACTIVE_64
DIRECTACTIVE
DIRECTACTIVE_8
DIRECTACTIVE_16
DIRECTACTIVE_32
DIRECTACTIVE_64
DIRECTACTIVERES
DIRECTACTIVERES_8
DIRECTACTIVERES_16
DIRECTACTIVERES_32
DIRECTACTIVERES_64
SLEEPSTANDBY
TCR_0 (ATOMPro Plus only)
TIORA_0 (ATOMPro Plus only)
TIORC_0 (ATOMPro Plus only)
TSR_0 (ATOMPro Plus only)
TIER_0 (ATOMPro Plus only)
POCR_0 (ATOMPro Plus only)
TCNT_0 (ATOMPro Plus only)
GRA_0 (ATOMPro Plus only)
GRB_0 (ATOMPro Plus only)
GRC_0 (ATOMPro Plus only)
GRD_0 (ATOMPro Plus only)
TCR_1 (ATOMPro Plus only)
TIORA_1 (ATOMPro Plus only)
TIORC_1 (ATOMPro Plus only)
TSR_1 (ATOMPro Plus only)

TIER_1 (ATOMPro Plus only)
POCR_1 (ATOMPro Plus only)
TCNT_1 (ATOMPro Plus only)
GRA_1 (ATOMPro Plus only)
GRB_1 (ATOMPro Plus only)
GRC_1 (ATOMPro Plus only)
GRD_1 (ATOMPro Plus only)
TSTR (ATOMPro Plus only)
TMDR (ATOMPro Plus only)
TPMR (ATOMPro Plus only)
TFCR (ATOMPro Plus only)
TOER (ATOMPro Plus only)
TOCR (ATOMPro Plus only)
RSECDR (ATOMPro Plus only)
RMINDR (ATOMPro Plus only)
RHRDR (ATOMPro Plus only)
RWKDR (ATOMPro Plus only)
RTCCR1 (ATOMPro Plus only)
RTCCR2 (ATOMPro Plus only)
RTCSR (ATOMPro Plus only)
LVDCR (ATOMPro Plus only)
LVDSR (ATOMPro Plus only)
SMR_2 (ATOMPro Plus only)
BRR_2 (ATOMPro Plus only)
SCR3_2 (ATOMPro Plus only)
TDR_2 (ATOMPro Plus only)
SSR_2 (ATOMPro Plus only)
RDR_2 (ATOMPro Plus only)
ICCR1 (ATOMPro Plus only)
ICCR2 (ATOMPro Plus only)
ICMR (ATOMPro Plus only)
ICIER (ATOMPro Plus only)
ICSR (ATOMPro Plus only)
SAR (ATOMPro Plus only)
ICDRT (ATOMPro Plus only)
ICDRR (ATOMPro Plus only)
TMB1 (ATOMPro Plus only)
TCB1 (ATOMPro Plus only)
TLB1 (ATOMPro Plus only)
PCRS1
PCRS2
PCRS3 (ATOMPro Plus only)
PCRS5
PCRS6 (ATOMPro Plus only)
PCRS7
PCRS8
TMRW (ATOMPro only)
TCRW (ATOMPro only)

TIERW (ATOMPro only)
TSRW (ATOMPro only)
TIOR0 (ATOMPro only)
TIOR1 (ATOMPro only)
TCNT (ATOMPro only)
GRA (ATOMPro only)
GRB (ATOMPro only)
GRC (ATOMPro only)
GRD (ATOMPro only)
FLMCR1
FLMCR2
FLPWCR
EBR1
FENR
TCRV0
TCSRV
TCORA
TCORB
TCNTV
TCRV1
TMA (ATOMPro only)
TCA (ATOMPro only)
SMR
BRR
SCR3
TDR
SSR
RDR
ADDRA
ADDRB
ADDRC
ADDRD
ADCSR
ADCR
PWDRL (ATOMPro Plus only)
PWDRU (ATOMPro Plus only)
PWCR (ATOMPro Plus only)
TCSRWD
TCWD
TMWD
ICCR (ATOMPro only)
ICSR (ATOMPro only)
ICDR (ATOMPro only)
SARX (ATOMPro only)
ICMR (ATOMPro only)
SAR (ATOMPro only)
ABRKCR
ABRKSR
BARH
BARL
BDRH
BDRL
PUCR1
PUCR5
PDR1
PDR2
PDR3 (ATOMPro Plus only)
PDR5
PDR6 (ATOMPro Plus only)
PDR7
PDR8
PDRB
PMR1
PMR5
PMR3 (ATOMPro Plus only)
PCR1
PCR2
PCR3 (ATOMPro Plus only)
PCR5
PCR6 (ATOMPro Plus only)
PCR7
PCR8
SYSCR1
SYSCR2
IEGR1
IEGR2
IENR1
IENR (ATOMPro Plus only)
IRR1
IRR2 (ATOMPro Plus only)
IWPR
MSTCR1
TSCR
BUFEB (ATOMPro only)
BUFEA (ATOMPro only)
PWMD (ATOMPro only)
PWMC (ATOMPro only)
PWMB (ATOMPro only)
CKS2 (ATOMPro only)
CKS1 (ATOMPro only)
CKS0 (ATOMPro only)
TOD (ATOMPro only)
TOC (ATOMPro only)
TOB (ATOMPro only)
TOA (ATOMPro only)
IMIED (ATOMPro only)

IMIEC (ATOMPro only)
IMIEB (ATOMPro only)
IMIEA (ATOMPro only)
IMFD (ATOMPro only)
IMFC (ATOMPro only)
IMFB (ATOMPro only)
IMFA (ATOMPro only)
IOB1 (ATOMPro only)
IOB0 (ATOMPro only)
IOA2 (ATOMPro only)
IOA1 (ATOMPro only)
IOA0 (ATOMPro only)
IOD1 (ATOMPro only)
IOD0 (ATOMPro only)
IOC2 (ATOMPro only)
IOC1 (ATOMPro only)
IOC0 (ATOMPro only)
TCNT15 (ATOMPro only)
TCNT14 (ATOMPro only)
TCNT13 (ATOMPro only)
TCNT12 (ATOMPro only)
TCNT11 (ATOMPro only)
TCNT10 (ATOMPro only)
TCNT9 (ATOMPro only)
TCNT8 (ATOMPro only)
TCNT7 (ATOMPro only)
TCNT6 (ATOMPro only)
TCNT5 (ATOMPro only)
TCNT4 (ATOMPro only)
TCNT3 (ATOMPro only)
TCNT2 (ATOMPro only)
TCNT1 (ATOMPro only)
TCNT0 (ATOMPro only)
GRA15 (ATOMPro only)
GRA14 (ATOMPro only)
GRA13 (ATOMPro only)
GRA12 (ATOMPro only)
GRA11 (ATOMPro only)
GRA10 (ATOMPro only)
GRA9 (ATOMPro only)
GRA8 (ATOMPro only)
GRA7 (ATOMPro only)
GRA6 (ATOMPro only)
GRA5 (ATOMPro only)
GRA4 (ATOMPro only)
GRA3 (ATOMPro only)
GRA2 (ATOMPro only)
GRA1 (ATOMPro only)
GRA0 (ATOMPro only)
GRB15 (ATOMPro only)
GRB14 (ATOMPro only)
GRB13 (ATOMPro only)
GRB12 (ATOMPro only)
GRB11 (ATOMPro only)
GRB10 (ATOMPro only)
GRB9 (ATOMPro only)
GRB8 (ATOMPro only)
GRB7 (ATOMPro only)
GRB6 (ATOMPro only)
GRB5 (ATOMPro only)
GRB4 (ATOMPro only)
GRB3 (ATOMPro only)
GRB2 (ATOMPro only)
GRB1 (ATOMPro only)
GRB0 (ATOMPro only)
GRC15 (ATOMPro only)
GRC14 (ATOMPro only)
GRC13 (ATOMPro only)
GRC12 (ATOMPro only)
GRC11 (ATOMPro only)
GRC10 (ATOMPro only)
GRC9 (ATOMPro only)
GRC8 (ATOMPro only)
GRC7 (ATOMPro only)
GRC6 (ATOMPro only)
GRC5 (ATOMPro only)
GRC4 (ATOMPro only)
GRC3 (ATOMPro only)
GRC2 (ATOMPro only)
GRC1 (ATOMPro only)
GRC0 (ATOMPro only)
GRD15 (ATOMPro only)
GRD14 (ATOMPro only)
GRD13 (ATOMPro only)
GRD12 (ATOMPro only)
GRD11 (ATOMPro only)
GRD10 (ATOMPro only)
GRD9 (ATOMPro only)
GRD8 (ATOMPro only)
GRD7 (ATOMPro only)
GRD6 (ATOMPro only)
GRD5 (ATOMPro only)
GRD4 (ATOMPro only)
GRD3 (ATOMPro only)
GRD2 (ATOMPro only)
GRD1 (ATOMPro only)

GRD0 (ATOMPro only)
SWE
ESU
PSU
EV
PV
E
P
FLER
PDWND
EB4
EB3
EB2
EB1
EB0
FLSHE
CMIEB
CMIEA
OVIE
CCLR1
CCLR0
CKS2
CKS1
CKS0
CMFB
CMFA
OVF
OS3
OS2
OS1
OS0
TCORA7
TCORA6
TCORA5
TCORA4
TCORA3
TCORA2
TCORA1
TCORA0
TCORB7
TCORB6
TCORB5
TCORB4
TCORB3
TCORB2
TCORB1
TCORB0
TCNTV7
TCNTV6
TCNTV5
TCNTV4
TCNTV3
TCNTV2
TCNTV1
TCNTV0
TVEG1
TVEG0
TRGE
OCKS0
TMA7 (ATOMPro only)
TMA6 (ATOMPro only)
TMA5 (ATOMPro only)
TMA3 (ATOMPro only)
TMA2 (ATOMPro only)
TMA1 (ATOMPro only)
TMA0 (ATOMPro only)
TCA7 (ATOMPro only)
TCA6 (ATOMPro only)
TCA5 (ATOMPro only)
TCA4 (ATOMPro only)
TCA3 (ATOMPro only)
TCA2 (ATOMPro only)
TCA1 (ATOMPro only)
TCA0 (ATOMPro only)
COM
CHR
PE
PM
STOP
MP
CKS1
CKS0
BRR7
BRR6
BRR5
BRR4
BRR3
BRR2
BRR1
BRR0
TIE
RIE
TE
RE
MPIE

TEIE
CKE1
CKE0
TDR7
TDR6
TDR5
TDR4
TDR3
TDR2
TDR1
TDR0
TDRE
RDRF
OER
FER
PER
TEND
MPBR
MPBT
RDR7
RDR6
RDR5
RDR4
RDR3
RDR2
RDR1
RDR0
ADF
ADIE
ADST
SCAN
CKS
CH2
CH1
CH0
TRGE
B6WI
TCWE
B4WI
TCSRWE
B2WI
WDON
B0WI
WRST
TCWD7
TCWD6
TCWD5
TCWD4
TCWD3
TCWD2
TCWD1
TCWD0
CKS3
CKS2
CKS1
CKS0
ICE (ATOMPro only)
IEIC (ATOMPro only)
MST (ATOMPro only)
TRS (ATOMPro only)
ACKE (ATOMPro only)
BBSY (ATOMPro only)
IRIC (ATOMPro only)
SCP (ATOMPro only)
ESTP (ATOMPro only)
STOP (ATOMPro only)
IRTR (ATOMPro only)
AASX (ATOMPro only)
AL (ATOMPro only)
AAS (ATOMPro only)
ADZ (ATOMPro only)
ACKB (ATOMPro only)
ICDR7 (ATOMPro only)
ICDR6 (ATOMPro only)
ICDR5 (ATOMPro only)
ICDR4 (ATOMPro only)
ICDR3 (ATOMPro only)
ICDR2 (ATOMPro only)
ICDR1 (ATOMPro only)
ICDR0 (ATOMPro only)
SVAX6 (ATOMPro only)
SVAX5 (ATOMPro only)
SVAX4 (ATOMPro only)
SVAX3 (ATOMPro only)
SVAX2 (ATOMPro only)
SVAX1 (ATOMPro only)
SVAX0 (ATOMPro only)
FSX (ATOMPro only)
MLS (ATOMPro only)
WAIT (ATOMPro only)
CKS2 (ATOMPro only)
CKS1 (ATOMPro only)
CKS0 (ATOMPro only)
BC2 (ATOMPro only)
BC1 (ATOMPro only)
BC0 (ATOMPro only)

SVA6 (ATOMPro only)
SVA5 (ATOMPro only)
SVA4 (ATOMPro only)
SVA3 (ATOMPro only)
SVA2 (ATOMPro only)
SVA1 (ATOMPro only)
SVA0 (ATOMPro only)
FS (ATOMPro only)
RTINTE
CSEL1
CSEL0
ACMP2
ACMP1
ACMP0
DCMP1
DCMP0
ABIF
ABIE
BARH7
BARH6
BARH5
BARH4
BARH3
BARH2
BARH1
BARH0
BARL7
BARL6
BARL5
BARL4
BARL3
BARL2
BARL1
BARL0
BDRH7
BDRH6
BDRH5
BDRH4
BDRH3
BDRH2
BDRH1
BDRH0
BDRL7
BDRL6
BDRL5
BDRL4
BDRL3
BDRL2
BDRL1
BDRL0
PUCR17
PUCR16
PUCR15
PUCR14
PUCR12
PUCR11
PUCR10
PUCR55
PUCR54
PUCR53
PUCR52
PUCR51
PUCR50
PIN17
PIN16
PIN15
PIN14
PIN12
PIN11
PIN10
PIN22
PIN21
PIN20
PIN57
PIN56
PIN55
PIN54
PIN53
PIN52
PIN51
PIN50
PIN76
PIN75
PIN74
PIN87
PIN86
PIN85
PIN84
PIN83
PIN82
PIN81
PIN80
PINB7
PINB6
PINB5
PINB4

PINB3
PINB2
PINB1
PINB0
IRQ3
IRQ2
IRQ1
IRQ0
TXD
TMOW
PMR5_WKP5
PMR5_WKP4
PMR5_WKP3
PMR5_WKP2
PMR5_WKP1
PMR5_WKP0
PCR17
PCR16
PCR15
PCR14
PCR12
PCR11
PCR10
PCR22
PCR21
PCR20
PCR57
PCR56
PCR55
PCR54
PCR53
PCR52
PCR51
PCR50
PCR76
PCR75
PCR74
PCR87
PCR86
PCR85
PCR84
PCR83
PCR82
PCR81
PCR80
SSBY
STS2
STS1
STS0
NESEL
SMSEL
LSON
DTON
MA2
MA1
MA0
SA1
SA0
NMIEG
IEG3
IEG2
IEG1
IEG0
WPEG5
WPEG4
WPEG3
WPEG2
WPEG1
WPEG0
IENDT
IENTA
IENWP
IEN3
IEN2
IEN1
IEN0
IENTB1 (ATOMPro Plus only)
IRRDT
IRRTA
IRRI3
IRRI2
IRRI1
IRRI0
IWPF5
IWPF4
IWPF3
IWPF2
IWPF1
IWPF0
MSTIIC
MSTS3
MSTAD
MSTWD
MSTTW
MSTTV
MSTTA

IICRST
IICX
DIRE
DIRS
DIRES
DIRL
DIRH
DIREL
DIREH
DIRA
DIRB
DIRC
DIRD
DIREA
DIREB
DIREC
DIRED
DIR0
DIR1
DIR2
DIR3
DIR4
DIR5
DIR6
DIR7
DIR8
DIR9
DIR10
DIR11
DIR12
DIR13
DIR14
DIR15
DIR16
DIR17
DIR18
DIR19
DIR20
DIR21
DIR22
DIR23
DIR24
DIR25
DIR26
DIR27
DIR28
DIR29
DIR30
DIR31
INE
INS
INES
INL
INH
INEL
INEH
INA
INB
INC
IND
INEA
INEB
INEC
INED
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
IN8
IN9
IN10
IN11
IN12
IN13
IN14
IN15
IN16
IN17
IN18
IN19
IN20
IN21
IN22
IN23
IN24
IN25
IN26
IN27
IN28
IN29
IN30
IN31

OUTE
OUTS
OUTES
OUTL
OUTH
OUTEL
OUTEH
OUTA
OUTB
OUTC
OUTD
OUTEA
OUTEB
OUTEC
OUTED
OUT0
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
OUT8
OUT9
OUT10
OUT11
OUT12
OUT13
OUT14
OUT15
OUT16
OUT17
OUT18
OUT19
OUT20
OUT21
OUT22
OUT23
OUT24
OUT25
OUT26
OUT27
OUT28
OUT29
OUT30
OUT31

Index

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X