ZBasic System Library Reference Manual

Version 4.3.2

Publication History

November 2005	First publication
May 2006	Added new routine descriptions, minor corrections
October 2006	Added new routine descriptions, minor corrections
February 2007	Added information on new ZX models
August 2007	Updated for a new ZX model and added new routine descriptions
March 2008	Updated for new ZX models and added new routine descriptions
October 2008	Added new routine descriptions
January 2009	Added information on a new ZX model
April 2009	Added new routine descriptions, minor corrections
June 2009	Updated for new ZX models
January 2010	Updated routine descriptions, added new descriptions
June 2010	Updated for new ZX models
October 2010	Updated for new ZX models
March 2011	Updated for new compiler features
September 2011	Updated for generic target devices and new routines.
March 2012	Updated for new compiler features
January 2013	Updated for new compiler features
January 2014	Updated for new devices, minor corrections and improvements
February 2014	Updated for new devices, minor corrections and improvements
February 2015	Minor corrections and improvements
September 2015	Minor corrections and improvements

Disclaimer

Elba Corp. makes no warranty regarding the accuracy of or the fitness for any particular purpose of the information in this document or the techniques described herein. The reader assumes the entire responsibility for the evaluation of and use of the information presented. The Company reserves the right to change the information described herein at any time without notice and does not make any commitment to update the information contained herein. No license to use proprietary information belonging to the Company or other parties is expressed or implied.

Critical Applications Disclaimer

ELBA CORP. PRODUCTS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE OR TO BE USED IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS IN LIFE-SUPPORT OR SAFETY DEVICES OR SYSTEMS, CLASS III MEDICAL DEVICES, NUCLEAR FACILITIES, APPLICATIONS RELATED TO THE DEPLOYMENT OF AIRBAGS, OR ANY OTHER APPLICATIONS WHERE DEFECT OR FAILURE COULD LEAD TO DEATH, PERSONAL INJURY OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE (INDIVIDUALLY AND COLLECTIVELY, "CRITICAL APPLICATIONS"). FURTHERMORE, ELBA CORP. PRODUCTS ARE NOT DESIGNED OR INTENDED FOR USE IN ANY APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE OR AIRCRAFT. CUSTOMER AGREES, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE ELBA CORP. PRODUCTS, TO THOROUGHLY TEST THE SAME FOR SAFETY PURPOSES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF ELBA CORP. PRODUCTS IN CRITICAL APPLICATIONS.

Trademarks

ZBasic, ZX-24, ZX-24a, ZX-24n, ZX-24p, ZX-24r, ZX-24s, ZX-24t, ZX-24x, ZX-24u, ZX-40, ZX-40a, ZX-40n, ZX-40p, ZX-40r, ZX-40s, ZX-40t, ZX-44, ZX-44a, ZX-44n, ZX-44p, ZX-44r, ZX-44s, ZX-44t, ZX-328n, ZX-328l, ZX-32n, ZX-32l, ZX-1280, ZX-1280n, ZX-1281, ZX-1281n, ZX-32a4, ZX-128a4u and ZX-128a1 are trademarks of Elba Corp.

ZX-24e, ZX-24ae, ZX-24ne, ZX-24pe, ZX-24nu, ZX-24pu, ZX-24ru, ZX-24su, ZX-24xu, ZX-328nu, ZX-128e, ZX-128ne, ZX-1281e and ZX-1281ne are trademarks of Oak Micros used under license from Elba Corp.

AVR is a registered trademark of Atmel Corp.

BasicX, BX-24 and BX-35 are trademarks of NetMedia, Inc.

PBasic is a trademark and Basic Stamp is a registered trademark of Parallax, Inc.

Visual Basic is a registered trademark of Microsoft Corp.

Arduino is a trademark of the Arduino Team.

Other brand and product names are trademarks or registered trademarks of their respective owners.

Table of Contents

This page is intentionally blank

System Library Reference

Section 1 - Routines by Category

The ZBasic System Library provides a rich collection comprising hundreds of subroutines and functions that you can use to add functionality to your application. The routines may be divided into several conceptual categories as shown below.

Type Conversion Functions

CBit()	convert a value to type `Bit`
CBool()	convert a value to type `Boolean`
CByte()	convert a value to type `Byte`
CByteArray()	convert an integral value to a reference to a `Byte` array
CInt()	convert a value to type `Integer`
CLng()	convert a value to type `Long`
CNibble()	convert a value to type `Nibble`
CSng()	convert a value to type `Single`
CStr()	convert a value to type `String`
CStrHex()	convert a value to a `String` containing hexadecimal characters
CType()	convert a value to an enumeration member
CUInt()	convert a value to type `UnsignedInteger`
CULng()	convert a value to type `Long`
FixB()	convert a `Single` value to type `Byte`
FixI()	convert a `Single` value to type `Integer`
FixL()	convert a `Single` value to type `Long`
FixUI()	convert a `Single` value to type `UnsignedInteger`
FixUL()	convert a `Single` value to type `UnsignedLong`
To<*enum*>()	convert a value to an enumeration member

Mathematical Functions

Abs()	absolute value
Acos()	arc cosine
Asin()	arc sine
Atn()	arc tangent
Atn2()	arc tangent (quadrant-correct)
Ceiling()	largest integer not greater than a `Single` value
Cos()	cosine
DegToRad()	convert degrees to radians
Exp()	e^x
Exp10()	10^x
Fix()	integer portion of a `Single` value
Floor()	smallest integer not less than a `Single value`
Fraction()	fractional portion of a `Single` value
Log()	natural logarithm
Log10()	common logarithm
Max()	determine the largest of two values
Min()	determine the smallest of two values
Pow()	raise a value to a power
RadToDeg()	convert radians to degrees
Signum()	determine if a value is negative, zero or positive
Sin()	sine
SngClass()	return the class information for a `Single` value
Sqr()	square root
Tan()	tangent

Memory-related Routines

`BitCopy()`	copy a sequence of bits from one part of RAM to another
`BlockMove()`	copy data from one part of RAM to another
`GetBit()`	extract a bit from a value in RAM
`GetEEPROM()`	copy data from Program Memory to RAM
`GetPersistent()`	copy data from Persistent Memory to RAM
`GetProgMem()`	copy data from Program Memory to RAM
`MemAddress()`	determine the RAM address of a variable
`MemAddressU()`	determine the RAM address of a variable
`MemCmp()`	compare two blocks of data in RAM
`MemCopy()`	copy data from one part of RAM to another
`MemSet()`	initialize a block of memory with a byte value
`PersistentPeek()`	read a byte from Persistent Memory
`PersistentPoke()`	write a byte to Persistent Memory
`PutBit()`	set or clear a bit in a value in RAM
`PutEEPROM()`	copy data from RAM to Program Memory
`PutPersistent()`	copy data from RAM to Persistent Memory
`PutProgMem()`	copy data from RAM to Program Memory
`RamPeek()`	read a byte from RAM
`RamPeekDword()`	read a 32-bit value from RAM
`RamPeekWord()`	read a 16-bit value from RAM
`RamPoke()`	write a byte to RAM
`RamPokeDword()`	write a 32-bit value to RAM
`RamPokeWord()`	write a 16-bit value to RAM
`System.Alloc()`	allocate a block of memory
`System.Free()`	deallocate a block of memory
`System.HeapHeadRoom()`	determine the amount of unused space in the heap
`System.HeapSize()`	determine the amount of space reserved for the heap
`VarPtr()`	determine the RAM address of a variable

String-related Routines

`Asc()`	extract a character value from a string
`Chr()`	convert a character value to a string
`Fmt()`	convert a `Single` value to a string
`LCase()`	convert upper case letters to lower case in a string
`Left()`	return the leftmost characters from a string
`Len()`	determine the number of characters in a string
`Mid()`	extract or set a substring in a string
`Right()`	return the rightmost characters from a string
`StrAddress()`	determine the address where string characters are stored
`StrCompare()`	compare two strings, optionally ignoring alphabetic case
`StrFind()`	search for the first occurrence of a string within a string
`StrReplace()`	replace character sequences in a string
`StrType()`	determine the characteristics of a string
`Trim()`	remove leading and trailing spaces from a string
`UCase()`	convert lower case letters to upper case in a string
`ValueI()`	convert string characters to the equivalent `Integer` value
`ValueL()`	convert string characters to the equivalent `Long` value
`ValueS()`	convert string characters to the equivalent `Single` value

Data Manipulation Routines

`FlipBits()`	reverse the order of bits in a byte
`HiByte()`	extract the high byte of a value
`HiWord()`	extract the high word of a value

`LoByte()`	extract the low byte of a value
`LoWord()`	extract the low word of a value
`MakeDword()`	construct a 32-bit value from two 16-bit values
`MakeWord()`	construct a 16-bit value from two 8-bit values
`MakeString()`	construct a string from a sequence of bytes
`MidWord()`	extract the middle two bytes of a 4-byte value
`SetBits()`	set the state of specified bits in a byte
`Shl()`	shift a value to the left
`Shr()`	shift a value to the right
`ToggleBits()`	change the state of specified bits in a byte

Serial Communication Routines

`Debug.Print`	send strings to the debug console
`CloseCom()`	terminate the use of a serial channel
`ComChannels()`	prepare for using multiple serial channels
`Console.Read()`	retrieve a character from the console input queue
`Console.ReadLine()`	retrieve a line from the console input queue
`Console.Write()`	send a string to the console output queue
`Console.WriteLine()`	send a string to the console output queue
`ControlCom()`	specify flow control pins for a serial channel
`DefineCom()`	set the characteristics of a serial channel
`DefineCom3()`	set the characteristics of serial Com3
`OpenCom()`	prepare a serial channel for use
`SerialIn()`	read a character from an input pin
`SerialOut()`	send a character or the characters of a string out on a pin
`StatusCom()`	determine the status of a serial channel

Queue Management Routines

`ClearQueue()`	delete data from a queue
`DrainQueue()`	delete a number of bytes of data from a queue
`GetQueue()`	retrieve data from a queue
`GetQueueBufferSize()`	determine the size of the data area of a queue
`GetQueueCount()`	determine the number of bytes of data in a queue
`GetQueueSpace()`	determine the amount of space available in a queue
`GetQueueStr()`	populate a string with characters from a queue
`OpenQueue()`	prepare a queue for use
`PeekQueue()`	copy data from a queue without removing it
`PutQueue()`	put data in a queue
`PutQueueByte()`	put a byte into a queue
`PutQueueStr()`	put the characters of a string in a queue
`SearchQueue()`	search a queue for a data byte or sequence
`StatusQueue()`	determine if a queue has data available

Date/Time Routines

`GetDate()`	get the month, day, year corresponding to a day number
`GetDateVdlue()`	get the month, day, year corresponding to a day number (packed)
`GetDayNumber()`	compute the day number corresponding to a day of a year
`GetDayOfWeek()`	get the day of the week corresponding to a date value
`GetDayOfYear()`	get the ordinal day of the year corresponding to a date value
`GetElapsedMicroTime()`	compute an elapsed time relative to previous timing data
`GetMicroTime()`	populate a buffer with high resolution timing data
`GetTime()`	get the current hour, minute and second
`GetTimestamp()`	get the current date and time information

`GetTimeValue()`	get the current hour, minute and second (packed)
`PutDate()`	set the current month, day, year
`PutTime()`	set the current hour, minute and second

Input/Output Routines

`ADCtoCom1()`	stream analog conversion data to Com1
`BusRead()`	read data from a bus-oriented device
`BusWrite()`	write data to a bus-oriented device
`CloseI2C()`	deinitialize an I2C communication channel
`ClosePWM()`	deinitialize a 16-bit PWM channel
`ClosePWM8()`	deinitialize an 8-bit PWM channel
`CloseSPI()`	deinitialize an SPI communication channel
`CloseX10()`	deinitialize an X-10 communication channel
`Com1toDAC()`	receive stream of analog conversion data
`CountTransitions()`	count transitions on an input pin
`DACPin()`	produce an analog voltage on an output pin
`DefineBus()`	specify the parameters for accessing a bus-oriented device
`DefineSPI()`	specify the parameters for software-based SPI communication
`DefineX10()`	specify the communication parameters for an X-10 channel
`FreqOut()`	produce a dual-frequency sine wave on an output pin
`Get1Wire()`	receive a bit using the 1-Wire protocol
`Get1WireByte()`	receive a byte using the 1-Wire protocol
`Get1WireData()`	receive one or more bytes using the 1-Wire protocol
`GetADC()`	perform an analog to digital conversion on an input
`GetPin()`	read the state of an input pin
`I2CCmd()`	send/receive data over an I2C channel
`I2CGetByte()`	receive a byte on an I2C channel
`I2CPutByte()`	send a byte on an I2C channel
`I2CStart()`	create a Start condition on an I2C channel
`I2CStop()`	create a Stop condition on an I2C channel
`InputCapture()`	record the high/low times of a pulse train on an input pin
`InputCaptureEx()`	record the high/low times of a pulse train on an input pin
`OpenI2C()`	prepare for I2C communication with an external device
`OpenI2CSlave()`	activate I2C slave mode
`OpenSPI()`	prepare for SPI communication with an external device
`OpenSPISlave()`	activate SPI slave mode
`OpenPWM()`	prepare for 16-bit PWM generation
`OpenPWM8()`	prepare for 8-bit PWM generation
`OpenX10()`	prepare an X-10 communication channel for use
`OutputCapture()`	produce a pulse train
`OutputCaptureEx()`	produce a pulse train on any output pin
`PinHigh()`	set an output pin to the high state
`PinInput()`	configure a pin as an input
`PinLow()`	set an output pin to the low state
`PinOutput()`	configure a pin as an ouput
`PinRead()`	read the logic level present on a pin
`PinToggle()`	set an output pin to the opposite of the current state
`PlaySound()`	reproduce sampled audio on an output pin
`PortBit()`	compose a designator for a specific bit in an I/O port
`PortMask()`	compute the bitmask for the port with which a pin is associated
`PulseIn()`	measure a pulse width on an input pin
`PulseOut()`	generate a pulse on an output pin
`Put1Wire()`	send a bit using the 1-Wire protocol
`Put1WireByte()`	send a byte using the 1-Wire protocol
`Put1WireData()`	send one or more bytes using the 1-Wire protocol

PutDAC()	produce an analog voltage on an output pin
PutPin()	configure an I/O pin
PWM()	initiate 16-bit PWM generation or change the duty cycle
PWM8()	initiate 8-bit PWM generation or change the duty cycle
RCTime()	measure an RC charge/discharge time
Reset1Wire()	send a reset signal using the 1-Wire protocol
SetQueueX10()	specify an additional queue for low-level X-10 operation
ShiftIn()	perform synchronous serial input
ShiftInEx()	perform synchronous serial input with more configurability
ShiftOut()	perform synchronous serial output
ShiftOutEx()	perform synchronous serial output with more configurability
SPICmd()	perform SPI communication with an external device
SPIGetByte()	retrieve a byte from an SPI slave
SPIGetData()	retrieve a series of bytes from an SPI slave
SPIPutByte()	send a byte to an SPI slave
SPIPutData()	send a series of bytes to an SPI slave
SPIStart()	initialize an SPI channel
SPIStop()	deinitialize an SPI channel
StatusX10()	determine the status of an X-10 communication channel
X10Cmd()	send commands using the X-10 protocol
PutTimeStamp()	set the current date and time information
Timer()	get the current clock tick value

Task-related Routines

CallTask	prepare a task to begin execution
DisableInt()	disable interrupts
Delay()	pause a task
DelayUntilClockTick()	pause a task
EnableInt()	conditionally re-enable interrupts
ExitTask()	cause a task to terminate
LockTask()	suspend normal task switching
Pause()	pause a task without relinquishing control
ResumeTask()	cause a waiting task to resume execution
RunTask()	cause a specific task to run
Semaphore()	coordinate the use of a resource
SetInterval()	set the interval timer period
Sleep()	pause a task
StackCheck()	enable or disable stack checking
StatusTask()	determine the status of a task
System.TaskHeadRoom()	determine the unused space in a task's stack
TaskIsLocked()	determine if a task is locked
TaskIsValid()	determine if a task stack is in the task list
UnlockTask()	resume normal task switching
UpdateRTC()	update RTC registers to account for missed ticks
WaitForInterrupt()	pause a task until an external event occurs
WaitForInterval()	pause a task until an interval timer expires
Yield()	allow another task to run

Miscellaneous Routines

CloseWatchDog()	deactivate the watchdog timer
CPUSleep()	cause the CPU to go into sleep mode
CRC16()	compute a 16-bit CRC value
CRC32()	compute a 32-bit CRC value
DelayCycles()	delay for a specified number of CPU cycles

`DelayMicroseconds()`	delay for a specified period of time
`DelayMilliseconds()`	delay for a specified period of time
`FirstTime()`	determine if this is the first the program has been run since downloading
`GetMicroTime()`	populate a buffer with higher precision timing information
`GetElapsedMicroTime()`	determine the elapsed time relative to previous time information
`IIf()`	select the value of one of two expressions
`LBound()`	determine the lower bound of an array
`LongJmp()`	perform a non-local goto (e.g. for exception handling)
`NoOp()`	execute a "nop" instruction
`OpenWatchDog()`	activate the watchdog timer
`ParityCheck()`	check the parity of a data byte
`Randomize()`	initialize the random number generator
`ResetProcessor()`	reset the CPU
`Rnd()`	retrieve the next random number
`SerialNumber()`	retrieve the system software serial number
`SetJmp()`	prepare for a non-local Goto (e.g. exception handling)
`SizeOf()`	determine the size of a data item
`SizeOfU()`	determine the size of a data item
`Span()`	determine the number of elements in an array dimension
`System.DeviceID()`	retrieve the identification characters for the device
`UBound()`	determine the upper bound of an array
`WatchDog()`	reset the watchdog timer
`ZXCmdMode()`	activate the "command mode" (for downloading)

Section 2 - Resource Usage

The various ZBasic target devices offer a variety of resources for use in your program, e.g. timers, interrupts, UART (hardware serial port), analog-to-digital converters, etc. Some of these resources are allocated to specific functions of ZBasic and/or are used by certain ZBasic System Library routines. The resources available on a particular target device vary and the remainder of this section documents the availability for each supported device. Consult the datasheet for the particular target device for detailed information about the device.

In some of the following sub-sections, resource usage is described separately for ZX devices and generic target devices. In others, resource usage is described only in terms of the base device and it is thus necessary to know the base device underlying your particular ZX device; the table below shows the correspondence.

Base CPU Type for ZX Devices

ZX Device	Base CPU Type
ZX-24, ZX-40, ZX-44, ZX-24e	mega32
ZX-24a, ZX-40a, ZX-44a, ZX-24ae	mega644
ZX-24p, ZX-40p, ZX-44p, ZX-24n, ZX-40n, ZX-44n, ZX-24ne, ZX-24pe, ZX-24nu, ZX-24pu	mega644P
ZX-24r, ZX-40r, ZX-44r, ZX-24s, ZX-40s, ZX-44s, ZX-24t, ZX-40t, ZX-44t, ZX-24ru, ZX-24su	mega1284P
ZX-328n, ZX-328l, ZX-32n, ZX-32l, ZX-328nu	mega328P
ZX-1280, ZX-1280n	mega1280
ZX-1281, ZX-1281n, ZX-1281e, ZX-1281ne	mega1281
ZX-128e, ZX-128ne	mega128
ZX-128a1	xmega128A1
ZX-24u, ZX-128a4u	xmega128A4U
ZX-24x, ZX-32a4, ZX-24xu	xmega32A4

Package Designation Codes

In the following sub-sections, some of the tables include package designation codes for the different processor types because the pin assignments vary by package type. The table below gives package designation codes and the corresponding package types for various devices. Note, particularly, that suffixes like A, P and PA have been omitted because the package availability is generally the same irrespective of the suffix.

Package Designation Codes

Code	Package Types	Device or Family
L44	PLCC-44	mega8515, mega8535
P14	PDIP-14, SOIC-14	tiny24, tiny44, tiny441, tiny84, tiny441, tiny841
P20	PDIP-20, SOIC-20	tiny2313, tiny4313
P28	PDIP-28	tiny48, tiny88, ATmega
P40	PDIP-40	ATmega
Q20	VQFN-20, QFN-20, MLF-20	tiny24, tiny44, tiny441, tiny84, tiny441, tiny841, tiny2313, tiny4313, tiny1634
S20	TSSOP-20, SOIC-20	tiny87, tiny167, tiny1634
T28	TQFP-28, MLF-28, QFN-28	tiny48, tiny88, ATmega
T32	TQFP-32, MLF-32, QFN-32	tiny48, tiny88, tiny87, tiny167, tiny828, ATmega
T44	TQFP-44, MLF-44, QFN-44	various ATmega. ATxmega
T64	TQFP-64, MLF-64, QFN-64	various ATmega. ATxmega
T100	TQFP-100, MLF-100, QFN-100	various ATmega. ATxmega

UARTs

An on-board hardware serial port, (UART, USART, or LIN/UART), is used for the Com1 serial channel (if available). By default, the UART is configured to operate at 19,200 baud and is utilized by the System Library Routines Console.Read, Console.ReadLine, Console.Write, Console.WriteLine and Debug.Print. You may set the console to a different initial speed using the compiler directive `Option ConsoleSpeed` (described in the ZBasic Language Reference Manual). You may also reconfigure the UART to a different speed by using the System Library routine OpenCom, specifying the console channel. The UART is also used for the ADCtoCom1 and Com1toDAC routines (available only on ATmega-based ZX devices). In both of these cases, the Com1 speed is automatically configured.

Some target devices have multiple hardware UARTs. In these cases, one of the UARTs is assigned to the Com1 serial channel, another UART is assigned to the Com2 serial channel, etc. as shown in the tables below. The effect of these assignments is generally only important with respect to which I/O pins are available for other purposes if the additional hardware USARTs are not being used. It also will be important if your program manipulates the UART registers directly.

It is important to note that on the ZX-24p, ZX-24n, ZX-24r and ZX-24s, the Com2 serial channel cannot be used at the same time as the hardware I2C channel because the pin 11 is shared between the TxD pin of Com2 and the SDA signal.

Hardware UART Channel Assignment and I/O Pin Usage for ZX Devices

ZX Device	UART	Channel	Tx Pin	Rx Pin
ZX-24, ZX-24a	USART0	Com1[1]	1, D.1	2, D.0
ZX-24p, ZX-24n, ZX-24r, ZX-24s, ZX-24t	USART0	Com1[1]	1, D.1	2, D.0
	USART1	Com2	11, D.3	6, D.2
ZX-40, ZX-40a	USART0	Com1	15, D.1	14, D.0
ZX-40p, ZX-40n, ZX-40r, ZX-40s, ZX-40t	USART0	Com1	15, D.1	14, D.0
	USART1	Com2	17, D.3	16, D.2
ZX-44, ZX-44a	USART0	Com1	10, D.1	9, D.0
ZX-44p, ZX-44n, ZX-44r, ZX-44s, ZX-44t	USART0	Com1	10, D.1	9, D.0
	USART1	Com2	12, D.3	11, D.2
ZX-328n, ZX-328l	USART0	Com1	3, D.1	2, D.0
ZX-32n, ZX-32l	USART0	Com1	31, D.1	30, D.0
ZX-1281, ZX-1281n	USART1	Com1	28, D.3	27, D.2
	USART0	Com2	3, E.1	2, E.0
ZX-1280, ZX-1280n	USART0	Com1	3, E.1	2, E.0
	USART1	Com2	46, D.3	45, D.2
	USART2	Com7	13, H.1	12, H.0
	USART3	Com8	64, J.1	63, J.0
ZX-24x, ZX-24u	USARTD0	Com1[1]	1, D.3	2, D.2
	USARTD1	Com2	D.7	D.6
	USARTC0	Com7	9, C.3	10, C.2
	USARTC1	Com8	5, C.7	6, C.6
	USARTE0	Com9	19, E.3	18, E.2
ZX-32a4, ZX-128a4u	USARTD0	Com1	23, D.3	22, D.2
	USARTD1	Com2	27, D.7	26, D.6
	USARTC0	Com7	13, C.3	12, C.2
	USARTC1	Com8	17, C.7	16, C.6
	USARTE0	Com9	33, E.3	32, E.2
ZX-128a1	USARTD0	Com1	28, D.3	27, D.2
	USARTD1	Com2	32, D.7	31, D.6
	USARTC0	Com7	18, C.3	17, C.2
	USARTC1	Com8	22, C.7	21, C.6
	USARTE0	Com9	38, E.3	37, E.2
	USARTE1	Com10	42, E.7	41, E.6
	USARTF0	Com11	48, F.3	47, F.2
	USARTF1	Com12	52, F.7	51, F.6

ZX-24e, ZX-24ae		USART0	Com1[1]	1, D.1	2, D.0
ZX-24ne, ZX-24pe		USART0	Com1[1]	1, D.1	2, D.0
		USART1	Com2	17, D.3	18, D.2
ZX-24nu, ZX-24pu, ZX-24ru, ZX-24su		USART0	Com1	1, D.1	2, D.0
		USART1	Com2	17, D.3	18, D.2
ZX-24xu		USARTD0	Com1	1, D.3	2, D.2
		USARTD1	Com2	13, D.7	14, D.6
		USARTC0	Com7	9, C.3	10, C.2
		USARTC1	Com8	5, C.7	6, C.6
		USARTE0	Com9	21, E.3	22, E.2
ZX-328nu		USART0	Com1	19, D.1	20, D.0
ZX-128e, ZX-128ne, ZX-1281e, ZX-1281ne		USART0	Com1[1]	19, E.1	20, E.0
		USART1	Com2	9, D.3	10, D.2

[1]For these devices, the Com1 signals are logically inverted.

Hardware UART Channel Assignment and I/O Pin Usage for Generic Target Devices

Target Device	Pkg.	UART	Chan.	Tx Pin	Rx Pin
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A, tiny48, tiny88	all	-	-	-	-
tiny87, tiny167	S20	LIN/UART	Com1	2, A.1	1, A.0
	T32	LIN/UART	Com1	30, A.1	29, A.0
tiny2313, tiny2313A, tiny4313	P20	USART0	Com1	3, D.1	2, D.0
	Q20	USART0	Com1	1, D.1	20, D.0
tiny828	T32	USART0	Com1	2, C.3	1, C.2
tiny441, tiny841	S14	USART0	Com1	12, A.1	11, A.2
		USART1	Com2	8, A.5	9, A.4
	Q20	USART0	Com1	4, A.1	3, A.2
		USART1	Com2	20, A.5	1, A.4
tiny1634	S20	USART0	Com1	1, B.0	2, A.7
		USART1	Com2	19, B.2	20, B.1
	Q20	USART0	Com1	19, B.0	20, A.7
		USART1	Com2	17, B.2	18, B.1
mega16, mega16A, mega32, mega32A, mega644, mega644A	P40	USART0	Com1	15, D.1	14, D.0
	T44	USART0	Com1	10, D.1	9, D.0
mega164A, mega164P, mega164PA, mega324P, mega324PA, mega644P, mega644PA, mega1284P	P40	USART0	Com1	15, D.1	14, D.0
		USART1	Com2	17, D.3	16, D.2
	T44	USART0	Com1	10, D.1	9, D.0
		USART1	Com2	12, D.3	11, D.2
mega48, mega48A, mega48P, mega48PA, mega48PB mega8, mega8A, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	P28	USART0	Com1	3, D.1	2, D.0
	T29	USART0	Com1	27, D.1	26, D.0
	T32	USART0	Com1	31, D.1	30, D.0
(mega328PB only)	T32	USART1	Com2	15, B.3	16, B.4
mega64, mega64A, mega128, mega128A, mega1281, mega2561, AT90CAN32, AT90CAN64, AT90CAN128	T64	USART0	Com1	3, E.1	2, E.0
		USART1	Com2	28, D.3	27, D.2
mega640, mega1280, mega2560	T100	USART0	Com1	3, E.1	2, E.0
		USART1	Com2	46, D.3	45, D.2
		USART2	Com7	13, H.1	12, H.0
		USART3	Com8	64, J.1	63, J.0
mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162	T32	USART1	Com1	9, D.3	8, D.2

Device	Timer	USART	Com	TX	RX
mega16U4, mega32U4	T44	USART1	Com1	21, D.3	20, D.2
mega8515, mega8535	P40	USART0	Com1	15, D.1	14, D.0
	T44	USART0	Com1	10, D.1	9, D.0
	L44	USART0	Com1	13, D.1	11, D.0
mega161	P40	USART0	Com1	11, D.1	10, D.0
	T44	USART0	Com1	8, D.1	4, D.0
mega162	P40	USART0	Com1	11, D.1	10, D.0
		USART1	Com2	4, B.3	3, B.2
	T44	USART0	Com1	8, D.1	7, D.0
		USART1	Com2	43, B.3	42, B.2
mega163, mega323	P40	USART0	Com1	15, D.1	14, D.0
	T44	USART0	Com1	10, D.1	9, D.0
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P	T64	USART0	Com1	2, E.1	3, E.0
mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	T100	USART0	Com1	2, E.1	3, E.0
AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	T64	USART1	Com1	28, D.3	27, D.2
xmegaA1, xmegaA1U	T100	USARTD0	Com1	28, D.3	27, D.2
		USARTD1	Com2	32, D.7	31, D.6
		USARTC0	Com7	18, C.3	17, C.2
		USARTC1	Com8	22, C.7	21, C.6
		USARTE0	Com9	38, E.3	37, E.2
		USARTE1	Com10	42, E.7	41, E.6
		USARTF0	Com11	48, F.3	47, F.2
		USARTF1	Com12	52, F.7	51, F.6
xmegaA3, xmegaA3B, xmegaA3U, xmegaA3BU	T64	USARTD0	Com1	29, D.3	28, D.2
		USARTD1	Com2	33, D.7	32, D.6
		USARTC0	Com7	19, C.3	18, C.2
		USARTC1	Com8	23, C.7	22, C.6
		USARTE0	Com9	39, E.3	38, E.2
xmegaA4, xmegaA4U	T44	USARTD0	Com1	23, D.3	22, D.2
		USARTD1	Com2	27, D.7	26, D.6
		USARTC0	Com7	13, C.3	12, C.2
		USARTC1	Com8	17, C.7	16, C.6
		USARTE0	Com9	33, E.3	32, E.2
xmegaD3	T64	USARTD0	Com1	29, D.3	28, D.2
		USARTC0	Com2	19, C.3	18, C.2
xmegaD4	T44	USARTD0	Com1	23, D.3	22, D.2
		USARTC0	Com2	13, C.3	12, C.2

For native code devices, the table below indicates which ISRs may be automatically included in your application when OpenCom() is used in your program. In the ISR Name column, the symbol # should be replaced with the corresponding USART indicator (e.g. 0 for ATtiny and ATmega or D0 for ATxmega) and the symbol * should be replaced by the software UART timer indicator (see the Timers section). If the compiler cannot determine which specific channel is being opened, ISRs for all channels will be included.

Target CPU	Com Channel	ISR Name
tiny87, tiny167	Com1	`LIN_TC`
	Com3-Com6	`Timer0_CompA`
tiny2313, tiny2313A, tiny4313, tiny828, tiny441, tiny841, tiny1634	Com1, Com2	`USART#_RX, USART#_TX, USART#_UDRE`
	Com3-Com6	`Timer0_CompA`
all other ATtiny	Com3-Com6	`Timer*_CompA`
all ATmega	Com1, Com2, Com7, Com8	`USART#_RX, USART#_TX, USART#_UDRE`
	Com3-Com6	`Timer*_CompA`
all ATxmega	Com1, Com2, Com7-Com12	`USART#_RXC, USART#_TXC, USART#_DRE`
	Com3-Com6	`Timer*_CCA`

Note, particularly, that if the console (typically Com1) is implicitly opened for an application, the ISRs for the console channel will be included in the application even if OpenCom() is not explicitly invoked. The console channel is implicitly opened by default for all ZX devices but not so for generic target devices.

Timers

ZBasic devices generally have multiple timers, depending on the underlying CPU type, that are used for various purposes. One of the timers is (optionally) used to implement the real time clock (RTC), another is used for the software-based serial ports and a third timer is used to provide the precise timing required for certain I/O routines. The specific timer that is used for a particular function varies depending on the underlying CPU type as shown in the table below.

Timer Usage by Target Device

Target Device	RTC Timer	I/O Timer	Serial Timer	PWM8 Timer	PWM 16 Timer	Input Capt.	Output Capt.
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A	Timer0	Timer1	Timer0	Timer0	Timer1	Timer1	Timer1
tiny48, tiny88	Timer0	Timer1	Timer0	-	Timer1	Timer1	Timer1
tiny87, tiny167, tiny2313, tiny2313A, tiny4313, tiny828, tiny1634	Timer0	Timer1	Timer0	Timer0	Timer1	Timer1	Timer1
tiny441, tiny841	Timer0	Timer1	Timer0	Timer0	Timer1 Timer2	Timer1 Timer2	Timer1 Timer2
mega8, mega8A	Timer2	Timer1	Timer2	Timer2	Timer1	Timer1	Timer1
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	Timer0	Timer1	Timer2	Timer2	Timer1	Timer1	Timer1
mega16, mega16A, mega164A, mega164P, mega164PA, mega32, mega32A, mega324P, mega324PA, mega644, mega644A, mega644P, mega644PA, mega8535	Timer0	Timer1	Timer2	Timer2	Timer1	Timer1	Timer1
mega1284P	Timer0	Timer3	Timer2	Timer2	Timer1 Timer3	Timer1 Timer3	Timer1 Timer3
mega8515	Timer0	Timer1	Timer0	Timer0	Timer1	Timer1	Timer1
mega161	Timer0	Timer1	Timer2	Timer2	-	Timer1	Timer1
mega162	Timer0	Timer3	Timer2	Timer2	Timer1 Timer3	Timer1 Timer3	Timer1 Timer3
mega163	Timer2	Timer1	Timer2	Timer2	-	Timer1	Timer1
mega323	Timer0	Timer1	Timer2	Timer2	-	Timer1	Timer1
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P,	Timer0	Timer1	Timer2	Timer2	Timer1	Timer1	Timer1

mega329PA, mega649, mega649A, mega649P							
mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	Timer0	Timer1	Timer2	Timer2	Timer1	Timer1	Timer1
mega1281, mega2561	Timer2	Timer4	Timer0	Timer0	Timer1 Timer3	Timer1 Timer3	Timer1 Timer3
mega64, mega64A, mega128, mega128A	Timer0	Timer1	Timer2	Timer2	Timer1 Timer3	Timer1 Timer3	Timer1 Timer3
mega640, mega1280, mega2560	Timer2	Timer4	Timer0	Timer0	Timer1 Timer3 Timer4 Timer5	Timer1 Timer3 Timer4 Timer5	Timer1 Timer3 Timer4 Timer5
mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162	Timer0	Timer1	Timer0	Timer0	Timer1	Timer1	Timer1
mega16U4, mega32U4	Timer0	Timer3	Timer4	Timer4	Timer1 Timer3	Timer1 Timer3	Timer1 Timer3
AT90CAN32, AT90CAN64, AT90CAN128, AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	Timer2	Timer3	Timer0	Timer0	Timer1 Timer3	Timer1 Timer3	Timer1 Timer3
xmegaA1, xmegaA1U	TimerC1	TimerF1	TimerD1	-	TimerC0 TimerD0 TimerD1 TimerE0 TimerE1 TimerF0 TimerF1	TimerC0 TimerD0 TimerD1 TimerE0 TimerE1 TimerF0 TimerF1	TimerC0 TimerD0 TimerD1 TimerE0 TimerE1 TimerF0 TimerF1
xmegaA3, xmegaA3U, xmegaA3B, xmegaA3BU	TimerC1	TimerE1	TimerD1	-	TimerC0 TimerD0 TimerD1 TimerE0 TimerE1 TimerF0	TimerC0 TimerD0 TimerD1 TimerE0 TimerE1 TimerF0	TimerC0 TimerD0 TimerD1 TimerE0 TimerE1 TimerF0
xmegaA4, xmegaA4U	TimerC1	TimerE0	TimerD1	-	TimerC0 TimerD0 TimerD1 TimerE0	TimerC0 TimerD0 TimerD1 TimerE0	TimerC0 TimerD0 TimerD1 TimerE0
xmegaD3	TimerC1	TimerF0	TimerD0	-	TimerC0 TimerD0 TimerE0 TimerF0	TimerC0 TimerD0 TimerE0 TimerF0	TimerC0 TimerD0 TimerE0 TimerF0
xmegaD4	TimerC1	TimerE0	TimerD0	-	TimerC0 TimerD0 TimerE0	TimerC0 TimerD0 TimerE0	TimerC0 TimerD0 TimerE0

When used, the RTC Timer is configured to generate an interrupt that is used to update the RTC and to trigger task switching. Because its role is so central, the RTC Timer (if enabled) cannot be used for any other purpose. The I/O Timer is used by several I/O related routines as explained in more detail below. The Serial Timer is used to generate interrupts to implement the timing required for serial channels Com3 to Com6. If none of the channels 3-6 are open, the Serial Port Timer can be used for other purposes in your program. Timers are also used for some specialized I/O functions as indicated in the table above.

On ATtiny and ATmega targets, the Serial timer is also used for 8-bit PWM generation. Consequently, use of 8-bit PWM and use of Com3 to Com6 are mutually exclusive. On some target devices, the same timer is indicated for both the RTC and the Serial/8-bit PWM functions. For these devices, the application can employ the RTC or the Serial/8-bit PWM functions but not both.

For each timer, there exists a built-in variable that indicates when the timer is in use. For example, `Register.Timer0Busy` and `Register.TimerC1Busy` are Boolean values that indicate when Timer0

(ATtiny, ATmega) and TimerC1 (ATxmega), respectively, are in use. Prior to using a timer, the ZBasic System Library code checks the value of this variable to see if it is already being used. If it is not in use, the system sets the flag to `True` and then proceeds to use the timer. When it is finished using the timer, the system sets the busy flag to `False`.

Your appplication may do the same by passing the Register variable as a parameter to the Semaphore() function in order to get exclusive access to the timer. Of course, you must set timer busy flag to `False` when your code is finished with the timer to indicate that the timer is no longer in use. Likewise, you may want to acquire a semaphore on a timer busy flag for the I/O Timer before calling a System Library routine that uses I/O Timer. If you succeed in setting the semaphore you'll know that the timer is not already in use. An example of code for this purpose (for ZBasic devices that use Timer1 for the I/O Timer) is shown below.

```
' wait until the timer is available
Do While (Not Semaphore(Register.Timer1Busy))
    Call Sleep(0.5)
Loop

' use the timer
Call LockTask()
Register.Timer1Busy = False
Call ShiftOut(12, 13, 8, &H55)
Call UnlockTask()
```

Note, particularly, the line immediately before the call to `ShiftOut()`. After the semaphore is acquired `Regsister.Timer1Busy` will be `True`. Unless it is set to `False`, the call to `ShiftOut()` will fail because that subroutine will think that the timer is in use.

Caution: setting the busy flag for a timer to `True` and never setting it back to `False` will prevent the proper functioning of all System Library routines that require that timer.

I/O Timer Prescaler Values

Some of the System Library routines that use the I/O Timer allow you to modify the frequency used to clock the timer while others use a fixed frequency determined by the requirements of the routine. The routines that do allow frequency modification are divided into two groups, one controlled by the value of `Register.TimerSpeed1` and the other controlled by the value of `Register.TimerSpeed2`. The table below shows the System Library routines that use a timer and, where applicable, the timer speed variable that controls the timer frequency.

System Library Routines Using TimerSpeed Values

Routine	TimerSpeed Value
ADCtoCom1()	
Com1toDAC()	
CountTransitions()	TimerSpeed1
FreqOut()	
Get1Wire()	
Get1WireByte()	
Get1WireData()	
I2CCmd()[2]	TimerSpeed1
I2CGetByte()[2]	TimerSpeed1
I2CPutByte()[2]	TimerSpeed1
InputCapture()	TimerSpeed1
InputCaptureEx()	TimerSpeed1
OutputCapture()	TimerSpeed1
OutputCaptureEx()	TimerSpeed1
OpenPWM()	
OpenPWM8()	

`RCTime()`	`TimerSpeed2`[1]
`PlaySound()`	
`PulseIn()`	`TimerSpeed2`[1]
`PulseOut()`	`TimerSpeed2`[1]
`Put1Wire()`	
`Put1WireByte()`	
`Put1WireData()`	
`PWM()`	
`PWM8()`	
`Reset1Wire()`	
`ShiftIn()`	`TimerSpeed1`
`ShiftInEx()`	`TimerSpeed1`
`ShiftOut()`	`TimerSpeed1`
`ShiftOutEx()`	`TimerSpeed1`
`SPICmd()`[2]	`TimerSpeed1`
`X10Cmd()`	

Notes:
1) The timer frequency is scaled in some cases. See below.
2) The timer is used only for software based channels.

The table below shows the correspondence between the allowable values for the TimerSpeed registers and the resulting clock frequency applied to the I/O Timer in terms of the CPU frequency. The divisor specified is applied to the CPU clock frequency to yield the I/O Timer clock frequency. For compatibility with BasicX (but only for ZX processors running at 14.7456MHz), some of the routines effectively divide the timer frequency by 2 so that the time units associated with parameters or return values are preserved. If you change the timer speed setting, the scale factor is still applied.

TimerSpeed Selector Values

TimerSpeed Value	Frequency ATtiny, ATmega	Frequency ATxmega
0	0	0
1	`F_CPU / 1`	`F_CPU / 1`
2	`F_CPU / 8`	`F_CPU / 2`
3	`F_CPU / 64`	`F_CPU / 4`
4	`F_CPU / 256`	`F_CPU / 8`
5	`F_CPU / 1024`	`F_CPU / 64`
6	`External T1`	`F_CPU / 256`
7	`External T2`	`F_CPU / 1024`
8-15	`n/a`	`Event 0-7`

The default values of `Register.TimerSpeed1` and `Register.TimerSpeed2` are shown in the table below.

Default TimerSpeed Values

CPU Family	TimerSpeed1	TimerSpeed2
ATmega, ATtiny	1	2
ATxmega	2	4

Note that setting the value of either of the timer speed registers other than by direct assignment using an assignment statement will produce undefined results.

There are several important facts to keep in mind if you modify either of the timer speed values. Firstly, the timer speed values are initialized by the system when it begins running and they are never modified by the system thereafter. If you change a timer speed value, that value will be used by all of the related System Library routines until you change it again. Secondly, the applicable TimerSpeed value is used during the configuration and setup of each I/O function. If you change the TimerSpeed value after a particular I/O function is configured, the change will not affect I/O functions configured before that change.

Note, also, that values returned by some of the System Library routines are scaled based on the default timer speed values. If you change the timer speed setting, you'll have to apply an additional scale factor in order to get the correct results. For example, if you set `Register.TimerSpeed2` to 3 on an ATmega-based device running at 14.7MHz and then call the subroutine `PulseIn()`, a pulse having a width of 100µS will return the value of approximately 12.5µS since the clock speed that you specified is 1/8 that of the default. In order to get the correct pulse width, in seconds, you will have to multiply the value returned by 8. Those return values that are not scaled to seconds represent a number of periods of the timer frequency. So, for example, if you change `Register.TimerSpeed1` to 2 on an ATmega-based device running at 14.7MHz, the values returned by `InputCapture()` represent units of 542nS instead of the default 67.8nS.

Timer Output Compare Pin Mapping

For those devices that support mapping of timer compare outputs to physical pins, the table below shows the default mapping. You can specify a different mapping for your application by using the device configuration parameter `TimerOCPin`. See the description of this device parameter in the ZBasic Lanugage Reference Manual.

Default Output Compare Pin Mapping

Device	Timer Compare Output	TOCCn	Pin
tiny828	OC0A	0	C.0
	OC0B	1	C.1
	OC1A	6	C.6
	OC1B	7	C.7
tiny441, tiny841	OC0A	5	A.6
	OC0B	4	A.5
	OC1A	7	B.2
	OC1B	6	A.7
	OC2A	3	A.4
	OC2B	2	A.3

16-Bit PWM Timers

The tables below give the set of valid 16-bit PWM channels, the associated timer, and the corresponding output pin for each channel. See OpenPWM for the details of setting up a 16-bit PWM output.

16-bit PWM Timers, Channels and Pins for ZX Devices

ZX Device	Timer	Chan.	Pin	Chan.	Pin	Chan.	Pin
ZX-24, ZX-24a, ZX-24p, ZX-24n	Timer1	1	26, D.5	2	27, D.4		
ZX-24e, ZX-24ae, ZX-24ne, ZX-24pe, ZX-24nu, ZX-24pu	Timer1	1	15, D.5	2	16, D.4		
ZX-40, ZX-40a, ZX-40p, ZX-40n	Timer1	1	19, D.5	2	18, D.4		
ZX-44, ZX-44a, ZX-44p, ZX-44n	Timer1	1	14, D.5	2	13, D.4		
ZX-24r, ZX-24s, ZX-24t	Timer1	1	26, D.5	2	27, D.4		
	Timer3	3	B.6	4	B.7		
ZX-24ru, ZX-24su	Timer1	1	15, D.5	2	16, D.4		
	Timer3	3	22, B.6	4	21, B.7		
ZX-40r, ZX-40s, ZX-40t	Timer1	1	19, D.5	2	18, D.4		
	Timer3	3	7, B.6	4	8, B.7		
ZX-44r, ZX-44s, ZX-44t	Timer1	1	14, D.5	2	13, D.4		
	Timer3	3	2, B.6	4	3, B.7		
ZX-328n, ZX-328l	Timer1	1	15, B.1	2	16, B.2		
ZX-32n, ZX-32l	Timer1	1	13, B.1	2	14, B.2		
ZX-328nu	Timer1	1	12, B.1	2	13, B.2		

Target Device	Timer	Chan.	Pin	Chan.	Pin	Chan.	Pin
ZX-1281, ZX-1281n	Timer1	1	15, B.5	2	16, B.6	3	17, B.7
	Timer3	4	5, E.3	5	6, E.4	6	7, E.5
ZX-128e, ZX-128ne, ZX-1281e, ZX-1281ne	Timer1	1	23, B.5	2	22, B.6	3	21, B.7
	Timer3	4	17, E.3	5	16, E.4	5	16, E.5
ZX-1280, ZX-1280n	Timer1	1	24, B.5	2	25, B.6	3	26, B.7
	Timer3	4	5, E.3	5	6, E.4	6	7, E.5
	Timer4	7	15, H.3	8	16, H.4	9	17, H.5
	Timer5	10	38, L.3	11	39, L.4	12	40, L.5
ZX-24x, ZX-24u	TimerD0	1	26, D.0	2	27, D.1		
	TimerD1	3	D.4	4	D.5		
	TimerC0	5	12, C.0	6	11, C.1		
		7	10, C.2	8	9, C.3		
	TimerE0	9	25, E.0	10	17, E.1		
		11	18, E.2	12	19, E.3		
ZX-32a4	TimerD0	1	20, D.0	2	21, D.1		
	TimerD1	3	24, D.4	4	25, D.5		
	TimerC0	5	10, C.0	6	11, C.1		
		7	12, C.2	8	13, C.3		
	TimerE0	9	28, E.0	10	29, E.1		
		11	32, E.2	12	33, E.3		
ZX-24xu	TimerD0	1	20, D.0	2	19, D.1		
	TimerD1	3	16, D.4	4	15, D.5		
	TimerC0	5	12, C.0	6	11, C.1		
		7	10, C.2	8	9, C.3		
	TimerE0	9	24, E.0	10	23, E.1		
		11	11, E.2	12	12, E.3		
ZX-128a1	TimerD0	1	25, D.0	2	26, D.1		
	TimerD1	3	29, D.4	4	30, D.5		
	TimerC0	5	15, C.0	6	16, C.1		
		7	17, C.2	8	18, C.3		
	TimerE0	9	35, E.0	10	36, E.1		
		11	37, E.2	12	38, E.3		
	TimerE1	13	39, E.4	14	40, E.5		
	TimerF0	15	45, F.0	16	46, F.1		
		17	47, F.2	18	48, F.3		
	TimerF1	19	49, F.4	20	50, F.5		

16-bit PWM Timers, Channels and Pins for Generic Target Devices

Target Device	Pkg.	Timer	Chan.	Pin	Chan.	Pin	Chan.	Pin
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A	P14	Timer1	1	7, A.6	2	8, A.5		
	Q20	Timer1	1	16, A.6	2	20, A.5		
tiny48, tiny88	P28	Timer1	1	15, B.1	2	16, B.2		
	T28	Timer1	1	11, B.1	2	12, B.2		
	T32	Timer1	1	13, B.1	2	14, B.2		
tiny441, tiny841[1]	S14	Timer1	1	5, B.2	2	6, A.7		
		Timer2	3	9, A.4	4	10, A.3		
	Q20	Timer1	1	14, B.2	2	15, A.7		
		Timer2	3	1, A.4	4	2, A.3		
tiny87, tiny167	S20	Timer1	1	20, B.0	2	19, B.1		
	T32	Timer1	1	28, B.0	2	27, B.1		
tiny2313, tiny2313A, tiny4313	P20	Timer1	1	15, B.3	2	16, B.4		
	Q20	Timer1	1	13, B.3	2	14, B.4		
tiny828[1]	T32	Timer1	1	7, C.6	2	8, C.7		
tiny1634	S20	Timer1	1	18, B.3	2	3, A.6		
	Q20	Timer1	1	16, B.3	2	1, A.6		
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A,	P28	Timer1	1	15, B.1	2	16, B.2		

Device	Package	Timer						
mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB								
	T28	Timer1	1	11, B.1	2	12, B.2		
	T32	Timer1	1	13, B.1	2	14, B.2		
(mega328PB only)	T32	Timer3	3	30, D.0	4	32, D.2		
(mega328PB only)	T32	Timer4	5	31, D.1	6	32, D.2		
mega16, mega16A, mega32, mega32A, mega644, mega644A, mega164A, mega164P, mega164PA, mega324P, mega324PA, mega644P, mega644PA	P40	Timer1	1	19, D.5	2	18, D.4		
	T44	Timer1	1	14, D.5	2	13, D.4		
mega1284P	P40	Timer1	1	19, D.5	2	18, D.4		
		Timer3	3	7, B.6	4	8, B.7		
	P44	Timer1	1	14, D.5	2	13, D.4		
		Timer3	3	2, B.6	4	3, B.7		
mega64, mega64A, mega128, mega128A, mega1281, mega2561, AT90CAN32, AT90CAN64, AT90CAN128	T64	Timer1	1	15, B.5	2	16, B.6	3	5, E.3
		Timer3	4	17, B.7	5	6, E.4	6	7, E.5
AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	T64	Timer1	1	15, B.5	2	16, B.6	3	17, B.7
		Timer3	4	41, C.6	5	40, C.5	6	39, C.4
mega640, mega1280, mega2560	T100	Timer1	1	24, B.5	2	25, B.6	3	26, B.7
		Timer3	4	5, E.3	5	6, E.4	6	7, E.5
		Timer4	7	15, H.3	8	16, H.4	9	17, H.5
		Timer5	10	38, L.3	11	39, L.4	12	40, L.5
mega8U2, mega16U2, mega32U2 AT90USB82, AT90USB162	T32	Timer1	1	23, C.6	2	25, C.5	3	12, B.7
mega16U4, mega32U4	T44	Timer1	1	29, B.5	2	30, B.6	3	21, B.7
		Timer3	4	31, C.6				
mega8515	P40	Timer1	1	15, D.5	2	29, E.2		
	T44	Timer1	1	11, D.5	2	26, E.2		
	L44		1	17, D.5	2	32, E.2		
mega8535	P40	Timer1	1	19, D.5	2	18, D.4		
	T44	Timer1	1	14, D.5	2	13, D.4		
	L44	Timer1	1	20, D.5	2	19, D.4		
mega162	P40	Timer1	1	19, D.5	2	29, E.2		
		Timer3	3	14, D.4	4	5, B.4		
	T44	Timer1	1	11, D.5	2	26, E.2		
		Timer3	3	10, D.4	4	44, B.4		
mega161, mega163, mega323	all	-						
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P	T64	Timer1	1	15, B.5	2	16. B.6		
mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	T100	Timer1	1	24, B.5	2	25. B.6		
xmegaA1, xmegaA1U	T100	TimerD0	1	25, D.0	2	26, D.1		
		TimerD1	3	29, D.4	4	30, D.5		
		TimerC0	5	15, C.0	6	16, C.1		
			7	17, C.2	8	18, C.3		
		TimerE0	9	35, E.0	10	36, E.1		
			11	37, E.2	12	38, E.3		
		TimerE1	13	39, E.4	14	40, E.5		
		TimerF0	15	45, F.0	16	46, F.1		
			17	47, F.2	18	48, F.3		

		TimerF1	19	49, F.4	20	50, F.5
xmegaA3, xmegaA3B, xmegaA3U, xmegaA3BU	T64	TimerD0	1	26, D.0	2	27, D.1
		TimerD1	3	30, D.4	4	31, D.5
		TimerC0	5	16, C.0	6	17, C.1
			7	18, C.2	8	19, C.3
		TimerE0	9	36, E.0	10	37, E.1
			11	38, E.2	12	39, E.3
		TimerE1	13	40, E.4	14	42, E.5
		TimerF0	15	46, F.0	16	47, F.1
			17	48, F.2	18	49, F.3
xmegaA4, xmegaA4U	T44	TimerD0	1	20, D.0	2	21, D.1
		TimerD1	3	24, D.4	4	25, D.5
		TimerC0	5	10, C.0	6	11, C.1
			7	12, C.2	8	13, C.3
		TimerE0	9	28, E.0	10	29, E.1
			11	32, E.2	12	33, E.3
xmegaD3	T64	TimerD0	1	26, D.0	2	27, D.1
		TimerC0	3	16, C.0	4	17, C.1
			5	18, C.2	6	19, C.3
		TimerE0	7	36, E.0	8	37, E.1
			9	38, E.2	10	39, E.3
		TimerF0	11	46, F.0	12	47, F.1
			13	48, F.2	14	49, F.3
xmegaD4	T44	TimerD0	1	20, D.0	2	21, D.1
		TimerC0	3	10, C.0	4	11, C.1
			5	12, C.2	6	13, C.3
		TimerE0	7	28, E.0	8	29, E.1
			9	32, E.2	10	33, E.3

[1]These devices support mapping of timer output compare pins; the pins listed are the default pins.

8-Bit PWM Timers

The tables below give the set of valid 8-bit PWM channels, the associated timer, and the corresponding output pin for each channel. See OpenPWM8 for the details of setting up an 8-bit PWM output.

8-bit PWM Timers, Channels and Pins for ZX Devices

ZX Device	Timer	Chan.	Pin	Chan.	Pin
ZX-24	Timer2	1	25, D.7		-
ZX-40	Timer2	1	21, D.7		-
ZX-44	Timer2	1	16, D.7		-
ZX-24a, ZX-24p, ZX-24n, ZX-24r, ZX-24s, ZX-24t	Timer2	1	25, D.7	2	12, D.6
ZX-40a, ZX-40p, ZX-40n, ZX-40r, ZX-40s, ZX-40t	Timer2	1	21, D.7	2	20, D.6
ZX-44a, ZX-44p, ZX-44n, ZX-44r, ZX-44s, ZX-44t	Timer2	1	16, D.7	2	15, D.6
ZX-328n, ZX-328l	Timer2	1	17, B.3	2	5, D.3
ZX-32n, ZX-32l	Timer2	1	15, B.3	2	1, D.3
ZX-1281, ZX-1281n	Timer0	1	17, B.7	2	1, G.5
ZX-1280, ZX-1280n	Timer0	1	26, B.7	2	1, G.5
ZX-24e	Timer2	1	13, D.7		-
ZX-24ae, ZX-24ne, ZX-24pe, ZX-24nu, ZX-24pu, ZX-24ru, ZX-24su	Timer2	1	13, D.7	2	14, D.6
ZX-128e, ZX-128ne	Timer2	1	17, B.7		-
ZX-1281e, ZX-1281ne	Timer0	1	21, B.7	2	G.5
ZX-328nu	Timer2	1	14, B.3	2	6, D.3

8-bit PWM Timers, Channels and Pins for Generic Target Devices

Target Devices	Pkg.	Timer	Chan.	Pin	Chan.	Pin
tiny48, tiny88	all	-				
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A	P14	Timer0	1	5, B.2	2	6, A.7
	Q20		1	14, B.2	2	15, A.7
tiny87, tiny167	S20	Timer0	1	3, A.2		
	T32		1	31, A.2		
tiny2313, tiny2313A, tiny4313	P20	Timer0	1	14, B.2	2	9, D.5
	Q20		1	7, B.2	2	12, D.5
tiny828[†]	T32	Timer0	1	31, C.0	2	32, C.1
tiny441, tiny841[†]	S14	Timer0	1	7, A.6	2	8, A.5
	Q20		1	16, A.6	2	20, A.5
tiny1634	S20	Timer0	1	17, C.0	2	4, A.5
	Q20		1	15, C.0	2	2, A.5
mega8, mega8A	P28	Timer2	1	17, B.3		
	T28		1	13, B.3		
	T32		1	15, B.3		
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	P28	Timer2	1	17, B.3	2	5, D.3
	T28		1	13, B.3	2	1, D.3
	T32		1	15, B.3	2	1, D.3
mega16, mega16A, mega32, mega32A	P40	Timer2	1	21, D.7		
	T44		1	16, D.7		
mega644, mega644A, mega164A, mega164P, mega164PA, mega324P, mega324PA, mega644P, mega644PA, mega1284P	P40	Timer2	1	21, D.7	2	20, D.6
	T44		1	16, D.7	2	15, D.6
mega64, mega64A, mega128, mega128A, AT90CAN32, AT90CAN64, AT90CAN128	T64	Timer0	1	17, B.7		
mega1281, mega2561	T64	Timer0	1	17, B.7	2	1, G.5
AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	T64	Timer0	1	17, B.7	2	25, D.0
mega640, mega1280, mega2560	T100	Timer0	1	26, B.7	2	1, G.5
mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162	T32	Timer0	1	21, B.7	6	6, B.0
mega16U4, mega32U4	T44	Timer4	1	32, C.7	2	30, B.6
			3	27, D.7		
mega8515	P40	Timer0	1	1, B.0		
	T44		1	40, B.0		
	L44		1	2, B.0		
mega8535	P40	Timer2	1	21, D.7		
	T44		1	16, D.7		
	L44		1	22, D.7		
mega161, mega162	P40	Timer2	1	2, B.1		
	T44		1	41, B.1		
mega163, mega323	P40	Timer2	1	21, D.7		
	T44		1	16, D.7		
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P	T64	Timer2	1	17, B.7		
mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	T100	Timer2	1	26, B.7		
all xmega	all	-				

[†]These devices support mapping of timer output compare pins; the pins listed are the default pins.

Input Capture Timers

The tables below give the set of valid pins for the InputCaptureEx subroutine and also indicate (with an asterisk) the default pin used by the InputCapture subroutine. When an input capture operation has been started successfully, the corresponding timer busy flag (e.g. `Register.Timer1Busy`) will be set `True` for the duration of the input capture operation. Note that an input capture cannot be used at the same time as an output capture operation involving the same timer.

For native code ZX devices and all generic target devices, at least two ISRs will be automatically provided by the compiler to facilitate the input capture operation corresponding to the capture interrupt and the timer overflow interrupt. The names of the interrupt vectors are related to the timer being used. For example, for Timer1 the names are `Timer1_Capt` and `Timer1_OVF`. Note that if the compiler cannot determine at compile time which timer will be used, the capture and timer overflow ISRs for all possible timers will be included in the application.

Valid Input Capture Pins for ZX Devices

ZX Device	Timer	Pin
ZX-24, ZX-24a, ZX-24p, ZX-24n	Timer1	12, D.6*
ZX-40, ZX-40a, ZX-40p, ZX-40n	Timer1	20, D.6*
ZX-44, ZX-44a, ZX-44p, ZX-44n	Timer1	15, D.6*
ZX-24r, ZX-24s, ZX-24t	Timer1	12, D.6*
	Timer3	B.5
ZX-40r, ZX-40s, ZX-40t	Timer1	20, D.6*
	Timer3	6, B.5
ZX-44r, ZX-44s, ZX-44t	Timer1	15, D.6*
	Timer3	1, B.5
ZX-24e, ZX-24ae, ZX-24ne, ZX-24pe, ZX-24nu, ZX-24pu	Timer1	14, D.6*
ZX-24ru, ZX-24su	Timer1	14, D.6*
	Timer3	23, B.5
ZX-328n, ZX-328l	Timer1	14 B.0*
ZX-32n, ZX-32l	Timer1	12 B.0*
ZX-1281, ZX-1281n	Timer1	29, D.4*
	Timer3	9, E.7
ZX-1280, ZX-1280n	Timer1	47, D.4*
	Timer3	9, E.7
	Timer4	35, L.0
	Timer5	36, L.1
ZX-24x, ZX-24u	TimerC0	12, C.0*
	TimerD0	26, D.0
	TimerE0	25, E.0
ZX-32a4	TimerC0	10, C.0*
	TimerD0	20, D.0
	TimerE0	28, E.0
ZX-128a1	TimerC0	15, C.0*
	TimerD0	25, D.0
	TimerE0	35, E.0
	TimerE1	39, E.4
	TimerF0	45, F.0
	TimerF1	49, F.4
ZX-128e, ZX-128ne, ZX-1281e, ZX-1281ne	Timer1	8, D.4*
	Timer3	13, E.7
ZX-328nu	Timer1	11, B.0*
ZX-24xu	TimerC0	12, C.0*
	TimerD0	20, D.0
	TimerE0	24, E.0

Valid Input Capture Pins for Generic Target Devices

Underlying CPU Type	Pkg.	Timer	Pin
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A	P14	Timer1	6, A.7*
	Q20	Timer1	15, A.7*
tiny48, tiny88	P28	Timer1	14, B.0*
	T28	Timer1	10, B.0*
	T32	Timer1	12, B.0*
tiny441, tiny841	S14	Timer1	6, A.7*
		Timer2	5, B.2
	Q20	Timer1	15, A.7*
		Timer2	14, B.2
tiny87, tiny167	S20	Timer1	7, A.4*
	T32	Timer1	9, A.4*
tiny2313, tiny2313A, tiny4313	P20	Timer1	11, D.6*
	Q20	Timer1	9, D.6*
tiny828	T32	Timer1	6, C.5*
tiny1634	S20	Timer1	16, C.1*
	Q20	Timer1	14, C.1*
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	P28	Timer1	14, B.0*
	T32	Timer1	12, B.0*
(mega328PB only)	T32	Timer3	19, E.2
(mega328PB only)	T32	Timer4	3, E.0*
mega16, mega16A, mega164A, mega164P, mega164PA, mega32, mega32A, mega324P, mega324PA, mega644, mega644A, mega644P, mega644PA	P40	Timer1	20, D.6*
	T44	Timer1	15, D.6*
mega1284P	P40	Timer1	20, D.6*
		Timer3	6, B.5
	T44	Timer1	15, D.6*
		Timer3	1, B.5
mega8515	P40	Timer1	31*
	T44	Timer1	21*
	L44	Timer1	35*
mega8535, mega163, mega323	P40	Timer1	20, D.6*
	T44	Timer1	15, D.6*
	L44	Timer1	21, D.6*
mega161, mega162	P40	Timer1	31, E.0*
	T44	Timer1	29, E.0*
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P	T64	Timer1	25, D.0*
mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	T100	Timer1	43, D.0*
mega1281, mega2561, mega64, mega64A, mega128, mega128A AT90CAN32, AT90CAN64, AT90CAN128, AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	T64	Timer1	29, D.4*
		Timer3	9, E.7
mega640, mega1280, mega2560	T100	Timer1	47, D.4*
		Timer3	9, E.7
		Timer4	35, L.0
		Timer5	36, L.1
mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162	T32	Timer1	22, C.7*
mega16U4, mega32U4	T44	Timer1	25, D.4*
		Timer3	32, C.7
xmegaA1, xmegaA1U	T100	TimerC0	15, C.0*

		TimerD0	25, D.0
		TimerE0	35, E.0
		TimerE1	39, E.4
		TimerF0	45, F.0
		TimerF1	49, F.4
xmegaA3, xmegaA3BU, xmegaA3U, xmegaA3BU	T64	TimerC0	16, C.0*
		TimerD0	26, D.0
		TimerE0	36, E.0
		TimerF0	46, F.0
xmegaA4, xmegaA4U	T44	TimerC0	10, C.0*
		TimerD0	20, D.0
		TimerE0	28, E.0
xmegaD3	T64	TimerC0	16, C.0*
		TimerD0	26, D.0
		TimerE0	36, E.0
		TimerF0	46, F.0
xmegaD4	T44	TimerC0	10, C.0*
		TimerD0	20, D.0
		TimerE0	28, E.0

Output Capture Timers

The tables below give the set of pins and corresponding timers that can be used by the OutputCaptureEx subroutine and also indicate (with an asterisk) the default pin used by the OutputCapture subroutine. When an output capture operation has been started successfully, the corresponding timer busy flag (e.g. `Register.Timer1Busy`) will be set `True` for the duration of the output capture operation. Note that an output capture cannot be used at the same time as an input capture operation involving the same timer.

When performing an output capture on a general I/O pin (i.e. a pin not listed in the tables below), any available 16-bit timer will be used to generate the required timing. If no 16-bit timer is available at the time, the routine will return immediately.

For native code ZX devices and all generic target devices, at least one ISR will be provided by the compiler automatically to facilitate the output capture operation corresponding to the timer compare interrupt. The names of the interrupt vectors are related to the timer and the compare register being used. For example, for an ATtiny or ATmega device using Timer1 the ISR name would be `Timer1_COMPB` while for an xmega device for TimerC0 the ISR name would be `TIMERC0_CCB`. Note that if the compiler cannot determine at compile time which timer and compare register will be used, or if output capture on a general I/O pin is specified, the "compare B" ISRs for all possible timers will be included in the application.

Hardware Output Capture Pins for ZX Devices

ZX Device	Timer	Output Pin	Timer	Output Pin
ZX-24, ZX-24a, ZX-24p, ZX-24n	Timer1	27, D.4*		
ZX-40, ZX-40a, ZX-40p, ZX-40n	Timer1	18, D.4*		
ZX-44, ZX-44a, ZX-44p, ZX-44n	Timer1	13, D.4*		
ZX-24r, ZX-24s, ZX-24t	Timer1	27, D.4*	Timer3	B.7
ZX-40r, ZX-40s, ZX-40t	Timer1	18, D.4*	Timer3	8, B.7
ZX-44r, ZX-44s, ZX-44t	Timer1	13, D.4*	Timer3	3, B.7
ZX-328n, ZX-328l	Timer1	16, B.2*		
ZX-32n, ZX-32l	Timer1	14, B.2*		
ZX-1281, ZX-1281n	Timer1	16, B.6*	Timer1	17, B.7[†]
	Timer3	6, E.4		
ZX-1280, ZX-1280n	Timer1	25, B.6*	Timer1	26, B.7[†]
	Timer3	6, E.4	Timer4	16, H.4
	Timer5	39, L.4		
ZX-24x, ZX-24u	TimerC0	11, C.1	TimerD0	27, D.1*

Target Device	Timer	Pin	Timer	Pin
	TimerE0	17, E.1		
ZX-32a4	TimerC0	11, C.1	TimerD0	21, D.1*
	TimerE0	29, E.1		
ZX-128a1	TimerC0	16, C.1	TimerD0	26, D.1*
	TimerE0	36, E.1	TimerE1	40, E.5
	TimerF0	46, F.1	TimerF1	50, F.5
ZX-24e, ZX-24ae, ZX-24ne, ZX-24pe, ZX-24nu, ZX-24pu	Timer1	13, D.4*		
ZX-24ru, ZX-24su	Timer1	16, D.4*	Timer3	21, B.7
ZX-128e, ZX-128ne, ZX-1281e, ZX-1281ne	Timer1	22, B.6*	Timer1	21, B.7[1]
	Timer3	16, E.4		
ZX-328nu	Timer1	13, B.2*		
ZX-24xu	TimerC0	11, C.1	TimerD0	13, D.1*
	TimerE0	23, E.1		

* Denotes the default OutputCapture pin.

[1] Requires the TIMER1_COMPC ISR and supports OutputCapture modulation.

Hardware Output Capture Pins for Generic Target Devices

Target Device	Pkg.	Timer	Output Pin
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A	P14	Timer1	8, A.5*
	Q20	Timer1	20, A.5*
tiny48, tiny88	P28	Timer1	16, B.2*
	T28	Timer1	12, B.2*
	T32	Timer1	14, B.2*
tiny441, tiny841[2]	S14	Timer1	6, A.7*
		Timer2	10, A.3
	Q20	Timer1	15, A.7*
		Timer2	2, A.3
tiny87, tiny167	S20	Timer1	19, B.1*
	T32	Timer1	27, B.1*
tiny2313, tiny2313A, tiny4313	P20	Timer1	16, B.4*
	Q20	Timer1	14, B.4*
tiny828[2]	T32	Timer1	8, C.7*
tiny1634	S20	Timer1	3, A.6*
	Q20	Timer1	1, A.6*
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	P28	Timer1	16, B.2*
	T28	Timer1	12, B.2*
	T32	Timer1	14, B.2*
(mega328PB only)	T32	Timer3	32, D.2
(mega328PB only)	T32	Timer4	32, D.2
mega16, mega16A, mega32, mega32A, mega644, mega644A, mega164A, mega164P, mega164PA, mega324P, mega324PA, mega644P, mega644PA	P40	Timer1	18, D.4*
	T44	Timer1	13, D.4*
mega1284P	P40	Timer1	18, D.4*
		Timer3	8, B.7
	T44	Timer1	13, D.4*
		Timer3	3, B.7
mega64, mega64A, mega128, mega128A, mega1281, mega2561, AT90CAN32, AT90CAN64, AT90CAN128	T64	Timer1	16, B.6*
		Timer1	17, B.7[1]
		Timer3	6, E.4
AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	T64	Timer1	16, B.6*
		Timer1	17, B.7[1]
		Timer3	40, C.5
mega640, mega1280, mega2560	T100	Timer1	25, B.6*
		Timer1	26, B.7[1]
		Timer3	6, E.4

		Timer4	16, H.4
		Timer5	39, L.4
mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162	T32	Timer1	25, C.5*
mega16U4, mega32U4	T44	Timer1	30, B.6*
mega8515	P40	Timer1	29, E.2*
	T44	Timer1	26, E.2*
	L44	Timer1	32, E.2*
mega8535, mega163, mega323	P40	Timer1	18, D.4*
	T44	Timer1	13, D.4*
	L44	Timer1	19, D.4*
mega161	P40	Timer1	29, E.2*
	T44	Timer1	26, E.2*
mega162	P40	Timer1	29, E.2*
		Timer3	5, B.4
	T44	Timer1	26, E.2*
		Timer3	44, B.4
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P	T64	Timer1	16. B.6*
mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	T100	Timer1	25. B.6*
xmegaA1, xmegaA1U	T100	TimerC0	16, C.1
		TimerD0	26, D.1*
		TimerE0	36, E.1
		TimerE1	40, E.5
		TimerF0	46, F.1
		TimerF1	50, F.5
xmegaA3, xmegaA3B, xmegaA3B, xmegaA3BU	T64	TimerC0	17, C.1
		TimerD0	27, D.1*
		TimerE0	37, E.1
		TimerE1	41, E.5
		TimerF0	47, F.1
xmegaA4, xmegaA4U	T44	TimerC0	11, C.1
		TimerD0	21, D.1*
		TimerE0	29, E.1
xmegaD3	T64	TimerC0	17, C.1
		TimerD0	27, D.1*
		TimerE0	37, E.1
		TimerF0	47, F.1
xmegaD4	T44	TimerC0	11, C.1
		TimerD0	21, D.1*
		TimerE0	29, E.1

*Denotes the default OutputCapture pin.
[1]Requires the TIMER1_COMPC ISR and supports OutputCapture modulation.
[2]These devices support mapping of timer output compare pins; the pins listed are the default pins.

As noted in the two tables above, some devices support OutputCapture modulation on a dedicated pin. This capability allows the OutputCapture waveform to modulate the compare output of the Serial Timer. See the description of OutputCaptureEx for more information.

SPI Controllers

On some ZX devices, your program is stored in an external EEPROM that is read and written using the SPI interface. A dedicated I/O pin is required for selecting the EEPROM device during SPI operations and this I/O pin cannot be used for other purposes. However, the SPI bus itself can be used to communicate with other SPI devices. Although most SPI devices are tolerant of the ZX device using the SPI bus to fetch instructions from your program, some are not. Generally speaking, if you can send and

receive all of the data that an SPI device requires using a single call to SPICmd(), then that SPI device is usable with the ZX models that utilize an external EEPROM.

The table below indicates which ZX devices use an external EEPROM for user program storage and, if so, the I/O pin used for the chip select. For devices that do not use an external EEPROM for user program storage, the indicated chip select pin can be used for general purpose I/O with the proviso that if the SPI controller is used in the application program the chip select pin must either be an output or it must be held high during SPI transactions.

SPI EEPROM Usage and Control/Data Pins By Controller Index for ZX Devices

ZX Device	Uses SPI EEPROM	Ctrl. Idx.	CS Pin	SCK Pin	MOSI Pin	MISO Pin
ZX-24, ZX-24a, ZX-24p	Yes	0[1]	B.4	B.7	B.5	B.6
ZX-40, ZX-40a, ZX-40p	Yes	0	5, B.4	8, B.7	6, B.5	7, B.6
ZX-44, ZX-44a, ZX-44p	Yes	0	44, B.4	3, B.7	1, B.5	2, B.6
ZX-24x, ZX-24u	No	0[1]	D.4	D.7	D.5	D.6
		1	8, C.4	5. C.7	7, C.5	6, C.6
ZX-24n, ZX-24r, ZX-24s, ZX-24t	No	0[1]	B.4	B.7	B.5	B.6
ZX-40n, ZX-40r, ZX-40s, ZX-40t	No	0	5, B.4	8, B.7	6, B.5	7, B.6
ZX-44n, ZX-44r, ZX-44s, ZX-44t	No	0	44, B.4	3, B.7	1, B.5	2, B.6
ZX-328n, ZX-328l	No	0	16, B.2	19, B.5	17, B.3	18, B.4
ZX-32n, ZX-32l	No	0	14, B.2	17, B.5	15, B.3	16, B.4
ZX-1281, ZX-1281n	No	0	10, B.0	11, B.1	12, B.2	13, B.3
ZX-1280, ZX-1280n	No	0	19, B.0	20, B.1	21, B.2	22, B.3
ZX-32a4	No	0	24, D.4	27, D.7	25, D.5	26, D.6
		1	23, C.4	17, C.7	15, C.5	16, C.6
ZX-128a1	No	0	29, D.4	32, D.7	30, D.5	31, D.6
		1	19, C.4	22, C.7	20, C.5	21, C.6
		2	39, E.4	42, E.7	40, E.5	41, E.6
		3	49, F.4	52, F.7	50, F.5	51, F.6
ZX-24e, ZX-24ae, ZX-24pe, ZX-24pu	Yes	0	24, B.4	21, B.7	23, B.5	22, B.6
ZX-24ne, ZX-24nu, ZX-24ru, ZX-24su	No	0	24, B.4	21, B.7	23, B.5	22, B.6
ZX-128e, ZX-128ne, ZX-1281e, ZX-1281ne	No	0	28, B.0	27, B.1	26, B.2	25, B.3
ZX-328nu	No	0	13, B.2	16, B.5	14, B.3	15, B.4
ZX-24xu	No	0	16, D.4	13, D.7	15, D.5	14, D.6
		1	8, C.4	5. C.7	7, C.5	6, C.6

[1] The SPI pins are found along the edge of the board between pins 1 and 24

For generic target devices, user programs are always stored in internal Flash memory. The table below shows the chip select pin associated with each on-chip SPI controller as well as the SPI control/data pins. As described above, the SPI chip select pin(s) may be used for general purpose I/O except that if the related SPI controller is used in the application program the chip select pin must either be an output or it must be held high during SPI transactions

SPI Control/Data Pins By Controller Index for Generic Target Devices

Target Device	Pkg.	Ctrl. Idx.	CS Pin	SCK Pin	MOSI Pin	MISO Pin
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A, tiny2313, tiny2313A, tiny4313, tiny1634	all	-	-			
tiny48, tiny88	P28	0	16, B.2	19, B.5	17. B.3	18, B.4
	T28		12, B.2	15, B.5	13. B.3	14, B.4
	T32		14, B.2	17, B.5	15. B.3	16, B.4
tiny441, tiny841	S14	0	6, A.7	9, A.4	7, A.6	8, A.5
	Q20		15, A.7	1, A.4	16, A.6	20, A.5
tiny87, tiny167	S20	0	9, A.6	8, A.5	7, A.4	3, A.2
	T32		11, A.6	10, A.5	9, A.4	31, A.2
tiny828	T32	0	31, C.0	30, D.3	27, D.0	28, D.1

Devices	Pkg	#				
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	P28	0	16, B.2	19, B.5	17, B.3	18, B.4
	T28		12, B.2	15, B.5	13, B.3	14, B.4
	T32		14, B.2	17, B.5	15, B.3	16, B.4
(mega328PB only)	T32	1	19, E.2	24, C.1	22, E.3	23, C.0
mega16, mega16A, mega164A, mega164P, mega164PA, mega32, mega32A, mega324P, mega324PA, mega644, mega644A, mega644P, mega644PA, mega1284P, mega161, mega162, mega163, mega323	P40	0	5, B.4	8, B.7	6, B.5	7, B.6
	T44		44, B.4	3, B.7	1, B.5	2, B.6
mega64, mega64A, mega128, mega128A, mega1281, mega2561, AT90CAN32, AT90CAN64, AT90CAN128	T64	0	10, B.0	11, B.1	12, B.2	13, B.3
mega640, mega1280, mega2560	T100	0	19, B.0	20, B.1	21, B.2	22, B.3
mega8U2, mega16U2, mega32U2 AT90USB82, AT90USB162	T32	0	14, B.0	15, B.1	16, B.2	17, B.3
mega16U4, mega32U4	T44	0	8, B.0	9, B.1	10, B.2	11, B.3
mega8515, mega8535	P40	0	5, B.4	8, B.7	6, B.5	7, B.6
	T44		44, B.4	8, B.7	6, B.5	7, B.6
	L44		6, B.4	9, B.7	7, B.5	8, B.6
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P	T64	0	10, B.0	11, B.1	12, B.2	13, B.3
mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	T100	0	19, B.0	20, B.1	21, B.2	22, B.3
AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	T64	0	10, B.0	11, B.1	12, B.2	13, B.3
xmegaA1, xmegaA1U	T100	0	29, D.4	32, D.7	30, D.5	31, D.6
		1	19, C.4	22, C.7	20, C.5	21, C.6
		2	39, E.4	42, E.7	40, E.5	41, E.6
		3	49, F.4	52, F.7	50, F.5	51, F.6
xmegaA3, xmegaA3B, xmegaA3U, xmegaA3BU	T64	0	30, D.4	33, D.7	31, D.5	32, D.6
		1	20, C.4	23, C.7	21, C.5	22, C.6
		2	40, E.4	43, E.7	41, E.5	42, E.6
xmegaA4, xmegaA4U	T44	0	24, D.4	27, D.7	25, D.5	26, D.6
		1	23, C.4	17, C.7	15, C.5	16, C.6
xmegaD3	T64	0	30, D.4	33, D.7	31, D.5	32, D.6
		1	20, C.4	23, C.7	21, C.5	22, C.6
xmegaD4	T44	0	24, D.4	27, D.7	25, D.5	26, D.6
		1	23, C.4	17, C.7	15, C.5	16, C.6

I2C Controllers

For the available hardware I2C channels, the table below gives the pin numbers used for SDA and SCL.

SCL and SDA Pins by Channel for ZX Devices

ZX Device	Chan.	SCL	SDA
ZX-24, ZX-24a, ZX-24p, ZX-24n, ZX-24r, ZX-24s, ZX-24t	0	12, C.0	11, C.1
ZX-40, ZX-40a, ZX-40p, ZX-40n, ZX-40r, ZX-40s, ZX-40t	0	22, C.0	23, C.1
ZX-44, ZX-44a, ZX-44p, ZX-44n, ZX-44r, ZX-44s, ZX-44t	0	19, C.0	20, C.1
ZX-24e, ZX-24ae, ZX-24ne, ZX-24pe, ZX-24nu, ZX-24pu, ZX-24ru, ZX-24su	0	12, C.0	11, C.1
ZX-328n, ZX-328l, ZX-32n, ZX-32l	0	28, C.5	27, C.4
ZX-1281, ZX-1281n	0	25, D.0	26, D.1
ZX-1280, ZX-1280n	0	43, D.0	44, D.1
ZX-24x, ZX-24u	0	11, C.1	12, C.0
	1	17, E.1	25, E.0
ZX-32a4	0	11, C.1	10, C.0
	1	29, E.1	28, E.0
ZX-128a1	0	16, C.1	15, C.0
	1	36, E.1	35, E.0
	2	26, D.1	25, D.0
	3	46, F.1	45, F.0
ZX-128e, ZX-128ne, ZX-1281e, ZX-1281ne	0	12, D.0	11, D.1
ZX-328nu	0	22, C.5	21, C.4
ZX-24xu	0	11, C.1	12, C.0
	1	23, E.1	24, E.0

It is important to note that on the ZX-24n, ZX-24p, ZX-24r, ZX-24s and ZX-24t, the hardware I2C channel cannot be used while Com2 is open since pin 11 is shared by the SDA signal and TxD for Com2.

SCL and SDA Pins by Channel for Generic Target Devices

Target Device	Pkg.	Chan.	SCL	SDA
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A, tiny87, tiny167, tiny2313, tiny2313A, tiny4313, tiny828, tiny441, tiny841, tiny1634	all	-	-	-
tiny48, tiny88	P28	0	28, C.5	27, C.4
	T28	0	24, C.5	23, C.4
	T32	0	28, C.5	27, C.4
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	P28	0	28, C.5	27, C.4
	T28	0	28, C.5	27, C.4
	T32	0	28, C.5	27, C.4
(mega328PB only)	T32	1	6, E.1	3, E.0
mega16, mega16A, mega164A, mega164P, mega164PA, mega32, mega32A, mega324P, mega324PA, mega644, mega644A, mega644P, mega644PA, mega1284P, mega8535, mega163, mega323	P40	0	22, C.0	23, C.1
	T44	0	19, C.0	20, C.1
mega64, mega64A, mega128, mega128A, mega1281, mega2561, AT90CAN32, AT90CAN64, AT90CAN128, AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	T64	0	25, D.0	26, D.1
mega8U2, mega16U2, mega32U2 AT90USB82, AT90USB162, mega8515, mega161, mega162	T32	-	-	-
mega16U4, mega32U4	T44	0	18, D.0	19, D.1
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA,	T64	0	6, E.4	7, E.5

mega329, mega329P, mega329PA, mega649, mega649A, mega649P				
mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	T100	0	6, E.4	7, E.5
mega640, mega1280, mega2560	T100	0	43, D.0	44, D.1
xmegaA1, xmegaA1U	T100	0	16, C.1	15, C.0
		1	36, E.1	35, E.0
		2	26, D.1	25, D.0
		3	46, F.1	45, F.0
xmegaA3, xmegaA3U, xmegaA3B, xmegaA3BU, xmegaD3	T64	0	16, C.1	15, C.0
		1	36, E.1	35, E.0
xmegaA4, xmegaA4U	T44	0	11, C.1	10, C.0
		1	29, E.1	28, E.0
xmegaD4	T44	0	11, C.1	10, C.0
		1	29, E.1	28, E.0

Analog-to-Digital Converters

Most ZBasic target devices have a multiple analog inputs. These inputs may be fed to an internal analog-to-digital converter (ADC) or they may be used to perform analog level comparisons. The I/O port containing the analog inputs varies by target device as indicated in the table below. The System Library routines `GetADC()` and `ADCtoCom1()` use the ADC. The analog comparator is used by `WaitForInterrupt()` when configured to await an analog comparator event.

Analog Input Pins for ZX Devices

ZX Device	Analog Inputs
ZX-24, ZX-24a, ZX-24p, ZX-24n, ZX-24r, ZX-24s, ZX-24t	13-20
ZX-40, ZX-40a, ZX-40p, ZX-40n, ZX-40r, ZX-40s, ZX-40t	33-40
ZX-44, ZX-44a, ZX-44p, ZX-44n, ZX-44r, ZX-44s, ZX-44t	30-37
ZX-24x, ZX-24u	13-20, B.0-B.3
ZX-328n, ZX-328l	23-28
ZX-32n, ZX-32l	23-28, 19, 22
ZX-1281, ZX-1281n	54-61
ZX-1280, ZX-1280n	82-89, 90-97
ZX-32a4	40-44, 1-3, 4-7
ZX-128a1	95-100, 1-2, 5-12
ZX-24e, ZX-24ae, ZX-24pe, ZX-24ne	29-36
ZX-24pu, ZX-24nu, ZX-24ru, ZX-24su	29-36
ZX-128e, ZX-128ne, ZX-1281e, ZX-1281ne	54-61
ZX-24xu	29-36, 25-28
ZX-328nu	17-24

Analog Input Pins for Generic Target Devices

Target Device	Pkg.	Analog Inputs
tiny2313, tiny2313A, tiny4313, mega8515, mega161, mega162, mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162	all	-
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A	P14	6-13
	Q20	1-5, 15, 16, 20
tiny441, tiny841	S14	2-13
	Q20	1-5, 11-16, 20
tiny48, tiny88	P28	23-28
	T28	19-24
	T32	23-28, 19, 22
tiny87, tiny167	S20	1-4, 7-10, 12, 13
	T32	29-31, 3, 9-12, 15,

Device	Package	Pins
		18, 19
tiny828	T32	1-3, 6-17, 19, 20, 22-32
tiny1634	S20	1-6, 15-20
	Q20	1-4, 13-18
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	P28	23-28
	T28	19-24
	T32	23-28, 19, 22
mega16, mega16A, mega164A, mega164P, mega164PA, mega32, mega32A, mega324P, mega324PA, mega644, mega644A, mega644P, mega644PA, mega1284P	P40	33-40
	T44	40-47
mega64, mega64A, mega128, mega128A, mega1281, mega2561, AT90CAN32, AT90CAN64, AT90CAN128	T64	54-61
mega640, mega1280, mega2560	T100	82-89, 90-97
mega16U4, mega32U4	T44	36-41
mega8535, mega163, mega323	P40	33-40
	T44	30-37
	L44	36-43
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P	T64	54-61
mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	T100	90-97
AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	T64	54-61
xmegaA1, xmegaA1U	T100	95-100, 1-2, 5-12
xmegaA3, xmegaA3U, xmegaA3B, xmegaA3BU	T64	62-64, 1-5, 6-13
xmegaA4, xmegaA4U	T44	40-44, 1-3, 4-7
xmegaD3	T64	62-64, 1-5
xmegaD4	T44	40-44, 1-3

Digital-to-Analog Converters

The table below indicates the available channels and the corresponding DAC hardware used.

Supported DAC Channels for ZX Devices

ZX Devices	DACB	DACA
ZX-24x, ZX-24u	Chan 1, Pin 8 Chan 2, Pin 9	-
ZX-32a4	Chan 1, Pin 6 (B.2) Chan 2, Pin 7 (B.3)	-
ZX-128a1	Chan 1, Pin 7 (B.2) Chan 2, Pin 8 (B.3)	Chan 3, Pin 97 (A.2) Chan 4, Pin 98 (A.3)
ZX-24xu	Chan 1, Pin 26 (B.2) Chan 2, Pin 25 (B.3)	-

Supported DAC Channels for Generic Target Devices

ZX Devices	DACB	DACA
xmegaA1, xmegaA1U	Chan 1, Pin 7 (B.2) Chan 2, Pin 8 (B.3)	Chan 3, Pin 97 (A.2) Chan 4, Pin 98 (A.3)
xmegaA3, xmegaA3B, xmegaA3U, xmegaA3BU	Chan 1, Pin 7 (B.2) Chan 2, Pin 8 (B.3)	-
xmegaA4, xmegaA4U	Chan 1, Pin 6 (B.2)	-

	Chan 2, Pin 7 (B.3)	
xmega16D3, xmegaD4	-	-

Note, particularly, that the ATxmega can produce two analog outputs from a single DAC. In the table above, channels 1 and 2 are the two outputs from one DAC and channels 3 and 4 are the two outputs from the second DAC (if available). In order to use the second channel on a given DAC, the first channel must have been opened in dual output mode (see the mode details in the description of OpenDAC). Also note that using both outputs from a DAC will result in analog levels with significantly more noise due to the sample-and-hold and automatic refresh circuitry employed. For this reason, it is generally recommended to use single output per DAC.

Interrupts in General

Some of the System Library routines disable interrupts in order to achieve the precise timing that is required. Having interrupts disabled for long periods of time can interfere with the operation of other parts of the system that use interrupts like task management, serial I/O and the real time clock. In most cases, the System Library routines have been implemented to keep track of real time clock interrupts that should have occurred during the time interrupts are disabled and then the RTC is updated at the end of the operation. This strategy avoids the problem of the RTC losing time. However, there is a limit to the amount of time that missed RTC timer interrupts can be accurately tracked, that limit being 65535 divided by the RTC fast tick frequency. See Section 3 for more information about the RTC fast tick frequency.

Unfortunately, there is no way to similarly protect the serial I/O process. You can reduce the impact of having interrupts disabled with respect to serial output by ensuring that all serial output queues are empty before calling a System Library routine that disables interrupts. This is not as critical for a hardware-based serial channel (e.g. Com1) as it is for the software-based serial channels Com3 to Com6. There is no way, however, to work around the problem of serial input data arriving while interrupts are disabled. The hardware-based serial channels will store one received character and hold it while interrupts are disabled but if a second character arrives while interrupts are disabled it will be lost. Channels 3-6 rely on interrupts for every bit received so the situation is much more problematic. In this case, having interrupts disabled for longer than approximately one-third of the bit time will likely cause garbled input if a character's transmit time overlaps the period when interrupts are disabled. For characters being transmitted by channels 3-6, having interrupts disabled for more than about 10% of the bit time may cause the receiver to lose synchronization.

For reference purposes, the table below indicates which I/O routines disable interrupts for the duration of their execution. See the individual descriptions for more detailed information.

System Library Routines that Disable Interrupts

```
CountTransitions    I2CPutByte       PutPin
DACPin              PlaySound        RCTime
FreqOut             PulseIn          Reset1Wire
Get1Wire            PulseOut         ShiftIn
Get1WireByte        Put1Wire         ShiftInEx
Get1WireData        Put1WireByte     ShiftOut
I2CCmd              Put1WireData     ShiftOutEx
I2CGetByte          PutDAC
```

The I2C routines do not disable interrupts when the hardware I2C controller is used (e.g. channel 0).

External Interrupts

ATtiny and ATmega target devices (and ZX devices based on them) support a varying number of external interrupt inputs. (External interrupts are not available on any ATxmega devices.) The table below gives the available external interrupt input pins for ZX devices.

ZX Device	Ext. Int.	Pin	Ext. Int.	Pin
ZX-24, ZX-24a, ZX-24p, ZX-24n, ZX-24r, ZX-24s, ZX-24t	INT0	6, D.2	INT1	11, D.3
	INT2	18, B.2		
ZX-40, ZX-40a, ZX-40p, ZX-40n, ZX-40r, ZX-40s, ZX-40t	INT0	16, D.2	INT1	17, D.3
	INT2	3, B.2		
ZX-44, ZX-44a, ZX-44p, ZX-44n, ZX-44r, ZX-44s, ZX-44t	INT0	11, D.2	INT1	12, D.3
	INT2	42, B.2		
ZX-328n, ZX-328l	INT0	4, D.2	INT1	5, D.3
ZX-32n, ZX-32l	INT0	32, D.2	INT1	1, D.3
ZX-1281, ZX-1281n	INT0	25, D.0	INT1	26, D.1
	INT2	27, D.2	INT3	28, D.3
	INT4	6, E.4	INT5	7, E.5
	INT6	8, E.6	INT7	9, E.7
ZX-1280, ZX-1280n	INT0	43, D.0	INT1	44, D.1
	INT2	45, D.2	INT3	46, D.3
	INT4	6, E.4	INT5	7, E.5
	INT6	8, E.6	INT7	9, E.7
ZX-24e, ZX-24ae, ZX-24pe, ZX-24ne, ZX-24pu, ZX-24nu, ZX-24ru, ZX-24su	INT0	18, D.2	INT1	17, D.3
	INT2	26, B.2		
ZX-128e, ZX-128ne, ZX-1281e, ZX-1281ne	INT0	12, D.0	INT1	11, D.1
	INT2	10, D.2	INT3	9, D.3
	INT4	16, E.4	INT5	15, E.5
	INT6	14, E.6	INT7	13, E.7
ZX-328nu	INT0	5, D.2	INT1	6, D.3

The table below gives the available external interrupt input pins for generic target devices. Note that external interrupts are not available on xmega devices.

External Interrupt Pins for Generic ATtiny and ATmega Targets

Target Device	Pkg.	Ext. Int.	Pin
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A	P14	INT0	5, B.2
	Q20	INT0	14, B.2
tiny48, tiny88	P28	INT0	4, D.2
		INT1	5, D.3
	T28	INT0	28, D.2
		INT1	1, D.3
	T32	INT0	32, D.2
		INT1	1, D.3
tiny441, tiny841	S14	INT0	3, B.1
	Q20	INT0	12, B.1
tiny87, tiny167	S20	INT0	12, B.6
		INT1	4, A.3
	T32	INT0	15, B.6
		INT1	3, A.3
tiny828	T32	INT0	32, C.1
		INT1	1, C.2
tiny1634	S20	INT0	15, C.2
	Q20	INT0	13, C.2
tiny2313, tiny2313A, tiny4313	P20	INT0	6, D.2
		INT1	7, D.3
	Q20	INT0	4, D.2
		INT1	5, D.3
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	P28	INT0	4, D.2
		INT1	5, D.3

	T32	INT0	32, D.2
		INT1	1, D.3
mega16, mega16A, mega164A, mega164P, mega164PA, mega32, mega32A, mega324P, mega324PA, mega644, mega644A, mega644P, mega644PA, mega1284P, mega323, mega8535	P40	INT0	16, D.2
		INT1	17, D.3
		INT2	3, B.2
	T44	INT0	11, D.2
		INT1	12, D.3
		INT2	42, B.2
mega161, mega162, mega8515	P40	INT0	12, D.2
		INT1	13, D.3
		INT2	31, E.0
	T44	INT0	8, D.2
		INT1	9, D.3
		INT2	29, E.0
	L44	INT0	14, D.2
		INT1	15, D.3
		INT2	35, E.0
mega163	P40	INT0	12, D.2
		INT1	13, D.3
	T44	INT0	8, D.2
		INT1	9, D.3
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P	T64	INT0	26, D.1
mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	T100	INT0	44, D.1
mega1281, mega2561, AT90CAN32, AT90CAN64, AT90CAN128	T64	INT0	25, D.0
		INT1	26, D.1
		INT2	27, D.2
		INT3	28, D.3
		INT4	6, E.4
		INT5	7, E.5
		INT6	8, E.6
		INT7	9, E.7
mega640, mega1280, mega2560	T100	INT0	43, D.0
		INT1	44, D.1
		INT2	45, D.2
		INT3	46, D.3
		INT4	6, E.4
		INT5	7, E.5
		INT6	8, E.6
		INT7	9, E.7
mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162	T32	INT0	6, D.0
		INT1	7, D.1
		INT2	8, D.2
		INT3	9, D.3
		INT4	22, C.7
		INT5	10, D.4
		INT6	12, D.6
		INT7	13, D.7
mega16U4, mega32U4	T44	INT0	18, D.0
		INT1	19, D.1
		INT2	20, D.2
		INT3	21, D.3
		INT6	1, E.6
AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	T64	INT0	25, D.0
		INT1	26, D.1
		INT2	27, D.2
		INT3	28, D.3

		INT4	18, E.4
		INT5	19, E.5
		INT6	1, E.6
		INT7	2, E.7

[1]The interrupt input configuration is shared between INT0 and INT1. Consequently, if both are used at the same time the last configured will control the input configuration.

Pin Change Interrupts

The table below shows how ports are mapped to the four possible pin change interrupts on ATtiny and ATmega devices. See the description of WaitForInterrupt for more information on preparing to await a pin change interrupt.

ATtiny and ATmega Pin Change Interrupt Support

Target Device	WaitForInterrupt `intNum`	PinChange Interrupt	Port Pins
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A, tiny441, tiny841	WaitPinChangeA	PCINT0	A.7-A.0
	WaitPinChangeB	PCINT1	B.3-B.0
tiny48, tiny88	WaitPinChangeB	PCINT0	B.7-B.0
	WaitPinChangeC	PCINT1	C.7-C.0
	WaitPinChangeD	PCINT2	D.7-D.0
	WaitPinChangeA	PCINT3	A.3-A.0
tiny87, tiny167	WaitPinChangeA	PCINT0	A.7-A.0
	WaitPinChangeB	PCINT1	B.7-B.0
tiny2313, tiny2313A	WaitPinChangeB	PCINT0	B.7-B.0
tiny4313	WaitPinChangeB	PCINT0	B.7-B.0
	WaitPinChangeA	PCINT1	A.3-A.0
	WaitPinChangeD	PCINT2	D.6-D.0
tiny828	WaitPinChangeA	PCINT0	A.7-A.0
	WaitPinChangeB	PCINT1	B.7-B.0
	WaitPinChangeC	PCINT2	C.7-C.0
	WaitPinChangeD	PCINT3	D.3-D.0
tiny1634	WaitPinChangeA	PCINT0	A.7-A.0
	WaitPinChangeB	PCINT1	B.3-B.0
	WaitPinChangeC	PCINT2	C.3-C.0
mega8, mega8A, mega16, mega16A, mega32, mega32A, mega64, mega64A, mega128, mega128A, mega8515, mega8535, mega161, mega163, mega323, AT90CAN32, AT90CAN64, AT90CAN128	-	-	-
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	WaitPinChangeB	PCINT0	B.7-B.0
	WaitPinChangeC	PCINT1	C.6-C.0
	WaitPinChangeD	PCINT2	D.7-D.0
(mega328PB only)	WaitPinChangeE	PCINT3	E.3-E.0
mega164A, mega164P, mega164PA, mega324P, mega324PA, mega644, mega644A, mega644P, mega644PA, mega1284P	WaitPinChangeA	PCINT0	A.7-A.0
	WaitPinChangeB	PCINT1	B.7-B.0
	WaitPinChangeC	PCINT2	C.7-C.0
	WaitPinChangeD	PCINT3	D.7-D.0
mega1281, mega2561	WaitPinChangeB	PCINT0	B.7-B.0
	WaitPinChangeE	PCINT1	E.0
mega640, mega1280, mega2560	WaitPinChangeB	PCINT0	B.7-B.0
	WaitPinChangeJ	PCINT1	J.7-J.0
	WaitPinChangeK	PCINT2	K.7-K.0
mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162	WaitPinChangeB	PCINT0	B.7-B.0
	WaitPinChangeC	PCINT1	D.5, C.2, C.4, C.5, C.6
mega16U4, mega32U4,	WaitPinChangeB	PCINT0	B.7-B.0

AT90USB646, AT90USB647, AT90USB1286, AT90USB1287			
mega162	WaitPinChangeA	PCINT0	A.7-A.0
	WaitPinChangeC	PCINT1	C.7-C.0
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P	WaitPinChangeE	PCINT0	E.7-E.0
	WaitPinChangeB	PCINT1	B.7-B.0
mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	WaitPinChangeE	PCINT0	E.7-E.0
	WaitPinChangeB	PCINT1	B.7-B.0
	WaitPinChangeH	PCINT2	H.7-H.0
	WaitPinChangeJ	PCINT3	J.6-J.0

The table below shows how ports are mapped to the four possible pin change interrupts on ATxmega devices. Note that for each port, there are two independent channels available. See the description of WaitForInterrupt for more information on preparing to await a pin change interrupt.

ATxmega Pin Change Interrupt Support

Processors	WaitForInterrupt intNum	Pin Change Trigger	Interrupt
xmegaA1, xmegaA1U, xmegaA3, xmegaA3U, xmegaA3B, xmegaA3BU, xmegaA4, xmegaA4U, xmegaD3, xmegaD4	WaitPinChangeA0	Port A, channel 0	PORTA_INT0
	WaitPinChangeA1	Port A, channel 1	PORTA_INT1
	WaitPinChangeB0	Port B, channel 0	PORTB_INT0
	WaitPinChangeB1	Port B, channel 1	PORTB_INT1
	WaitPinChangeC0	Port C, channel 0	PORTC_INT0
	WaitPinChangeC1	Port C, channel 1	PORTC_INT1
	WaitPinChangeD0	Port D, channel 0	PORTD_INT0
	WaitPinChangeD1	Port D, channel 1	PORTD_INT1
	WaitPinChangeE0	Port E, channel 0	PORTE_INT0
	WaitPinChangeE1	Port E, channel 1	PORTE_INT1
xmegaA1, xmegaA1U, xmegaA3, xmegaA3U, xmegaA3B, xmegaA3BU, xmegaD3	WaitPinChangeF0	Port F, channel 0	PORTF_INT0
	WaitPinChangeF1	Port F, channel 1	PORTF_INT1
xmegaA1, xmegaA1U	WaitPinChangeH0	Port H, channel 0	PORTH_INT0
	WaitPinChangeH1	Port H, channel 1	PORTH_INT1
	WaitPinChangeJ0	Port J, channel 0	PORTJ_INT0
	WaitPinChangeJ1	Port J, channel 1	PORTJ_INT1
	WaitPinChangeK0	Port K, channel 0	PORTK_INT0
	WaitPinChangeK1	Port K, channel 1	PORTK_INT1
	WaitPinChangeQ0	Port Q, channel 0	PORTQ_INT0
	WaitPinChangeQ1	Port Q, channel 1	PORTQ_INT1

Analog Comparator Interrupts

The table below shows analog comparator input pins for ZX devices. See the description of WaitForInterrupt for more information on preparing to await an analog comparator interrupt.

Analog Comparator Input Pins for ZX Devices

ZX Device	AIN0 Pin	AIN1 Pin
ZX-24, ZX-24a, ZX-24p, ZX-24n, ZX-24r, ZX-24s, ZX-24t	18, B.2	19, B.3
ZX-40, ZX-40a, ZX-40p, ZX-40n, ZX-40r, ZX-40s, ZX-40t	3, B.2	4, B.3
ZX-44, ZX-44a, ZX-44p, ZX-44n, ZX-44r, ZX-44s, ZX-44t	42, B.2	43, B.3
ZX-328n, ZX-328l	12, D.6	13, D.7
ZX-32n, ZX-32l	10, D.6	11, D.7
ZX-1281, ZX-1281n, ZX-1280, ZX-1280n	4, E.2	5, E.3
ZX-24e, ZX-24ae, ZX-24pe, ZX-24ne, ZX-24pu, ZX-24nu, ZX-24ru, ZX-24su	26, B.2	25, B.3
ZX-128e, ZX-128ne , ZX-1281e, ZX-1281ne	18, E.2	17, E.3
ZX-328nu	9, D.6	10, D.7

On the ZX-24 models, AIN0 is common with pin A.2 and AIN1 is common with pin A.0 so these I/O pins will need to be configured to be inputs in high-impedance mode. Note, however, that ZX-24 models built using boards earlier than Rev 5 (see the bottom side of the board), AIN1 has no external connection so the negative input must be supplied via the analog multiplexor.

The table below shows the analog comparator input pins for generic ATtiny and ATmega targets.

Analog Comparator Input Pins for Generic ATtiny and Atmega Devices

Target Device	Pkg.	AIN0 Pin	AIN1 Pin
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A	P14	12, A.1	11, A.2
	Q20	4, A.1	3, A.2
tiny48, tiny88	P28	12, D.6	13, D.7
	T28	8, D.6	9, D.7
	T32	10, D.6	11, D.7
tiny441, tiny841	S14	10, A.3	9, A.4
	Q20	2, A.3	1, A.4
tiny87, tiny167	S20	9, A.6	10, A.7
	T32	11, A.6	12, A.7
tiny2313, tiny2313A, tiny4313	P20	12, B.0	13, B.1
	Q20	10, B.0	11, B.1
tiny828	T32	10, A.1	11, A.2
tiny1634	S20	8, A.1	7, A.2
	Q20	6, A.1	5, A.2
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	P28	12, D.6	13, D.7
	T32	10, D.6	11, D.7
mega16, mega16A, mega164A, mega164P, mega164PA, mega32, mega32A, mega324P, mega324PA, mega644, mega644A, mega644P, mega644PA, mega1284P, mega161, mega162, mega163, mega323	P40	3, B.2	4, B.3
	T44	42, B.2	43, B.3
mega8515, mega8535	P40	3, B.2	4, B.3
	T44	42, B.2	43, B.3
	L44	4, B.2	5, B.3
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P	T64	4, E.2	5, E.3
mega3250, mega3250P, mega6450, mega6450A, mega6450P,	T100	4, E.2	5, E.3

mega3290, mega3290P, mega6490, mega6490A, mega6490P			
mega1281, mega2561, mega64, mega64A, mega128, mega128A, AT90CAN32, AT90CAN64, AT90CAN128	T64	4, E.2	5, E.3
mega640, mega1280, mega2560	T100	4, E.2	5, E.3
mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162	T32	7, D.1	8, D.2
AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	T64	1, E.6	2, E.7
mega16U4, mega32U4	T44	1, E.6	-

The table below shows the register containing the Analog Comparator Multiplexer Enable (ACME) bit. Where available, this bit can be used to allow the output of the analog input multiplexer to feed the negative input of the analog comparator. For target devices having n/a in the second column, this capaability is not present.

Register Containing the ACME Bit by Target Device

Target Device	ACME bit Register
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A, tiny48, tiny88, tiny87, tiny167	ADCSRB
tiny2313, tiny2313A, tiny4313, tiny828, tiny441, tiny841	n/a
tiny1634	ACSRB
mega8, mega8A, mega16, mega16A, mega32, mega32A, mega163, mega323, mega8535	SFIOR
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	ADCSRB
mega8515, mega161, mega162	n/a
mega164A, mega164P, mega164PA, mega324P, mega324PA, mega644, mega644A, mega644P, mega644PA, mega1284P	ADCSRB
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P	ADCSRB
mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	ADCSRB
mega64, mega64A, mega128, mega128A, AT90CAN32, AT90CAN64, AT90CAN128	SFIOR
mega1281, mega2561, mega640, mega1280, mega2560, mega16U4, mega32U4	ADCSRB
mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162	n/a
AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	ADCSRB

The table below shows the analog comparator input pins for ATxmega targets and ZX devices based on xmega chips.

ATxmega Analog Comparator Interrupt Support

Processors	WaitForInterrupt `intNum`	Interrupt Trigger	Interrupt

xmegaA1, xmegaA1U, xmegaA3, xmegaA3U, xmegaA3B, xmegaA3BU, xmegaA4, xmegaA4U, xmegaD3, xmegaD4	`waitAnalogCompA0`
	`waitAnalogCompA1`
	`waitAnalogCompAW`
		AC A, Channel 0
		AC A, Channel 1
		AC A, Window
			`ACA_AC0`
			`ACA_AC1`
			`ACA_ACW`

xmegaA1, xmegaA1U, xmegaA3, xmegaA3U, xmegaA3B, xmegaA3BU	`waitAnalogCompB0`
	`waitAnalogCompB1`
	`waitAnalogCompBW`
		AC B, Channel 0
		AC B, Channel 1
		AC B, Window
			`ACB_AC0`
			`ACB_AC1`
			`ACB_ACW`

Interrupt Service Routines

For the native code devices (e.g. ZX-24n), a few interrupt service routines (ISRs) are typically included in your program (e.g. for Com1 and the RTC if not specifically disabled) while others are included only if certain System Library routines are used in your program. In some cases, the additional ISRs that are included when a specific System Library routine is used depends on how the routine is invoked and what the compiler can deduce regarding which ISRs might be needed. For example, if OpenCom() is invoked

one or more times but the compiler can determine that the Com1 is always the channel being used, no additional ISRs are included since the Com1 ISRs are usually included anyway. On the other hand, if the compiler cannot determine which channel is being opened in one or more cases, it includes the ISRs for all Com channels, both hardware-based and software-based channels.

In the description of each System Library routine, information is given about the set of ISRs might be included in your program if you use that routine. This information is only important, of course, if you are also providing one or more ISRs in your code because conflicts may arise. (See the section entitled "Defining Interrupt Service Routines" in the ZBasic Language Reference Manual for more information on how this is done.) The table below gives an overview of which System Library routines may cause ISRs to be included atomically in your program.

System Library Routines that May Load ISRs

System Library Routine	ISR for ATtiny/ATmega	ISR for ATxmega
ADCtoCom1()	TIMER#_COMPA	n/a
Com1toDAC()	TIMER#_COMPA	n/a
InputCapture()	TIMER*_CAPT	TC*_CCA
	TIMER*_OVF	TC*_OVF
OutputCapture()	TIMER*_COMPB	TC*_CCB
OpenCom()	USART#_RX	USART#_RXC
	USART#_TX	USART#_TXC
	USART#_UDRE	USART#_UDRE
	TIMER&_COMPA	TC&_CCA
OpenX10()	INT*	ACA_AC0
	TIMER$_COMPB	TC$_CCB
WaitForInterrupt()	INT#	PORTx_INT#
	PCINT#	ACA_AC0
	ANALOG_COMP	ACA_AC1
		ACA_ACW
		ACB_AC0
		ACB_AC1
		ACB_ACW

In the table above some ISR names (shaded) are given symbolically in the interest of brevity, representing multiple possible ISR names. The table below describes how to interpret the symbolic ISR entries.

Key to Symbolic ISR Names

Symbolic ISR Name	Meaning
TIMER#_COMPA	Replace TIMER# with the name of the I/O Timer, e.g. TIMER1.
TIMER*_COMPB TIMER*_CAPT TIMER*_OVF	Replace TIMER* with the applicable 16-bit timer name, e.g. TIMER4.
TIMER&_COMPA	Replace TIMER& with the software UART timer name, e.g. TIMER2.
TIMER$_COMPB	Replace TIMER$ with the RTC timer name, e.g. TIMER0.
TC*_CCA TC*_CCB TC*_OVF	Replace TC* with the applicable timer name, e.g. TCC0.
TC&_CCA	Replace TC$ with the software UART timer name, e.g. TCC0.
TC$_CCB	Replace TC$ with the RTC timer name, e.g. TCC0.
USART#_RX USART#_TX USART#_UDRE	Replace USART# with the applicable UART name, e.g. USART0.
USART#_RXC USART#_TXC USART#_UDRE	Replace USART# with the applicable UART name, e.g. USARTC0.
INT#	Replace INT# with the external interrupt name, e.g. INT0.
INT*	Replace INT* with the X-10 zero-crossing interrupt name, e.g. INT0.
PCINT#	Replace PCINT# with the pin change interrupt name, e.g. PCINT0.
PORTx_INT#	Replace PORTx with the applicable port name, e.g. PORTA and replace

Program Memory Page Size

For ZBasic devices having Program Memory in internal Flash Memory, the page size of that memory is an important value. For example, for an application that writes to Program Memory a buffer of the length of the page size must be allocated from the heap in order to perform the necessary read-modify-write operation that is required for updating Flash Memory locations. This is one factor affecting the minimum heap size for a particular device and application. The tables below give the Program Memory page size for ZX devices and generic target devices.

Program Memory Page Size for ZX Devices

ZX Device	Page Size
ZX-24, ZX-24a, ZX-24p, ZX-40, ZX-40a, ZX-40p, ZX-44, ZX-44a, ZX-44p, ZX-24e, ZX-24ae, ZX-24pe, ZX-24pu	n/a
ZX-24n, ZX-40n, ZX-44n, ZX-24ne, ZX-24nu	256
ZX-24r, ZX-24s, ZX-24t, ZX-40r, ZX-40s, ZX-40t, ZX-44r, ZX-44s, ZX-44t, ZX-24ru, ZX-24su	256
ZX-328n, ZX-328l, ZX-32n, ZX-32l, ZX-328nu	128
ZX-1281, ZX-1281n, ZX-1280, ZX-1280n	256
ZX-24x, ZX-24u, ZX-32a4, ZX-24xu	256
ZX-128a1	512
ZX-128e, ZX-128ne, ZX-1281e, ZX-1281ne	256

Program Memory Page Size for Generic Target Devices

Target Device	Page Size
tiny2313, tiny2313A, tiny24, tiny24A, tiny1634[1]	32
tiny4313, tiny44, tiny44A, tiny84, tiny84A, tiny48, tiny88, tiny828, tiny441, tiny841[1]	64
tiny87, tiny167	128
mega48, mega48A, mega48P, mega48PA, mega48PB, mega8, mega8A, mega88, mega88A, mega88P, mega88PA, mega88PB	64
mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB, mega16, mega16A, mega164A, mega164P, mega164PA, mega32, mega32A, mega324P, mega324PA	128
mega644, mega644A, mega644P, mega644PA, mega1284P, mega64, mega64A, mega128, mega128A, mega1281, mega2561, mega640, mega1280, mega2560	256
AT90CAN32, AT90CAN64, AT90CAN128	256
mega8U2	64
mega16U2, mega32U2	128
mega16U4, mega32U4	256
mega8515, mega8535	64
mega161, mega162, mega163, mega323	128
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega3250, mega3250P, mega3290, mega3290P	128
mega645, mega645A, mega645P, mega649, mega649A, mega649P, mega6450, mega6450A, mega6450P, mega6490, mega6490A, mega6490P	256
AT90USB82, AT90USB162	128
AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	256
xmega with 64K or less of Flash memory	256
xmega with 128K or more of Flash memory	512

[1] For these devices, a block of 4 times the page size is required.

Section 3 - Processor Speed and Device Configuration Issues

For ZX devices, the processor speed and configuration are fixed and specific to each device. The clock speed for most ATmega-based ZX devices is 14.7456MHz but special versions are available that run slower and faster. ZX devices based on the ATxmega run at 29.4912MHz. The first table below summarizes the differences that arise due to the difference in operating speeds. The second table summarizes the differences at reduced operating speeds obtained by using the clock prescaler directive.

ZX Device Processor Speed Variations

Parameter	7.3728 MHz	14.7456 MHz	18.432 MHz	29.4912 MHz
RTC Tick Frequency	512 Hz	512 Hz	500 Hz	512 Hz
RTC Fast Tick Frequency	512 Hz	1024 Hz	1000 Hz	512 Hz
RTC Scale Factor	1	2	2	1
RTC Timer Frequency	115.2 KHz	230.4 KHz	72 KHz	115.2 KHz
Multi-tasking Time Slice	1.95 mS	1.95 mS	2.0 mS	1.95 mS
Default `TimerSpeed1` Units	135.6 nS	67.8 nS	54.4 nS	67.8 nS
Default `TimerSpeed2` Units	1.085 µS	1.085 µS*	434 nS	271 nS
`CountTransitions()` Sample Rate	204.8 KHz	409.6 KHz	512 KHz	737.3 KHz
Maximum SW serial baud rate	9600	19200	19200	19200

Parameter	460.8 KHz	921.6 KHz	1.8432 MHz	3.6864 MHz
RTC Tick Frequency	400 Hz	512 Hz	400 Hz	400 Hz
RTC Fast Tick Frequency	400 Hz	512 Hz	400 Hz	400 Hz
RTC Scale Factor	1	1	1	1
RTC Timer Frequency	7.2 KHz	115.2 KHz	7.2 KHz	3.6 KHz
Multi-tasking Time Slice	2.5 mS	1.95 mS	2.5 mS	2.5 mS
Default `TimerSpeed1` Units	2.17 µS	1.085 µS	542.5 nS	271.3 nS
Default `TimerSpeed2` Units	17.36 µS	8.68 µS	4.34 µS	2.17 µS
`CountTransitions()` Sample Rate	12.8 KHz	25.6 KHz	51.2 KHz	102.4 KHz
Maximum SW serial baud rate	600	1200	2400	4800

Note, particularly, that the "units" value for `TimerSpeed2` on ZX devices running at 14.7456MHz is scaled to match the value corresponding to operating at 7.3728MHz. This is done for compatibility with BasicX devices that all operate at the lower speed. Consult the section I/O Timer Prescaler Values for information on which routines use the `TimerSpeed1` and `TimerSpeed2` values. The scaling effect described above can be disabled by setting the value of `Register.IOScaling` to `False`.

For generic target devices, the processor speed can be any reasonable value and the RTC frequency can be any value attainable using the available prescaler and compare value settings for the RTC timer. Further, the initial I/O Timer prescaler settings Register.TimerSpeed1 and Register.TimerSpeed2 can be any useful values. Consquently, the meaning of results returned by some ZBasic System Library routines can only be described in terms of the value of these configurable items (all of which are specified at compile time).

The table below gives several important values that are dependent on device configuration parameter values both in the case of ZX devices (with fixed values) or generic target devices (with user-specified values). The symbols given in the table entries is used in the descriptions of various ZBasic System Library routines thus allowing you to infer the meaning of the results based on device configuration values.

Device Parameter	Symbol	Example Value for ZX-24n
Main Clock Frequency	F_CPU	14.7456 MHz
RTC Scale Factor	RTC_SCALE	2
RTC Fast Tick Frequency	F_RTC_FAST	1024 Hz
RTC Tick Frequency	F_RTC_TICK	512 Hz
RTC Timer Frequency	F_RTC_TIMER	230.4 KHz
TimerSpeed1 Frequency	F_TS1	14.7456 MHz[1]
TimerSpeed2 Frequency	F_TS2	1.8432 MHz[2]

1) Assuming the default Register.TimerSpeed1 value of 1.
2) Assuming the default Register.TimerSpeed2 value of 2.

Main Clock Frequency (F_CPU)

This value represents the operating speed of the target CPU. In the case of generic target devices, it is specified via the target device parameter ClockFrequency.

RTC Scale Factor (RTC_SCALE)

This value, limited to being 1 or 2, represents a scale factor for mapping RTC timer compare interrupts to Register.RTCFastTick and Register.RTCTick updates. If RTC_SCALE is 1, Register.RTCFastTick and Register.RTCTick change at the same rate. If RTC_SCALE is 2, Register.RTCFastTick changes at twice the rate as Register.RTCTick.

RTC Fast Tick Frequency (F_RTC_FAST)

This value represents the rate of change of Register.RTCFastTick which is updated on every RTC timer interrupt. The rate of change is equal to RTC_SCALE times the rate of change of Register.RTCTick.

RTC Tick Frequency (F_RTC_TICK)

This value represents the rate of change of Register.RTCTick which is equal to the rate of change of Register.RTCFastTick divided by RTC_SCALE.

RTC Timer Frequency (F_RTC_TIMER)

This value represents the rate of change of the counting register of the RTC timer and its value is a fraction of F_CPU determined by the RTC timer prescaler setting. For example, if the RTC timer prescaler setting indicates a divide-by-64 prescaler, F_RTC_TIMER will be F_CPU / 64. For generic target devices, the compiler computes the prescaler divisor based on the specified ClockFrequency, RTCFrequency and RTCScale configuration parameters using the smallest available prescaler setting given the maximum compare register value for the particular device.

TimerSpeed1 Frequency (F_TS1)

This value represents the rate of change of the counting register of the I/O timer and its value is computed by dividing F_CPU by the prescaler value selected by the value of Register.TimerSpeed1. For generic target devices, the initial value of Register.TimerSpeed1 is implied by the configuration parameter TimerSpeed1Divisor. See the section I/O Timer Prescaler Values for information on the relationship of prescaler selector values to divisor values.

TimerSpeed2 Frequency (F_TS2)

This value represents the rate of change of the counting register of the I/O timer and its value is computed by dividing F_CPU by the prescaler value selected by the value of Register.TimerSpeed2. For generic target devices, the initial value of Register.TimerSpeed2 is implied by the configuration parameter TimerSpeed2Divisor. See the section I/O Timer Prescaler Values for information on the relationship of prescaler selector values to divisor values.

It is highly recommended to use the built-in values `Register.CPUFrequency`, `Register.RTCTickFrequency`, and `Register.RTCTimerFrequency` in your application code instead of using hard-coded values. Doing so also simplifies code that must run on multiple devices that operate at different speeds. See the descriptions of these values in the ZBasic Language Reference Manual for more details.

Section 4 - Detailed Descriptions of Subroutines and Functions

In the descriptions that follow, the parameter types that are accepted by each routine are described. Some parameters accept a specific fundamental data type while others may accept a few similar types. Others accept virtually any parameter type. In order to more succinctly describe the types of parameters accepted, some descriptive type categories are used. For example, the category *integral* is used to connote those types that have the integral characteristic, such as `Byte`, `Integer`, `UnsignedInteger`, `Long` and `UnsignedLong`. The table below indicates which types belong to which categories.

Type Category Membership

Type/Category	any type	numeric	integral	signed	int8/16	int16	int32	any 32-bit
Boolean	x							
Bit	x	x	x		x			
Nibble	x	x	x		x			
Byte	x	x	x		x			
Integer	x	x	x	x	x	x		
UnsignedInteger	x	x	x		x	x		
Long	x	x	x	x			x	x
UnsignedLong	x	x	x				x	x
Single	x	x		x				x
Enum	x							
String	x							

The remainder of this document presents complete descriptions of each of the System Library routines, arranged in alphabetical order. Unless specifically noted otherwise, the descriptions apply to all ZBasic devices. In some cases, a routine exhibits different behavior in BasicX compatibility mode or operates in a manner that is slightly different from that implemented in the BasicX environment. In these cases, the heading **Compatibility** will appear in the description detailing the differences. The advanced System Library routines that are not present in the BasicX environment are also similarly noted. If you are not using BasicX compatibility mode or are not upgrading BasicX code these notations may be safely ignored.

Abs

Type   Function returning the same type as the parameter

Invocation   Abs(arg)

Parameter	Method	Type	Description
arg	ByVal	numeric	The value from which the absolute value will be computed.

Discussion

The absolute value function returns the magnitude of the passed value. It is primarily useful for signed numeric types such as `Single`, `Integer` and `Long`. Unsigned parameter values will be returned unchanged.

The type of the return value will be the same as the type of the parameter provided.

Example

```
Dim i as Integer, j as Integer

i = -45
j = Abs(i)  ' result is 45
```

Acos

Type   Function returning Single

Invocation   Acos(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value from which the arc cosine will be computed.

Discussion

The arc cosine function is the inverse of the cosine function. The return value will be the angle, expressed in radians, whose cosine corresponds to the passed value. The type of the return value will be `Single` and the value will range from 0.0 to   . If the argument is greater than 1.0 or less than −1.0, the result will be undefined.

Example

```
Dim val as Single, theta as Single

val = 0.5
theta = Acos(val)  'the result will be approximately 1.0472.
```

See Also   Cos, DegToRad, RadToDeg

ADCtoCom1

Type   Subroutine

Invocation   ADCtoCom1(pin, rate)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin number from which analog readings will be taken. Valid pins are those corresponding to PortA, pins 13 to 20.
rate	ByVal	int16	The rate at which conversions will be performed. The value is the number of conversions per second and may range from 28 to 11000 samples per second.

Discussion

Calling this subroutine causes a continuous series of analog-to-digital conversions to be performed on the signal appearing at the specified pin. Each 8-bit digital result is automatically sent out the Com1 serial port. Before starting the conversions, the baud rate of Com1 is set to 115,200. The specified pin is automatically set to the proper state for A/D conversion so no additional setup is required prior to use. The conversion stream will continue until ADCToCom1() is called again with the pin parameter set to zero (the rate parameter being meaningless in this case).

The analog input range is approximately 0.25 to 0.75 times Vcc (1.25 volts to 3.75 volts when running on 5 volts) and the resulting digital range is 0 to 255. Analog input levels below the low end of the range and above the high end of the range will produce the low and high digital values, respectively.

Note that the subroutine Com1ToDAC() is designed to receive the data stream generated by this Subroutine. For best accuracy, state changes on other pins of the port containing the analog input should be avoided during the conversion process.

Resource Usage

This subroutine uses the processor's A/D converter, Com1 and the I/O Timer. No other use of these resources should be attempted while the conversion is active. For native code devices, the following ISRs are required.

ISRs Required

Underlying CPU	ISR Name
mega328P, mega644P, mega128	Timer1_CompA
mega1284P	Timer3_CompA
mega1281	Timer4_CompA
mega1280	Timer4_CompA

Compatibility

This subroutine is only available on ATmega-based ZX devices.

See Also   Com1toDAC

Asc

Type	Function returning Byte

Invocation	Asc(str)
	Asc(str, index)

Parameter	Method	Type	Description
str	ByVal	String	The string from which a character will be returned.
index	ByVal	int8/16	The 1-based position in the string from which the character will be returned.

Discussion

This function returns the ASCII character code of the character at the position of the string that is specified. If the second parameter is missing, position 1 is assumed. Note that if the index is less than 1 or larger than the number of characters in the string the return value will be zero.

Example

```
Dim s as String
Dim b as Byte

s = "Howdy"
b = Asc(s)
```

After execution, the variable b will have the value of 72 (48 hex), the character code for H.

Compatibility

BasicX does not support the presence of the second parameter.

See Also	Chr

Asin

Type   Function returning Single

Invocation   Asin(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value from which the arc sine will be computed.

Discussion

The arc sine function is the inverse of the sine function. The return value will be the angle, expressed in radians, whose sine corresponds to the passed value. The type of the return value will be `Single` and the value will range from - /2 to /2. If the argument is greater than 1.0 or less than −1.0, the result will be undefined.

Example

```
Dim val as Single, theta as Single

val = 0.5
theta = Asin(val)         ' result is approximately 0.5236
```

See Also   Sin, DegToRad, RadToDeg

Atn

Type   Function returning Single

Invocation   Atn(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value from which the arc tangent will be computed.

Discussion

The arc tangent function is the inverse of the tangent function. The return value will be the angle, expressed in radians, whose tangent corresponds to the passed value. The return value will be of type `Single` and the value will range from - /2 to /2.

Example

```
Dim val as Single, theta as Single

val = 0.5
theta = Atn(val)  ' result is approximately 0.4636
```

See Also   Atn2**,** DegToRad**,** RadToDeg

Atn2

Type   Function returning Single

Invocation   Atn2(y, x)

Parameter	Method	Type	Description
y	ByVal	Single	y coordinate.
x	ByVal	Single	x coordinate.

Discussion

This function computes the principal value of the arc tangent of y/x, using the signs of both arguments to determine the quadrant of the return value. The return value will be the angle, expressed in radians, from the positive x-axis to the line connecting the origin and the given point. The type of the return value will be `Single` and the value will range from - to . If both `x` and `y` are zero, the value 0.0 will be returned as a special case.

Example

```
Dim x as Single, y as Single, theta as Single

x = 1.0
y = -1.0
theta = Atn2(y, x)        ' result is -0.7854
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   Atn**,** DegToRad**,** RadToDeg

BitCopy

Type	Subroutine

Invocation	BitCopy(destAddr, destBitOfst, srcAddr, srcBitOfst, bitCount)

Parameter	Method	Type	Description
dstAddr	ByVal	integral	The address to which to begin copying.
dstBitOfst	ByVal	integral	The bit offset to which to begin copying.
srcAddr	ByVal	integral	The address from which to begin copying.
srcBitOfst	ByVal	integral	The bit offset from which to begin copying.
bitCount	ByVal	integral	The number of bits to copy.

Discussion

This subroutine can be used to copy an arbitrary number of bits from one location in RAM to another. The copy operation may begin and/or end in the middle of a byte if desired. An overlapping copy (when the destination is in the midst of the data being copied) is handled correctly so that the data to be copied is not overwritten.

For the purposes of this subroutine, RAM considered a sequence of bits with the least significant bits of each byte preceding the more significant bits. This is the same model of RAM that is utilized by `GetBit()` and `PutBit()`. The least significant bit of a byte is at offset zero and the most significant bit is at offset 7.

Note that the bit offsets specified for the second and fourth parameters may have values greater than 7. If a bit offset greater than 7 is given, the corresponding address component is adjusted internally to give the same effect. For example, if an address of 200 and a bit offset of 19 are specified, these are converted internally to 202 and 3, respectively.

All six parameters are converted internally to `UnsignedInteger`.

Caution

This subroutine should be used with care because it is possible to overwrite important data on the stack or other areas of memory which may cause your program to malfunction.

Compatibility

This subroutine is not available on ZX models that are based on the ATmega32 processor (e.g. the ZX-24). Moreover, it is not available in BasicX compatibility mode.

See Also	MemCopy, MemSet

BlockMove

Type   Subroutine

Invocation   BlockMove(count, source, destination)

Parameter	Method	Type	Description
count	ByVal	integral	The number of bytes to copy.
source	ByVal	integral	The address from which to begin copying.
destination	ByVal	integral	The address to which to begin copying.

Discussion

This subroutine is provided for compatibility with BasicX. The more aptly named `MemCopy()` should be used by new applications. An overlapping copy (when the destination is in the midst of the data being copied) is handled correctly so that the data to be copied is not overwritten.

Compatibility

With VM firmware versions prior to v1.1.0 an overlapping copy is not handled correctly nor is it handled correctly in BasicX. A BasicX application that relies on the incorrect handling will, therefore, not work as expected when run on ZX processors.

See Also   BitCopy, MemCopy

BusRead

Type	Subroutine		

Invocation   BusRead(addr, data, count)
BusRead(addr, data, count, delta)

Parameter	Method	Type	Description
addr	ByVal	integral	The bus address at which to begin reading.
data	ByRef	anyType	A buffer to receive the data read.
count	ByVal	integral	The number of bytes to read.
delta	ByVal	integral	The amount by which the address should be changed after each byte is read.

Discussion

For ZBasic devices that support external RAM (e.g. ZX-1281), if the external RAM interface is enabled and a bus has not been defined using DefineBus(), then the external RAM interface is used for the read operation. In this case, the full 16 bits of the specified address are used and the delta parameter is interpreted as a signed 8-bit value that is sign-extended before adding it to the address with each iteration.

For ZBasic devices that do not support external RAM or if the external RAM interface is not enabled, this routine performs a series of read operations on the bus previously defined with the DefineBus() call. This is called the "bit bang" mode. For each read cycle, the low 8-bits of the address is output on the previously specified port and then the ALE pin is strobed (high, then back low). Next, the port is made an input and the RD pin is set low, data is read via the PIN register corresponding to the port, and the RD pin is set back high again. The data value read is stored in the buffer, the specified delta is added to the 8-bit bus address and the cycle is repeated until the specified number of bytes has been read.

It is important to remember that in the bit bang mode only 8 bits of the address are used. Depending on the values of the `addr`, `count` and `delta` parameters, the effective address may wrap around to zero. For example, with `delta=1` specifying a `count` parameter larger than `(256 - LoByte(addr))` will result in the effective address wrapping around to zero.

In either mode, if the optional `delta` parameter is not specified, the value of 1 is assumed. Specifying the delta as zero will result in multiple reads from the same address. A delta of `-1` or `&Hff` will result in the address being decremented after each read.

Example

```
Dim data(1 to 20) as Byte
Call DefineBus(Port.A, C.0, C.1, C.2)
Call BusRead(0, data, SizeOf(data))
```

Compatibility

This subroutine is not available on ZX models that are based on the ATmega32 processor (e.g. the ZX-24) nor is is available on ATxmega-based ZBasic devices. Moreover, it is not available in BasicX compatibility mode.

See Also   BusWrite, DefineBus

BusWrite

Type	Subroutine

Invocation	BusWrite(addr, data, count)
	BusWrite(addr, data, count, delta)

Parameter	Method	Type	Description
addr	ByVal	integral	The bus address at which to begin writing.
data	ByRef	anyType	The data to be written.
count	ByVal	integral	The number of bytes to write.
delta	ByVal	integral	The amount by which the address should be changed after each byte is written.

Discussion

For ZBasic devices that support external RAM (e.g. ZX-1281), if the external RAM interface is enabled and bus has not been defined using DefineBus(), then the external RAM interface is used for the write operation. In this case, the full 16 bits of the specified address are used and the delta parameter is interpreted as a signed 8-bit value that is sign-extended before adding it to the address with each iteration.

For ZBasic devices that do not support external RAM or if the external RAM interface is not enabled, this routine performs a series of write operations on the bus previously defined with the DefineBus() call. This is called the "bit bang" mode. For each write cycle, the low 8-bits of the address is output on the previously specified port and then the ALE pin is strobed (high, then back low). Then, the next data value to be written is output on the port and the WR pin is strobed (low then back high). Finally, the specified delta is added to the bus address and the cycle is repeated until the specified number of bytes has been written.

It is important to remember that in the bit bang mode only 8 bits of the address are used. Depending on the values of the addr, count and delta parameters, the effective address may wrap around to zero. For example, with delta=1 specifying a count parameter larger than (256 - LoByte(addr)) will result in the effective address wrapping around to zero.

In either mode, if the optional delta parameter is not specified, the value of 1 is assumed. Specifying the delta as zero will result in multiple writes to the same address. A delta of -1 or &Hff will result in the address being decremented after each write.

Example

```
Dim data(1 to 20) as Byte
Call DefineBus(Port.A, C.0, C.1, C.2)
Call BusWrite(0, data, SizeOf(data))
```

Compatibility

This subroutine is not available on ZX models that are based on the ATmega32 processor (e.g. the ZX-24) nor is is available on ATxmega-based ZBasic devices. Moreover, it is not available in BasicX compatibility mode.

See Also	BusRead, DefineBus

CallTask

Type	Special Purpose

Invocation	CallTask taskName, taskStack
	CallTask taskName, taskStack, taskStackSize
	CallTask taskName(parameterList), taskStack
	CallTask taskName(parameterList), taskStack, taskStackSize

Parameter	Method	Type	Description
taskName	ByVal	identifier	The name of the task to invoke.
parameterList	varies	varies	Zero or more parameters to be passed to the task, separated by commas.
taskStack	ByRef	array of Byte	The stack for the task (see discussion)
taskStackSize	ByVal	integral	The size of the stack.

Discussion

This construct is used to prepare a task for running; the task doesn't actually execute until its turn comes up in the normal task rotation. In the first and second cases, the `taskName` given must be the name of a user-defined subroutine that takes no parameters. In the third and fourth cases, the `taskName` given must be a user-defined subroutine that takes a number of parameters whose type and number match that of the supplied parameter list. The subroutine may be public or private but if it is private it must exist in the same module as the CallTask invocation that refers to it.

The `taskStack` may be a Byte array, typically defined at the module level, that contains a sufficient amount of space for the task's stack needs. The array can be public or private but if it is private it must exist in the same module as the CallTask invocation that refers to it. Alternately, the stack for a task may be specified by giving its address as an integral expression. In this case, it is usually also advisable to specify the size of the stack since the compiler cannot deduce the size. A task must have exclusive use of the memory dedicated to its task stack. A particular task stack may be used by more than one task but one task must terminate before the next task can re-use the task stack.

If a task is passed parameters when it is invoked, it is advisable that those parameters be passed ByVal because the lifetime of the task may exceed the lifetime of the routine from which the task was invoked. If parameters are passed ByRef (explicitly or implicitly), the compiler will issue a warning. Also, certain types of expressions (notably, those involving user-defined functions that return String types) may not be used as parameter values for task invocation because they require the creation of temporary variable space on the stack during evaluation. The compiler will issue an error message when it detects such situations. This problem can be rectified by manually creating a variable (preferably at the module level) to hold the parameter value.

For native mode devices (e.g. ZX-24n), the task stack size must either be explicitly specified or it must be determinable by the compiler from the size of the task stack array. The compiler will issue an error message if it cannot determine the size of the task stack.

Please read the section on multi-tasking in the ZBasic Reference Manual for more details, including information about how to determine the proper task stack size.

Example 1

```
Dim taskStack(1 to 50) as Byte

Sub Main()
    CallTask MyTask, taskStack
    Do
        Debug.Print "Hello from Main"
        Call Delay(1.0)
```

```
        Loop
End Sub

Sub MyTask()
    Do
        Debug.Print "Hello from MyTask"
        Call Delay(2.0)
    Loop
End Sub
```

Example 2

```
Dim taskStack(1 to 50) as Byte

Sub Main()
    CallTask MyTask(2.0), taskStack
    Do
        Debug.Print "Hello from Main"
        Call Delay(1.0)
    Loop
End Sub

Sub MyTask(ByVal taskDelay as Single)
    Do
        Debug.Print "Hello from MyTask"
        Call Delay(taskDelay)
    Loop
End Sub
```

Example 3

```
Dim taskStack(1 to 50) as Byte

Sub Main()
    Dim stkAddr as UnsignedInteger
    Dim stkSize as Integer

    stkAddr = taskStack.DataAddress
    stkSize = SizeOf(taskStack)
    CallTask MyTask(2.0), stkAddr, stkSize
    Do
        Debug.Print "Hello from Main"
        Call Delay(1.0)
    Loop
End Sub

Sub MyTask(ByVal taskDelay as Single)
    Do
        Debug.Print "Hello from MyTask"
        Call Delay(taskDelay)
    Loop
End Sub
```

Compatibility

In BasicX compatibility mode, the task name must be enclosed in quotes (i.e. so that it appears to be a string). Also, task parameters, specifying the task stack by address, and specifying the task stack size are not supported in BasicX compatibility mode. CallTask cannot be used unless the RTC is enabled in your application.

CBit

Type   Function returning Bit

Invocation   CBit(arg)

Parameter	Method	Type	Description
arg	ByVal	integral, String or Boolean	The value to convert to a Bit value.

Discussion

This function converts a numeric, String or Boolean value to a Bit value as described in the table below.

Input Type	Result
integral, Boolean	The value is the least significant bit of the supplied value.
String	The result is the least significant bit of the numeric value of the characters in the string, ignoring leading space and tab characters. The value string may begin with a plus or minus sign and an optional radix indicator (&H for hexadecimal, &O for octal, &B or &X for binary, all case insensitive). The conversion is terminated upon reaching the end of the string or encountering the first character that is not valid for the indicated radix.

Example

```
Dim pinVal as Bit

pinVal = CBit(GetPin(12))
```

Compatibility

This function is not available in BasicX compatibility mode.

CBool

Type   Function returning Boolean

Invocation   CBool(arg)

Parameter	Method	Type	Description
arg	ByVal	Byte	The value to convert to a Boolean value.

Discussion

This function converts a Byte value to a Boolean value. If the byte has the value 0 the result will be False, otherwise it will be True.

Example

```
Dim pinHi as Boolean

pinHi = CBool(GetPin(12))
```

CByte

Type   Function returning Byte

Invocation   CByte(arg)

Parameter	Method	Type	Description
arg	ByVal	numeric, String, Boolean or Enum	The value to convert to Byte.

Discussion

This function converts any numeric or enumeration value to a Byte value. See the table below for details of the conversion.

Input Type	Result
Boolean	Returns the byte value of the Boolean data item: 0 or 255.
Byte	No effect, the value is as supplied.
Integer	Returns the low byte of the value provided. However, if the supplied value is negative or greater than 255, the returned value will be 255.
UnsignedInteger	Returns the low byte of the value provided. However, if the supplied value is greater than 255, the returned value will be 255.
Enum	Returns the low byte of the value provided. However, if the supplied value is greater than 255, the returned value will be 255.
Long	Returns the low byte of the value provided. However, if the supplied value is negative or greater than 255, the returned value will be 255.
UnsignedLong	Returns the low byte of the value provided. However, if the supplied value is greater than 255, the returned value will be 255.
Single	The supplied value is converted to a Long value (signed 32-bit integer), rounded to the nearest integer. If the fractional part is exactly 0.5, the resulting integer will be even. This is known as "statistical rounding". If the resulting integer value is negative or larger than 255, the result will be 255. Otherwise, the result will be the integral value.
String	The result is the numeric value of the characters in the string, ignoring leading space and tab characters. The value string may begin with a plus or minus sign and an optional radix indicator (&H for hexadecimal, &O for octal, &B or &X for binary, all case insensitive). The conversion is terminated upon reaching the end of the string or encountering the first character that is not valid for the indicated radix.

Compatibility

In BasicX, calling `CByte()` with an `UnsignedInteger` argument returns the low byte of the value. This behavior is inconsistent with the other type conversions. This implementation attempts to make them consistent.

CByteArray

Type Function returning a reference to a Byte array

Invocation CByteArray(addr)

Parameter	Method	Type	Description
addr	ByVal	int16	The address to be converted to a reference to a Byte array.

Discussion

This special function is useful when you have an integral value that you know to be the address of a Byte array and you want to pass it to a subroutine or function that requires a Byte array parameter. The example below shows it being used to determine the number of bytes of data available in the system input queue.

Example

```
Dim cnt as Integer
cnt = GetQueueCount(CByteArray(Register.RxQueue))
```

See Also StatusTask

Ceiling

Type　　　　Function returning Single

Invocation　　Ceiling(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value of which to compute the ceiling.

Discussion

This function returns a Single value that is the smallest integer that is greater than or equal to the supplied value, effectively rounding up to the nearest integer.

Example

```
Dim ceil as Single

ceil = Ceiling(1.5)     ' result is 2.0
ceil = Ceiling(-1.5)    ' result is -1.0
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also　　Floor, Fraction

Chr

Type   Function returning String

Invocation   Chr(arg)

Parameter	Method	Type	Description
arg	ByVal	integral	The character code to place in the string.

Discussion

This function returns a string containing a single character having the value of the supplied parameter. If the parameter is a multi-byte type such as Integer or Long the least significant byte of the value is used and the remaining bytes are ignored.

Tables of ASCII character values may be found in many places on the Internet. A search for "ASCII table" or "ASCII chart" will produce many results.

Example

```
Dim s as String
s = Chr(33)
```

After execution, s will be "!" because 33 is the decimal code for the exclamation mark.

See Also   Asc

CInt

Type   Function returning Integer

Invocation   CInt(arg)

Parameter	Method	Type	Description
arg	ByVal	numeric, Boolean, String or Enum	The value to convert to Integer.

Discussion

This function converts any numeric or enumeration value to an Integer value. See the table below for details of the conversion.

Input Type	Result
`Byte, Boolean`	High byte zero, low byte as supplied.
`Integer`	No effect, the value is as supplied.
`UnsignedInteger`	Value bits are the same as supplied, although interpreted as a signed value.
`Enum`	The resulting value is the Enum member value.
`Long`	The resulting value will be the low word of the supplied value.
`UnsignedLong`	The resulting value will be the low word of the supplied value.
`Single`	The supplied value is converted to signed 32-bit integer, rounded to the nearest integer. If the fractional part is exactly 0.5, the resulting integer will be even. This is known as "statistical rounding". If the resulting integer is larger than will fit in 16-bits, the result is undefined.
`String`	The result is the numeric value of the characters in the string, ignoring leading space and tab characters. The value string may begin with a plus or minus sign and an optional radix indicator (&H for hexadecimal, &O for octal, &B or &X for binary, all case insensitive). The conversion is terminated upon reaching the end of the string or encountering the first character that is not valid for the indicated radix.

Example

```
Dim i as Integer

i = CInt(2.5)      ' result is 2
i = CInt(1.5)      ' result is 2
```

ClearQueue

Type   Subroutine

Invocation   ClearQueue(queue)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue to be cleared.

Discussion

This routine modifies the tracking information contained in the queue data structure to indicate that the queue is empty. If the queue is already empty, this has no effect. If there are characters in the queue, they will be discarded.

Note that before any queue operations are performed, the queue data structure must be initialized. See the discussion of `OpenQueue()` for more details.

Example

```
Dim inQueue(1 to 40) as Byte

Call OpenQueue(inQueue, SizeOf(inQueue))
Call PutQueueStr(inQueue, "Hello")
Call ClearQueue(inQueue)
```

After the call to ClearQueue() the queue will no longer contain the characters that were added.

Compatibility

BasicX allows any type for the first parameter. The ZBasic implementation requires that it be an array of `Byte`.

CLng

Type	Function returning Long

Invocation	CLng(arg)

Parameter	Method	Type	Description
arg	ByVal	numeric, Boolean, String or Enum	The value to convert to Long.

Discussion

This function converts any numeric or enumeration value to a Long value. See the table below for details of the conversion.

Input Type	Result
Byte, Boolean	High 3 bytes zero, low byte as supplied.
Integer	High word will be all ones if the supplied value is negative, zero otherwise. Low word as supplied.
UnsignedInteger	High word zero, low word as supplied.
Enum	The resulting value is the Enum member value.
Long	No effect, the value is as supplied.
UnsignedLong	Value bits are the same as supplied, although interpreted as a signed value.
Single	The supplied value is converted to a signed 32-bit integer, rounded to the nearest integer. If the fractional part is exactly 0.5, the resulting integer will be even. This is known as "statistical rounding". If the magnitude of the supplied value is too large to be represented in 32 bits, the result is undefined.
String	The result is the numeric value of the characters in the string, ignoring leading space and tab characters. The value string may begin with a plus or minus sign and an optional radix indicator (&H for hexadecimal, &O for octal, &B or &X for binary, all case insensitive). The conversion is terminated upon reaching the end of the string or encountering the first character that is not valid for the indicated radix.

Example

```
Dim l as Long

l = CLng(2.5)      ' result is 2
l = CLng(1.5)      ' result is 2
```

CloseCom

Type   Subroutine

Invocation   CloseCom(channel, inQueue, outQueue)

Parameter	Method	Type	Description
channel	ByVal	Byte	The serial channel to close.
inQueue	ByRef	array of Byte	The input queue associated with the channel.
outQueue	ByRef	array of Byte	The output queue associated with the channel.

Discussion

This routine shuts down the specified serial channel. All communication is terminated even if there are still characters in the output queue that have not yet been sent. This call does not clear the queues. If that is a requirement, calls to `ClearQueue()` will need to be made. Alternately, you may want to use the value returned by `StatusCom()` to wait for all queued characters to be transmitted before invoking `CloseCom()`.

When used with ZX devices, invoking this subroutine for Com1 (`channel` = 1) does not actually close the Com1 channel if the application was configured with Com1 implicitly open at startup time. In this case, the effect of the call is to cause Com1 to revert to the default speed (19.2K baud) and to using the default I/O queues.

If the specified serial channel is not open or if an invalid channel number is given the call has no effect. If the channel being closed is the only one of the software-based channels (Com3-Com6) that is open, the Serial Timer will be turned off and the corresponding timer busy flag will be set to False indicating that the Serial Timer is available for other uses.

See Also   ClearQueue, DefineCom, OpenCom, StatusCom

CloseDAC

Type   Subroutine

Invocation   CloseDAC(channel)
                  CloseDAC(channel, status)

Parameter	Method	Type	Description
channel	ByVal	Byte	The DAC channel to close.
status	ByRef	Boolean	A variable to receive the status code.

Discussion

This subroutine terminates the DAC operation on the specified channel. The `status` parameter, if supplied, receives a value to indicate success or failure of the call. If the second channel of the DAC channel pair is also open, it will continue to operate unaffected.

Example

```
Call CloseDAC(1)   ' terminate DAC on channel 1
```

Compatibility

This subroutine is only available for xmega devices and is not available in BasicX compatibility mode.

See Also   DAC, OpenDAC

CloseI2C

Type   Subroutine

Invocation   CloseI2C(channel)

Parameter	Method	Type	Description
channel	ByVal	Byte	The I2C channel number (0-4).

Discussion

This subroutine closes an I2C channel. For the hardware I2C channel, it disables the on-board I2C controller allowing the hardware I2C pins to be used for other purposes. For software I2C channels it has no effect.

Compatibility

This subroutine is not available in BasicX compatibility mode.

See Also   OpenI2C

ClosePWM

Type   Subroutine

Invocation   ClosePWM(channel)
                    ClosePWM(channel, status)

Parameter	Method	Type	Description
channel	ByVal	Byte	The PWM channel to close.
status	ByRef	Boolean	A variable to receive the status code.

Discussion

This subroutine terminates the PWM signal generation on the specified channel and all other PWM channels associated with the same 16-bit timer. The resulting state of the output pins for the affected channels is indeterminate. If your application requires a specific output state, it is recommended that you call `PutPin()` to set the desired state prior to calling `ClosePWM()`.

A side effect of a successful `ClosePWM()` call is that the timer busy flag for the associated timer (e.g. `Register.Timer1Busy`) will be set to `False` indicating that the timer may be used for other purposes.

Example

```
Call ClosePWM(1)   ' terminate PWM on channel 1 and 2
```

Compatibility

This subroutine is not available in Basic X compatibility mode.

See Also   OpenPWM, PWM

ClosePWM8

Type	Subroutine	

Invocation   ClosePWM8(channel)
                       ClosePWM8(channel, status)

Parameter	Method	Type	Description
channel	ByVal	Byte	The 8-bit PWM channel to close.
status	ByRef	Boolean	A variable to receive the status code.

Discussion

This subroutine terminates the PWM signal generation on the specified 8-bit channel and all other PWM channels associated with the same 8-bit timer. The resulting state of the output pins for the affected channels is indeterminate. If your application requires a specific output state, it is recommended that you call PutPin() to set the desired state prior to calling ClosePWM8().

The status parameter, if supplied, receives a value to indicate success or failure of the call.

A side effect of a successful ClosePWM8() call is that the timer busy flag for the associated timer (e.g. Register.Timer2Busy) will be set to False indicating that the timer may be used for other purposes.

Example

```
Call ClosePWM8(1)    ' terminate PWM on channel 1 (and channel 2)
```

Compatibility

This subroutine is not available in BasicX compatibility mode nor is it available on ATxmega-based ZBasic devices.

See Also   OpenPWM8, PWM8

CloseSPI

Type　　　　Subroutine

Invocation　　CloseSPI(channel)

Parameter	Method	Type	Description
channel	ByVal	Byte	The SPI channel number (1-4).

Discussion

This subroutine closes an SPI channel. The primary purpose for this subroutine is to cancel SPI Slave mode. It has no effect for channels that are not open or channels that are open in Master mode.

Compatibility

This subroutine is not available in BasicX compatibility mode.

See Also　　OpenSPI, SPICmd, SPIGetByte, SPIPutByte, SPIGetData, SPIPutData, SPIStart, SPIStop

CloseWatchDog

Type   Subroutine

Invocation   CloseWatchDog()

Discussion

This subroutine disables the watchdog timer.

Compatibility

This subroutine is not available in BasicX compatibility mode.

See Also   OpenWatchDog, WatchDog

CloseX10

Type   Subroutine

Invocation   CloseX10(channel, inQueue, outQueue)

Parameter	Method	Type	Description
channel	ByVal	Byte	The X-10 channel to close.
inQueue	ByRef	array of Byte	The input queue associated with the channel.
outQueue	ByRef	array of Byte	The output queue associated with the channel.

Discussion

This routine shuts down the specified X-10 communication channel. All communication is terminated even if there are still data in the output queue that have not yet been sent. This call does not clear the queues. If that is a requirement, calls to `ClearQueue()` will need to be made.

If the specified X-10 channel is not open or if an invalid channel number is given the call has no effect. The `inQueue` and `outQueue` parameters are currently not used but are present for congruency with `CloseCom()`. Zero values may be used for either or both parameters.

Resource Usage

The X-10 communication requires the use of a zero-crossing signal input to the ZX. See the description of OpenX10() for more information.

Compatibility

This subroutine is not available on ZX models that are based on the ATmega32 processor (e.g. the ZX-24). Moreover, it is not available in BasicX compatibility mode.

See Also   DefineX10**,** OpenX10**,** StatusX10

CNibble

Type   Function returning Nibble

Invocation   CNibble(arg)

Parameter	Method	Type	Description
arg	ByVal	integral, String or Boolean	The value to convert to a Nibble value.

Discussion

This function converts a numeric, String or Boolean value to a Nibble value as described in the table below.

Input Type	Result
integral, Boolean	The value is the four least significant bits of the supplied value.
String	The result is the four least significant bits of the numeric value of the characters in the string, ignoring leading space and tab characters. The value string may begin with a plus or minus sign and an optional radix indicator (&H for hexadecimal, &O for octal, &B or &X for binary, all case insensitive). The conversion is terminated upon reaching the end of the string or encountering the first character that is not valid for the indicated radix.

Example

```
Dim nVal as Nibble

nVal = CNibble(Register.PortC)
```

Compatibility

This function is not available in BasicX compatibility mode.

Com1toDAC

Type	Subroutine
Invocation	Com1toDAC(pin)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin number on which the analog voltage will be re-created.

Discussion

Calling this subroutine prepares Com1 to receive a continuous stream of 8-bit values from an external source. The baud rate is automatically set 115,200. When each value is received, the value is output as an analog voltage on the specified pin. The resulting analog voltage will range from near 0 volts corresponding to the received value of 0 to near the processor voltage (usually +5 volts) corresponding to the received value of 255. The method used to create the analog voltage is similar to that used for PutDAC() and the signal will require some filtering. See the description of PutDAC() for more details. The output pin is updated at a fixed rate of 11,000 times per second.

This routine returns immediately after setting up the conversion process. The conversion process will be terminated if Com1toDAC() is called again with a parameter of zero. Also, if data is not received for approximately 200 cycles, the conversion process will be automatically terminated.

Note that the subroutine ADCtoCom1() is designed to produce the data stream to be received by this subroutine.

Resource Usage

This subroutine uses Com1 and the I/O Timer. No other use of these resources should be attempted while the reception is active. For native code devices, the following ISRs are automatically loaded.

ISRs Required

Underlying CPU	ISR Name
mega328P, mega644P, mega128	Timer1_CompA
mega1284P	Timer3_CompA
mega1281	Timer4_CompA
mega1280	Timer4_CompA

Compatibility

This subroutine is only available on ATmega-based ZX devices.

See Also   ADCtoCom1

ComChannels

Type   Subroutine

Invocation   ComChannels(chanCount, maxSpeed)

Parameter	Method	Type	Description
chanCount	ByVal	Byte	The total desired number of software-based serial channels.
maxSpeed	ByVal	int8/16	The desired maximum baud rate to be supported.

Discussion

In addition to the serial channels implemented in hardware on the processor (e.g. Com1), ZBasic can support up to four additional serial communication channels that are implemented in the system software. The software-based serial channels are numbered Com3 through Com6. However, by default, only one additional channel, Com3, is supported. If you want to use serial channels 4 through 6 you must call this subroutine first to specify the maximum number (generally, up to 4) that you want to have available. This subroutine must be called only when there are no open software-based serial channels (COM3 through COM6). If it is called when one or more channels are already open, it will have no effect. For native mode devices, the upper limit of the `chanCount` parameter may be lower than 4 if the `Option ComChannels` directive is used. If the value of `chanCount` exceeds the upper limit the call will fail silently.

After ComChannels() has been invoked, the serial channels that will be available depends on the value specified by the `chanCount` parameter. If the value 2 is specified, for example, channels Com3 and Com4 will be available. Once the number of software-based serial channels has been established you may then use DefineCom(), OpenCom(), and CloseCom() to manage the available channels by specifying the appropriate channel number in those calls.

In addition to specifying the total number of software-based serial channels that you want, you must also specify the maximum baud rate that you wish to utilize. The supported rates are 300, 600, 1200, 2400, 4800, 9600 and 19,200 baud but see below for additional discussion about the maximum baud.

Because the COM3 to COM6 serial channels are implemented in software, when one or more of the channels is open there will be a certain amount of processing overhead that will reduce the speed at which program instructions will be executed. Moreover, the processing overhead is higher when supporting higher baud rates as compared to lower baud rates and the overhead is higher when supporting a larger number of channels. It is prudent, therefore, to choose the lowest baud rate and lowest number of channels that is practical for your application.

Also note that when supporting two or more channels, there is a small possibility that incoming characters might not be properly recognized at the highest rate. The probability of not being able to properly synchronize on the incoming character's start bit increases with each additional channel that is supported. For this reason, it is recommended that the maximum baud rate be limited to 9600 when configured for 2 or more channels.

For devices operating at speeds between 7.37MHz and 14.7456MHz, the number of software-implemented serial channels should be limited to two and the maximum speed should be limited to 9600 baud. At slower speeds, further reductions in the channel use and maximum speed may be necessary.

Resource Usage

The software-implemented serial channels utilize the Serial Timer for the bit rate timing. No other use of the Serial Timer should be attempted when serial channels 3-6 are open. The "Busy" flag for the timer used to implement the software serial channels will be set to True when one or more of the software-implemented serial channels is open.

Example

```
Dim iq4(1 to 20) as Byte
Dim oq4(1 to 20) as Byte

Call ComChannels(4, 4800)
Call DefineCom(6, 12, 13, &H80)
Call OpenCom(6, 4800, iq4, oq4)
```

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   DefineCom, CloseCom, OpenCom, StatusCom, SerialGetByte, SerialOut

Console.Read

Type   Function returning Byte

Invocation   Console.Read()

Discussion

This function can be invoked to retrieve a character from the input queue associated with Com1 (by default, but see Option Console in the ZBasic Language Reference Manual). If the value of `Register.Console.Echo` is `True`, the character will automatically be sent back out via the output queue associated with the designated serial channel. When this function is called it will not return until a character is available. However, other tasks will continue to execute. You may wish to query the designated queue to find out if there are characters available before calling this function. See the example below.

Example

```
Dim b as Byte
b = Console.Read()  ' this will wait until a character is available

If (GetQueueCount(CByteArray(Register.RxQueue)) > 0) Then
  b = Console.Read()  ' read the next available character
End If
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   Console.ReadLine, Console.Write, Console.WriteLine

Console.ReadLine

Type   Function returning String

Invocation   Console.ReadLine()

Discussion

This function can be invoked to retrieve a sequence of characters from from the input queue associated with Com1 (by default, but see Option Console in the ZBasic Language Reference Manual) terminated by an end-of-line character. If the value of `Register.Console.Echo` is `True`, each character received will automatically be sent back out via the output queue associated with the designated serial channel. When this function is called it will not return until an end-of-line character is received. However, other tasks will continue to execute. The end-of-line character is line feed (`&H0a`) by default but you may change it using `Register.Console.EOL`.

While the characters of the line are being read, if a backspace character is received (`&H08`) the most recently received character will be deleted from the internal buffer. Additional backspace characters will each remove another character from the buffer until it is empty. If a carriage return is received (`&H0d`) it will be ignored unless `Register.Console.EOL` is a carriage return.

The end-of-line character is not included in the returned string and the maximum length of the string is 255 characters. Additional characters received after the 255th character will be discarded while awaiting the end-of-line character.

Example

```
Dim s as String
s = Console.ReadLine()
```


Compatibility

This function is not available in BasicX compatibility mode.

See Also   Console.Read, Console.Write, Console.WriteLine

Console.Write

Type	Special Purpose

Invocation Console.Write(arg)

Parameter	Method	Type	Description
arg	ByVal	String	A string to send out the console port

Discussion

Console.Write is neither a subroutine nor a function. It has more in common with ZBasic statements but it is described here for ease of reference. This special purpose method is useful for outputting debugging information and other data to Com1 (by default, but see Option Console in the ZBasic Language Reference Manual). Note that no carriage return/new line is output after the string.

When this method is invoked, execution of the current task will not continue and no other task will be allowed to run until the string's characters have been transferred to the system output queue.

Example

```
Console.Write("Hello, world! ")

Console.Write("The value is " & CStr(val))
```

This example uses the concatenation operator to produce a single string that is passed to the method.

See Also Debug.Print, Console.Read, Console.ReadLine, Console.WriteLine

Console.WriteLine

Type   Special Purpose

Invocation   Console.WriteLine(arg)

Parameter	Method	Type	Description
arg	ByVal	String	A string to send out the console port

Discussion

Console.WriteLine is neither a subroutine nor a function. It has more in common with ZBasic statements but it described here for ease of reference. This special purpose method is useful for outputting debugging information and other data to Com1 (by default, but see Option Console in the ZBasic Language Reference Manual). Note that a carriage return/new line is always output following the string.

When this method is invoked, execution of the current task will not continue and no other task will be allowed to run until the string's characters have been transferred to the system output queue. This caveat applies separately to the string specified by the parameter and to the end-of-line sequence that is also output.

Examples

```
Console.WriteLine("Hello, world! ")

Console.WriteLine("The value is " & CStr(val))
```

The second example uses the concatenation operator to produce a single string that is passed to the method.

See Also   Debug.Print, Console.Read, Console.ReadLine, Console.Write

ControlCom

Type	Subroutine	
Invocation	ControlCom(chan, rxFlowPin, txFlowPin)	
	ControlCom(chan, rxFlowPin, txFlowPin, flags)	

Parameter	Method	Type	Description
chan	ByVal	Byte	The serial channel of interest.
rxFlowPin	ByVal	Byte	The pin to use for receive flow control.
txFlowPin	ByVal	Byte	The pin to use for transmit flow control.
flags	ByVal	Byte	Flag bits controlling the sense of the flow control lines.

Discussion

This subroutine sets a flow control pin for the receive side and/or transmit side of a serial channel. Either or both of the second and third parameters may be zero indicating that that type of flow control is not desired. If the fourth parameter is not specified, it defaults to the value zero indicating that the flow control pins should be active high. If the fourth parameter is specified, the bits of its value have the meaning given in the table below.

Flag Parameter Values

Value	Meaning
&H01	The receive flow control pin should be active low.
&H02	The transmit flow control pin should be active low.

The remaining bits are currently undefined but may be used in the future. For compatibility with new functionality that may be added in the future, the unused bits should always be zero.

If a receive flow control pin is specified, the pin will be made an output and placed in the active state. This indicates to the sender that the ZX is ready to accept serial data. When the channel's receive queue is nearly full (two bytes of space left), the receive flow control pin will be set to the inactive state indicating to the sender that data transmission should be temporarily suspended. When additional space becomes available in the receive queue (at least three bytes), the receive flow control pin will be set back to the active state.

If a transmit flow control pin is specified, the pin will be made an input. Before sending data, the ZX will check the state of the transmit flow control pin and, if it is at the inactive level, no data will be sent. Note that the input is checked periodically and transmission will resume if the transmit flow control pin is in the active state when sampled.

The current state of the flow control signals is part of the value returned by StatusCom().

It is important to note that a receive queue that is too small is likely to result in a deadlock since there will never be enough free space to activate the flow control signal. Also, when a channel is closed the flow control settings for the channel are cleared. For that reason, it is recommended that the call to ControlCom() be made, if desired, some time after a channel is opened and before it is closed.

Compatibility

This subroutine is not available on ZX devices based on the mega32 (e.g. ZX-24). Moreover it is not available in BasicX compatibility mode.

See Also	CloseCom, ComChannels, DefineCom, OpenCom, StatusCom

Cos

Type   Function returning Single

Invocation Cos(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The angle, in radians, of which the cosine will be computed.

Discussion

The return value will be the cosine of the supplied value, ranging from –1.0 to 1.0.

Example

```
Const pi as Single = 3.14159
Dim val as Single

val = Cos(pi)      ' result is –1.0
```

See Also   Acos**,** DegToRad**,** RadToDeg

CountTransitions

Type   Function returning Long

Invocation   CountTransitions(pin, interval)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin on which logic transitions will be counted.
interval	ByVal	Single or Long	The time interval specified in seconds or I/O Timer ticks, respectively, during which transitions will be counted. See the discussion below for information on range and resolution.

Discussion

When called, this routine will begin counting logic transitions on the specified pin and will continue until the specified interval has elapsed. During the counting process processor interrupts are disabled. This strategy allows high precision in measuring the interval but has the drawback that other processes that utilize interrupts will not function correctly. Among such affected processes are all serial communication and multi-tasking. For this reason, the counting interval should be kept as short as possible. RTC ticks that occur during the counting process are accumulated and the RTC is updated when the counting is finished.

The specified pin, which you must configure to be an input before calling, is sampled at a fixed rate of approximately 1/36 (ATtiny, ATmega) or 1/40 (ATxmega) of the CPU frequency. The sample rate, default resolution and maximum measurement interval are shown in the table below for various CPU frequencies. You may modify the range and resolution of the measurement interval by modifying the built-in variable `Register.TimerSpeed1`. See the special section I/O Timer Prescaler Values for more details.

Important Values for ZX Devices

Processor Family	Frequency	Sample Rate	Default Resolution	Maximum Interval
ATmega	7.3738 MHz	204.8 KHz	4.883 µS	10,485 sec.
ATmega	14.7456 MHz	409.6 KHz	2.441 µS	5,242 sec.
ATmega	18.432 MHz	512.0 KHz	1.953 µS	4,194 sec.
ATxmega	29.4912 MHz	737.3 KHz	1.356 µS	2,912 sec.

For generic target devices, which can operate at an arbitrary main frequency and RTC frequency, the important values related to CountTransitions are shown in the table below where F_CPU is the main processor frequency.

Important Values for Generic Target Devices

Processor Family	Sample Rate F_SAMP	Default Resolution	Maximum Interval
ATtiny, ATmega	F_CPU/36	1/F_SAMP	2147483647/F_SAMP
ATxmega	F_CPU/40	1/F_SAMP	2147483647/F_SAMP

Resource Usage

This function uses the I/O Timer and disables interrupts during the counting process. However, RTC ticks are accumulated during the process and the RTC is updated upon completion. The maximum number of missed RTC ticks that can be tracked is 65,535. A measurement interval longer than that number of RTC fast ticks will result in incorrect RTC accumulator values. The maximum measurement interval for correct adjustment the RTC is shown in the tables above.

Compatibility

In BasicX missed RTC ticks are not accounted for.

CPUSleep

Type	Subroutine		

Invocation	CPUSleep()		
	CPUSleep(mode)		

Parameter	Method	Type	Description
mode	ByVal	Byte	The sleep mode to use.

Discussion

This routine puts the processor into a special sleep mode in which activity and power consumption are reduced. The characteristics of the sleep mode are controlled by certain bits in one or more CPU registers (see the tables below). For more information about the modes, consult the Atmel documentation for the ATtiny, ATmega or ATxmega processor being used. If the optional `mode` parameter (not supported for VM devices) is not given, the existing sleep mode bit values will be used.

Registers Containing the Sleep Mode Bits for ZX Devices

ZX Device	Register
ZX-24, ZX-40, ZX-44, ZX-24e, ZX-128e, ZX-128ne	`Register.MCUCR`
ZX-24a, ZX-24p, ZX-24n, ZX-24r, ZX-24s, ZX-24t, ZX-24ae, ZX-24ne, ZX-24pe, ZX-24ru, ZX-24su	`Register.SMCR`
ZX-40a, ZX-40p, ZX-40n, ZX-40r, ZX-40s, ZX-40t	`Register.SMCR`
ZX-44a, ZX-44p, ZX-44n, ZX-44r, ZX-44s, ZX-44t	`Register.SMCR`
ZX-328n, ZX-328l, ZX-32n, ZX-32l, ZX-328nu	`Register.SMCR`
ZX-1281, ZX-1281n, ZX-1280, ZX-1280n, ZX-1281e, ZX-1281ne	`Register.SMCR`
ZX-24x, ZX-24u, ZX-32a4, ZX-128a1, ZX-24xu	`Register.SLEEP_CTRL`

Registers Containing the Sleep Mode Bits for Generic Target Devices

Target Device	Register
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A, tiny2313, tiny2313A, tiny4313, tiny1634, tiny441, tiny841	`Register.MCUCR`
tiny48, tiny88, tiny87, tiny167, tiny828	`Register.SMCR`
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	`Register.SMCR`
mega8515, mega161, mega162	`Register.MCUCR` `Register.MCUCSR` `Register.EMCUCR`
mega8, mega8A, mega16, mega16A, mega32, mega32A, mega64, mega64A, mega128, mega128A, mega163, mega323, mega8535	`Register.MCUCR`
mega164A, mega164P, mega164PA, mega324P, mega324PA, mega644, mega644A, mega644P, mega644PA, mega1284P, mega1281, mega2561, mega640, mega1280, mega2560	`Register.SMCR`
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P, mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P, mega16U4, mega32U4, mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162, AT90CAN32, AT90CAN64, AT90CAN128, AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	`Register.SMCR`
all ATxmega	`Register.SLEEP_CTRL`

CRC16

Type   Function returning UnsignedInteger

Invocation   CRC16(data, count, crcPoly, crcInit, crcFlags)

Parameter	Method	Type	Description
data	ByRef	anyType	The data bytes to add to the CRC value.
count	ByVal	integral	The number of bytes to process.
crcPoly	ByVal	UnsignedInteger	The CRC polynomial to use.
crcInit	ByVal	UnsignedInteger	The initial value of the CRC.
crcFlags	ByVal	integral	Flag bits that control the CRC computation.

Discussion

This function computes the CRC value over a number of data bytes using a specified polynomial and initial value. The values to use for the polynomial and the initial value depend on the style of CRC that you need to generate. See the discussion below for further details. The `flags` parameter contains bits that control aspects of the CRC computation as described in the table below.

Flag Values for the CRC Compuation

Constant	Hex	Binary	Description
zxCRCRefIn	&H01	xxxx xxx1	Each input data bytes will be "reflected".
zxCRCRefOut	&H02	xxxx xx1x	The final CRC value will be "reflected".

The remaining bits are reserved for future use and should always be zero.

In this context, the term "reflection" refers to reversing the order of the bits in a data item so that the most significant becomes the least significant and vice versa. For a multi-byte data item, the bits in each byte are reversed and the order of the bytes is reversed as well.

Although this function will typically be used to compute the CRC value for an entire block of data at once, it may also be used in a byte-by-byte or data burst mode. To do so, you would pass the computed CRC value from the previous iteration as the initial value. Note, however, that you shouldn't use the `zxRefOut` flag bit in this case. Rather, if you need reflected output you would perform the bit reversal on the final CRC value when you reach the end of the data stream. You can reverse the bit order of a 16-bit value by using the following code fragment.

```
crc = MakeWord(FlipBits(HiByte(crc)), FlipBits(LoByte(crc)))
```

CRC algorithms can be described by a parametric model known as the RockSoft model (see http://www.repairfaq.org/filipg/LINK/F_crc_v34.html#CRCV_005). This CRC implementation supports the POLY, INIT, REFIN and REFOUT parameters of the model with WIDTH=16 and XOROUT=0. If necessary, you can easily implement a non-zero XOROUT parameter by using the following code fragment.

```
crc = crc Xor XorOutValue
```

The Rocksoft model parameters for commonly used CRC computations are given in the table below.

Rocksoft Model Parameters for Common CRC Algorithms

Parameter/Type	CRC-16	CRC-CCITT	ModBus	CRC-32
WIDTH	16	16	16	32
POLY	&H8005	&H1021	&H8005	&H04c11db7
INIT	&H0000	&Hffff	&Hffff	&Hffffffff
REFIN	True	False	True	True
REFOUT	True	False	True	True
XOROUT	&H0000	&H0000	&H0000	&Hffffffff
CHECK	&Hbb3d	&H29b1	&H4b37	&Hcbf43926

The parameters are included in the table above for the CRC-32 algorithm but, of course, they must be used with the `CRC32()` function. The CHECK value is the CRC result for the string of characters `"123456789"`.

Additional information on CRC calculations may be found in many places on the Internet. One useful site that implements a CRC calculator is http://www.zorc.breitbandkatze.de/crc.html. If you don't know the parameters required for a particular CRC, you may be able to deduce the correct parameters by using the calculator if you have a sample message and its CRC value. One of the variables available in the CRC calculator on the web page mentioned is "direct" vs. "nondirect". This implementation uses the "direct" method.

Example

```
Dim data(1 to 20) as Byte
Dim crc as UnsignedInteger
' compute the CRC using the CRC-16 algorithm
crc = CRC16(data, 10, &H8005, &H0000, zxCRCRefIn Or zxCRCRefOut)
```

Compatibility

This function is not available in BasicX compatibility mode. Also, on ZX models that are based on the ATmega32 processor (e.g. the ZX-24) this function is implemented in "user code" (as opposed to being part of the VM) and is consequently slower than on other ZX models and ZBasic devices generally.

See Also   CRC32

CRC32

Type   Function returning UnsignedLong

Invocation   CRC32(data, count, crcPoly, crcInit, crcFlags)

Parameter	Method	Type	Description
data	ByRef	anyType	The data bytes to add to the CRC value.
count	ByVal	integral	The number of bytes to process.
crcPoly	ByVal	UnsignedLong	The CRC polynomial to use.
crcInit	ByVal	UnsignedLong	The initial value of the CRC.
crcFlags	ByVal	integral	Flag bits that control the CRC computation.

Discussion

This function computes the CRC value over a number of data bytes using a specified polynomial and initial value. The values to use for the polynomial and the initial value depend on the style of CRC that you need to generate. The `flags` parameter contains bits that control aspects of the CRC computation as described in the table below.

<div align="center">Flag Values for the CRC Compuation</div>

Constant	Hex	Binary	Description
zxCRCRefIn	&H01	xxxx xxx1	The input data bytes will be "reflected".
zxCRCRefOut	&H02	xxxx xx1x	The final CRC will be "reflected".

The remaining bits are reserved for future use and should always be zero.

Although this function will typically be used to compute the CRC value for an entire block of data at once, it may also be used in a byte-by-byte or data burst mode. To do so, you would pass the computed CRC value from the previous iteration as the initial value. Note, however, that you shouldn't use the `zxRefOut` flag bit in this case. Rather, if you need reflected output you would perform the bit reversal on the final CRC value when you reach the end of the data stream.

See the discussion of the `CRC16()` function for additional information.

Example

```
Dim data(1 to 20) as Byte
Dim crc as UnsignedLong
crc = Not CRC32(data, 10, &H04c11db7, &Hffffffff, zxCRCRefIn Or zxCRCRefOut)
```

Compatibility

This function is not available in BasicX compatibility mode. Also, on ZX models that are based on the ATmega32 processor (e.g. the ZX-24) this function is implemented in "user code" (as opposed to being part of the VM) and is consequently slower than on other ZX models and ZBasic devices generally.

See Also   CRC16

CSng

Type   Function returning Single

Invocation   CSng(arg)

Parameter	Method	Type	Description
arg	ByVal	numeric or Enum	The value to convert to Single.

Discussion

This function converts any numeric or enumeration value to a `Single` value. For integral and `Enum` types, the result will be the floating point approximation of the integral value. If a `Single` type parameter is supplied, the result is identical to the parameter value. If a `String` type parameter is supplied, the result will be the numeric value of the character string. The form of the character representation supported is identical to that supported by `ValueS()`.

Example

```
Dim b as Byte
Dim f as Single

b = 21
f = CSng(b)
```

Compatibility

In BasicX, passing an `UnsignedLong` value larger than 2,147,483,647 erroneously generates a negative Single result. This implementation handles `UnsignedLong` values correctly.

CStr

Type   Function returning String

Invocation   CStr(arg)

Parameter	Method	Type	Description
arg	ByVal	any type	The value to convert to String.

Discussion

This function converts any Boolean, numeric or enumeration value to a String value. See the table below for details of the conversion.

Input Type	Result
Boolean	The string `"True"` or `"False"`.
Byte, Bit, Nibble	A string containing decimal digits representing the value.
Integer	A string containing decimal digits representing the value. If the value is negative, the string will begin with a minus sign.
UnsignedInteger	A string containing decimal digits representing the value.
Enum	A string containing decimal digits representing the Enum member value.
Long	A string containing decimal digits representing the value. If the value is negative, the string will begin with a minus sign.
UnsignedLong	A string containing decimal digits representing the value.
Single	A string representing the value. Depending on the value, the form may be standard decimal form with a decimal point separating the whole and fractional parts or it may be in "scientific notation" form. In some cases, there will be no decimal point at all, e.g. with values having no fractional part.

When converting `Single` values, some special cases are detected resulting in the strings shown in the table below. See the function `SngClass()` for more information about the special cases.

Special Value	Result
NaN	`"*.*"`.
±Infinity	`"&.&"`
Denormalized value	`"#.#"`

Compatibility

In BasicX the special `Single` values are not handled properly.

See Also   CStrHex, Fmt

CStrHex

Type   Function returning String

Invocation   CStrHex(arg)

Parameter	Method	Type	Description
arg	ByVal	numeric	The value to convert to a hexadecimal String.

Discussion

This function converts any Boolean, numeric or enumeration value to a String value. The content of the string will be hexadecimal characters that represent the value of the bytes comprising the passed value. The number of characters in the string varies depending on the type of the value passed. See the table below.

Input Type	Number of Characters
Boolean, Bit, Nibble, Byte	2
Integer, UnsignedInteger, Enum	4
Long, UnsignedLong, Single	8

Compatibility

This function is not available in BasicX compatibility mode.

CType

Type	Function returning a converted type (see discussion below)
Invocation	CType(value1, enumType) or CType(value2, typeCast)

Parameter	Method	Type	Description
value1	ByVal	numeric or Enum	The value to convert to another type.
enumType	ByVal	Enum	The name of an Enum type.
value2	ByVal	any type	The value to convert to another type.
typeCast	ByVal	string	A C/C++ typecast.

Discussion

In the first form shown above, this function converts any numeric value or enumeration member to be a member of specified enumeration type. No checking is done to confirm that the given value actually corresponds to one of the members of the enumeration. See the section on enumerations in the ZBasic Reference Manual for more information.

Example

```
Enum Color
      Red
      Green
      Blue
End Enum

Dim c as Color

c = CType(1, Color)      ' c will have the value Green
```

In the second form, useful only for native mode devices, the specified value is emitted along with the specified typecast string in a form intended to coerce the value to a desired type. The typecast string, which may be any valid C/C++ cast, may have one of two forms. If the typecast string contains a dollar sign, the value given, which may be an arbitrarily complex expression, is substituted in place of the dollar sign and the result is emitted. If no dollar sign is present in the typecast string, the typecast string is emitted verbatim followed immediately by the value (enclosed in parentheses if the value comprises a complex expression).

Examples

```
Call foo(CType(val + 10, "(char *)"))
addr = CType(bar(3), "reinterpret_cast<uint16_t>($)")
```

In the first example, assuming that foo() is an external C/C++ function that requires a char * parameter, the generated code would look something like this:

```
foo((char *)(zv_val + 10));
```

In the second example, assuming that bar() is an external C/C++ function that returns a pointer of some type, the generated code would look something like this:

```
zv_addr = reinterpret_cast<uint16_t>(bar(3));
```

CUInt

Type   Function returning UnsignedInteger

Invocation CUInt(arg)

Parameter	Method	Type	Description
arg	ByVal	numeric, Boolean, String or Enum	The value to convert to UnsignedInteger.

Discussion

This function converts any numeric or enumeration value to an UnsignedInteger value. See the table below for details of the conversion.

Input Type	Result
`Byte, Boolean`	High byte zero, low byte as supplied.
`Integer`	Value bits are the same as supplied, although interpreted as an unsigned value.
`UnsignedInteger`	No effect, the value is as supplied.
`Enum`	Resulting value is the Enum member value.
`Long`	Resulting value is the low word of the supplied value.
`UnsignedLong`	Resulting value is the low word of the supplied value.
`Single`	The supplied value is converted to a signed 32-bit integer, rounded to the nearest integer. If the fractional part is exactly 0.5, the resulting integer will be even. This is known as "statistical rounding". If the resulting signed integer is negative or larger than 65535, the result is undefined. Otherwise, the result is the value of the integer.
`String`	The result is the numeric value of the characters in the string, ignoring leading space and tab characters. The value string may begin with a plus or minus sign and an optional radix indicator (&H for hexadecimal, &O for octal, &B or &X for binary, all case insensitive). The conversion is terminated upon reaching the end of the string or encountering the first character that is not valid for the indicated radix.

Example

```
Dim u as UnsignedInteger

u = CUInt(2.5)     ' result is 2
u = CUInt(1.5)     ' result is 2
```

Compatibility

The ability to convert from `Single` is not supported in BasicX compatibility mode.

CULng

Type   Function returning UnsignedLong

Invocation   CULng(arg)

Parameter	Method	Type	Description
arg	ByVal	numeric, Boolean, String or Enum	The value to convert to UnsignedLong.

Discussion

This function converts any numeric or enumeration value to an UnsignedLong value. See the table below for details of the conversion.

Input Type	Result
Byte, Boolean	High 3 bytes zero, low byte as supplied.
Integer	High word will be zero, low word as supplied.
UnsignedInteger	High word will be zero, low word as supplied.
Enum	High word zero, low word contains Enum member value.
Long	Value bits are the same as supplied, although interpreted as an unsigned value.
UnsignedLong	No effect, the value is as supplied.
Single	Supplied value converted to signed 32-bit integer, rounded to the nearest integer. If the fractional part is exactly 0.5, the resulting integer will be even. This is known as "statistical rounding". If the supplied value is negative or if it is too large to be represented in 32 bits, the result is undefined.
String	The result is the numeric value of the characters in the string, ignoring leading space and tab characters. The value string may begin with a plus or minus sign and an optional radix indicator (&H for hexadecimal, &O for octal, &B or &X for binary, all case insensitive). The conversion is terminated upon reaching the end of the string or encountering the first character that is not valid for the indicated radix.

Example

```
Dim ul as UnsignedLong

ul = CULng(2.5)    ' result is 2
ul = CULng(1.5)    ' result is 2
```

DAC

Type	Subroutine		

Invocation   DAC(channel, dacValue)
            DAC(channel, dacValue, stat)

Parameter	Method	Type	Description
channel	ByVal	Byte	The DAC channel to use.
dacValue	ByVal	integral	The desired DAC value (see discussion below).
stat	ByRef	Boolean	The variable to receive the status code.

Discussion

This routine creates an analog signal on the pin corresponding to the specified channel (see OpenDAC() for more information). Only the least significant 12 bits of the specified value are used and the resulting analog level will be approximately equal to `dacValue` divided by 4095 times the DAC reference voltage specified with OpenDAC().

Compatibility

This subroutine is only available for xmega devices and is not available in BasicX compatibility mode.

See Also   CloseDAC, OpenDAC

DACPin

Type   Subroutine

Invocation   DACPin(pin, dacValue, dacAccumulator)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin to which the DAC signal will be output.
dacValue	ByVal	Byte	The value representing the desired analog output. See the discussion below.
dacAccumulator	ByRef	Byte	A value used in the DAC process. See discussion below.

Discussion

This routine creates a digital approximation of an analog signal on the specified pin using a pseudo-PWM technique. ZBasic supports this routine for backward compatibility. New applications should use `PutDAC()` as it is more flexible. See the description of `PutDAC()` for more information.

For ZBasic devices based on the ATxmega, a hardware DAC is available. In most applications requiring a DAC, using the hardware DAC will produce much better results.

Resource Usage

This routine disables interrupts for approximately 200µS during the generation process.

See Also   DAC, OpenDAC, PutDAC

Debug.Print

Type	Special Purpose
Invocation	Debug.Print stringList

Parameter	Method	Type	Description
stringList	ByVal	String	One or more strings or values to send out the console port.

Discussion

Debug.Print is neither a subroutine nor a function. It has more in common with ZBasic statements but it described here for ease of reference. This special purpose method is useful for outputting debugging information and other data to Com1 (by default, but see Option Console in the ZBasic Language Reference Manual). The arguments provided to the command consist of zero or more strings or values each separated by a semicolon. Unless the list ends with a semicolon, a carriage return/new line will also be output after all of the strings have been output.

If a non-string scalar value is supplied, it is output as if it were converted to a string using the CStr() function. If a RAM-resident Byte array name is given, its content is output, byte by byte, up to but not including the first byte having the value zero (i.e it is treated as a null-terminated sequence of characters).

When this statement is invoked, execution of the current task will not continue and no other task will be allowed to run until the string's characters have been transferred to the system output queue. This caveat applies independently to each string in the semicolon-separated list as well as to the end-of-line string, if applicable.   The latency-inducing effect described above can be mitigated by preparing a new output queue that is sufficiently large such that there is always enough free space in the queue when this method is invoked.

Examples

```
Debug.Print "Hello, world! "
```

This prints the given string followed by a carriage return/new line.

```
Debug.Print "The value is "; val;
```

This prints the string followed immediately by the string equivalent of the value. Note that since the command ends with a semicolon, no carriage return/new line will be generated.

```
Dim ba(1 to 10) as Byte
ba(1) = Asc("A")
ba(2) = Asc("B")
ba(3) = Asc("C")
ba(4) = 0
Debug.Print ba
```

This is equivalent to `Debug.Print "ABC"`.

Compatibility

This function is not available on VM mode devices nor in BasicX compatibility mode.

See Also   Console.Write, Console.WriteLine

DefineBus

Type	Subroutine		

Invocation   DefineBus(port, alePin, rdPin, wrPin)

Parameter	Method	Type	Description
port	ByVal	integral	The port to use for address and data. PortA=0, PortB=1, etc.
alePin	ByVal	integral	The pin to use for the address latch strobe.
rdPin	ByVal	integral	The pin to use for the read data strobe.
wrPin	ByVal	integral	The pin to use for the write data strobe.

Discussion

This subroutine is used to define the parameters to use for subsequent BusRead() and BusWrite() operations. The port specified by the `port` parameter is used both for outputting the address from which to read/write and for reading/writing the data. The port is specified by giving a port index – PortA = 0, PortB = 1, etc. You may use the built-in constants `Port.A`, `Port.B`, etc. to specify the port index. If all the parameters are valid, the pin specified by the `alePin` parameter is set to output low while the pins specified by the `rdPin` and `wrPin` parameters are set to output high. If any of the provided parameters is invalid, the bus will not be properly configured and subsequent calls to `BusRead()` or `BusWrite()` will return immediately with no effect.

The pin numbers specified for the `alePin`, `rdPin` and `wrPin` parameters must all be different and none of them should be in the port specified by the `port` parameter. If these conditions are violated, the result is undefined.

Example

```
Call DefineBus(Port.A, C.0, C.1, C.2)
```

Compatibility

This subroutine is not available on ZX models that are based on the ATmega32 processor (e.g. the ZX-24) nor is is available on ATxmega-based ZBasic devices. Moreover, it is not available in BasicX compatibility mode.

See Also   BusRead, BusWrite

DefineCom

Type   Subroutine

Invocation   DefineCom(channel, rxPin, txPin, flags)
                  DefineCom(channel, rxPin, txPin, flags, stopBits)

Parameter	Method	Type	Description
channel	ByVal	Byte	The serial channel being defined.
rxPin	ByVal	Byte	The pin which will serve as the receive line.
txPin	ByVal	Byte	The pin which will serve as the transmit line.
flags	ByVal	Byte	Configuration flags. See the discussion below.
stopBits	ByVal	Byte	The desired number of stop bits.

Discussion

This routine configures a serial channel, preparing it to be opened using `OpenCom()`. This routine may be called with the `channel` parameter specifiying either a hardware UART channel (1, 2 and 7-12) or a software UART channel (3-6). If the specified channel is already open, this routine does nothing. Likewise, there is no effect if the specified channel is invalid or if either of the `rxPin` and `txPin` parameters is invalid for the target device or the specified serial channel. Further, if the number of data bits specified in the *flags* parameter in invalid for the specified serial channel, this routine does nothing.

Hardware Channels

For hardware UART channels the `rxPin` and `txPin` parameters are meaningless and, therefore, ignored. By default, a hardware channel is opened in "8 data bits, no parity, 1 stop bit" mode (sometimes referred to as 8-N-1), the most common serial format. Unless you want to configure a hardware channel for a different mode it is not necessary to call DefineCom() before opening the channel. Specifying a value for the optional `stopBits` parameter that is greater than 1 will select 2 stop bits; otherwise, the default of 1 stop bit is used. Note that some of the bits in the `flags` parameter are ignored for hardware channels as indicated by shaded entries in the table below.

Software Channels

For software UART channels, either of the `rxPin` and `txPin` parameters (but not both) may be zero allowing you to define a transmit-only or receive-only serial. If the two parameter values are different, the specified pins are automatically configured as input and output, respectively. As a special case, the `rxPin` and `txPin` parameters may specify the same pin and the pin is initially configured as an input to support half-duplex, bussed operation. In this mode, the the pin will be made an output when transmitting a zero bit if configured for non-inverted operation or when transmitting a one bit if configured for inverted operation. A pull-up resistor (non-inverted mode) or pull-down resistor (inverted mode) is required for bussed operation since the pin will only be actively driven in one of the two output states.

If the optional `stopBits` parameter is not specified, one stop bit is transmitted for each character sent. Otherwise, the specified number of stop bits is transmitted. The allowable range for `stopBits` is 1 to 240 for software-based channels and 1 to 2 for hardware based channels. If a value outside the acceptable range is specified, the default of 1 stop bit will be used. The ability to specify two or more stop bits is useful for slowing down the transmission of data in cases where the receiver needs additional time to process received data.

Configuration Flags

The `flags` parameter contains several bit fields used to specify some of the details of the operation of the serial channel. Note, however, that some of the bits are applicable only to software UART channels.

Serial Channel Configuration Flag Values

Function	Hex Value	Bit Mask
Inverted Logic[1]	&H80	1x xx xxxx
Non-inverted Logic	&H00	0x xx xxxx
Ignore Parity Bit	&H40	x1 xx xxxx
Store Parity Bit	&H00	x0 xx xxxx
Even Parity	&H30	xx 11 xxxx
Odd Parity	&H20	xx 10 xxxx
No Parity	&H00	xx 00 xxxx
5-bit Data[2]	&H05	xx xx 0101
6-bit Data[2]	&H06	xx xx 0110
7-bit Data	&H07	xx xx 0111
8-bit Data	&H08	xx xx 1000
7-bit Data, bussed mode[1]	&H0b	xx xx 1011
8-bit Data, bussed mode[1]	&H0c	xx xx 1100

[1] Applicable only to software-based channels (3-6).
[2] Applicable only to hardware-based channels (1, 2, 7 and up).

The remaining bit values are currently undefined but may be employed in the future.

When Non-inverted Logic is selected, the idle state of the transmit line will be logic 1. When a character transmission is begun, a "start bit" of logic zero will be sent for one bit time (the inverse of the baud rate). Next the data bits are sent, each for one bit time, beginning with the least significant bit and continuing through the eighth data bit or parity bit as the case may be. Finally, one or more "stop bits" of logic one are sent, each for one bit time. With Inverted Logic, each of these elements is complemented – the idle state of the transmit line is logic 0.

Whether you should choose the inverted or non-inverted mode depends on the device that you intend to communicate with and how many, if any, level converters exist between the two devices. Typically, if the other device is capable of sending and receiving TTL-level serial data, you'll likely choose non-inverted Logic.

If the "Ignore Parity" flag is asserted, in 7-bit mode the most significant bit of each character received will be zero and in 8-bit mode only one byte will be stored in the queue for each character received. If the "Ignore Parity" bit is not asserted, in 7-bit mode the MSB will contain the received parity bit and in 8-bit mode a second byte containing the parity bit will be stored in the queue for each character received. The `ParityCheck()` function is useful for checking the parity of a received character.

The software UART channels support a bussed mode where the transmit pin is actively driven only during logic zero periods (low for non-inverted mode, high for inverted mode). This mode, selected by using the special values shown in the table above for 7-bit and 8-bit data widths (i.e., the normal values augmented by 4), is useful for having multiple devices driving the same transmit line. This mode is commonly referred to as *open drain* (non-inverted mode) or *open source* (inverted mode) operation and requires a pullup resistor (non-inverted mode) or a pulldown resistor (inverted mode) on the common transmit line in order to establish the proper logic level when the line is not being actively driven by any device.

Note that a pullup resistor (non-inverted mode) or a pulldown resistor (inverted mode) is recommended on the transmit line to force the transmit line to the idle state prior to the time your program initializes the COM port. If you don't do this, the receiving device may see false transmissions prior to the first character actually transmitted. Depending on what other circuitry is connected to the receive line, you may need to do the same to prevent the ZBasic device from receiving false transmissions.

Example

```
Call ComChannels(2, 9600)
Call DefineCom(4, 0, 12, &H08)
```

This call sequence prepares software-based channel 4 for transmit-only using pin 12, eight data bits, no parity and non-inverted logic.

```
Call DefineCom(2, 0, 0, &H07, 2)
Call OpenCom(2, 19200, iq, oq)
```

This call sequence prepares hardware-based channel 2 for seven data bits, no parity, and two stop bits.

Compatibility

This routine is not available in BasicX compatibility mode; you must use DefineCom3(). Additionally, BasicX does not support 8-bit plus parity modes nor does it support the "Ignore Parity" mode. Furthermore, in BasicX characters received in 7-bit/no parity mode are aligned toward the MSB while in this implementation they are properly aligned toward the LSB. The 5-bit and 6-bit modes are only supported for native mode targets.

For mega32-based ZX devices (e.g. the ZX-24), the ability to define the characteristics of Com1 is not supported nor is half-duplex bussed mode supported. Specifying the same pin for rx and tx on these devices will produce undefined results.

See Also   ComChannels, ControlCom, OpenCom, StatusCom

DefineCom3

Type   Subroutine

Invocation   DefineCom3(rxPin, txPin, flags)

Parameter	Method	Type	Description
rxPin	ByVal	Byte	The pin which will serve as the receive line.
txPin	ByVal	Byte	The pin which will serve as the transmit line.
flags	ByVal	Byte	Configuration flags. See the discussion below.

Discussion

This routine is provided solely for BasicX compatibility. It is equivalent to using `Call DefineCom(3, rxPin, txPin, flags)`. See the DefineCom`()` routine for more information.

DefineSPI

Type Subroutine

Invocation DefineSPI(clkPin, mosiPin, misoPin)

Parameter	Method	Type	Description
clkPin	ByVal	Byte	The pin to serve as the SPI clock signal (output).
mosiPin	ByVal	Byte	The pin to serve as the SPI MOSI signal (output).
misoPin	ByVal	Byte	The pin to serve as the SPI MISO signal (input).

Discussion

This subroutine is used to specify the clock and data pins to use for the software driven SPI implementation (sometimes known as a "bit banged" implementation). If the `flags` parameter to the OpenSPI subroutine requests software SPI, OpenSPI will initialize the specified pins (`clkPin` and `mosiPin` as output, `misoPin` as input) and set `clkPin` to the idle state specified by the `flags` parameter to OpenSPI. If software SPI is not requested, OpenSPI will initialize the hardware SPI controller according to the `flags` parameter to OpenSPI.

It is important to be aware that the pin values set by DefineSPI are used by both the OpenSPI and SPICmd routines. This fact requires some extra attention if your application uses multiple SPI channels and two or more of them use the software-driven implementation. In such cases, you must ensure that the SPI pins have been correctly set by a prior call to DefineSPI before each call to OpenSPI and SPICmd. If your application uses just one channel with software SPI, a single call to DefineSPI will suffice and if it does not use software SPI at all then DefineSPI needn't be called either.

Compatibility

This subroutine is not supported in BasicX mode nor it is supported on any VM mode ZX device.

See Also CloseSPI, OpenSPI, OpenSPISlave, SPICmd, SPIGetByte, SPIPutByte, SPIGetData, SPIPutData, SPIStart, SPIStop

DefineX10

Type	Subroutine
Invocation	DefineX10(channel, rxPin, txPin, flags)
	DefineX10(channel, rxPin, txPin, flags, agcResetPin, agcWindowPin)

Parameter	Method	Type	Description
channel	ByVal	Byte	The X-10 channel being defined. The valid range is 1-2.
rxPin	ByVal	Byte	The pin which will serve as the receive line.
txPin	ByVal	Byte	The pin which will serve as the transmit line.
flags	ByVal	Byte	Configuration flags. See the discussion below.
agcResetPin	ByVal	Byte	The pin on which to generate the AGC reset signal.
agcWindowPin	ByVal	Byte	The pin on which to generate the AGC window signal.

Discussion

This routine configures an X-10 communication channel, preparing it to be opened using `OpenX10()`. If the specified channel is already open, this routine does nothing. Likewise if the specified channel is invalid or if both the `rxPin` and `txPin` parameters are zero or invalid. Note that either `rxPin` or `txPin` may be zero, allowing you to define a transmit-only or a receive-only X-10 channel. If valid, the pins specified by `rxPin` and `txPin` are automatically configured as input and output, respectively.

The `flags` parameter contains several bit fields used to specify some of the details of the operation of the X-10 channel.

<div align="center">

Configuration Flags Bit Values

</div>

Function	Hex Value	Bit Mask
50Hz mode (only used in 3-phase mode)	&H40	x1 xx xx xx
60Hz mode (only used in 3-phase mode)	&H00	x0 xx xx xx
Three phase mode	&H20	xx 1x xx xx
Single phase mode	&H00	xx 0x xx xx
Attempt to detect transmit/receive collisions	&H10	xx x1 xx xx
LSB-first Transmit Bit Order	&H08	xx xx 1x xx
MSB-first Transmit Bit Order	&H00	xx xx 0x xx
Inverted Transmit Logic	&H04	xx xx x1 xx
Non-inverted Transmit Logic	&H00	xx xx x0 xx
LSB-first Receive Bit Order	&H02	xx xx xx 1x
MSB-first Receive Bit Order	&H00	xx xx xx 0x
Inverted Receive Logic	&H01	xx xx xx x1
Non-inverted Receive Logic	&H00	xx xx xx x0

The remaining bits are currently undefined but may be employed in the future.

When non-inverted modes are selected, the idle state of the transmit line or receive line will be logic 0. Whether you should choose the inverted or non-inverted mode depends on the interface circuitry that you use to connect to your X-10 transmitter/receiver.

When LSB-first modes are selected, the first bit to be sent/received will be the least significant bit of each byte. This is useful when a Bit array is used to assemble/decompose the data that is sent/received since the lower-indexed bits in a byte are of lower significance.

The second form with the additional parameters is provided for use with the CM15A and similar modules. The fifth parameter specifies a pin number on which to generate an active low signal to reset the CM15A AGC circuitry. The sixth parameter specifies a pin number on which to generate an active high signal

marking the CM15A AGC window, approximately 1mS following each zero crossing. Note that support for these CM15A AGC functions is disabled in 3 phase mode.

Example

```
Call DefineX10(1, 0, 12, &H00)
```

This call prepares channel 1 for transmit-only using pin 12, non-inverted logic, MSB-first operation.

Compatibility

This subroutine is not available on ZX models that are based on the ATmega32 processor (e.g. the ZX-24). The second form is only available on native mode devices. Neither form is available in BasicX compatibility mode.

Support for 3 phase mode is only available on native mode devices.

See Also   CloseX10, OpenX10, SetQueueX10, StatusX10

DegToRad

Type Function returning Single

Invocation DegToRad(angle)

Parameter	Method	Type	Description
angle	ByVal	Single	The angle, in degrees, to convert to radian measure.

Discussion

The trigonometric functions in the System Library all use radian angle measure. Depending on the programming task, it is sometimes more convenient to think of angles in terms of degrees. This function and its inverse RadToDeg() facilitate the conversion between the two systems.

Depending on optimization settings, if the parameter supplied to this function is known to be constant at compile time, the compiler converts the value at compile time.   Otherwise, code is generated to perform the conversion (multiplication by a conversion factor) at run time.

Example

```
Dim f as Single
Dim theta as Single     ' the angle in degrees

f = Sin(DegToRad(theta))
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also RadToDeg

Delay

Type	Subroutine
Invocation	Delay (time)

Parameter	Method	Type	Description
time	ByVal	Single	The amount of time to delay, in seconds.

Discussion

This routine suspends the current task for a period of time at least as long as specified. The actual delay depends on what other tasks actually do that may run in the interim. It is possible that the task will be suspended indefinitely depending on what another task might do. If the RTC is not enabled in your application, the resolution of the delay period is 1mS. If the RTC is enabled, the resolution is the same as an RTC tick period, i.e. `1/F_RTC_TICK` (typically 1.95mS for ZX devices).

Note that if the current task is locked, this call will unlock it.

There is a subtle difference between `Delay()` and `Sleep()` when the RTC is enabled and the arguments are non-zero. For `Delay()` the specified time is the minimum amount of delay that the task will experience assuming that no other task is ready to run and the actual delay could be up to 1 unit longer than the specified delay. For `Sleep()`, the specified time is the maximum amount of delay that the task will experience assuming that no other task is ready to run and the actual delay could be up to 1 unit less than the specified delay.

Example

```
Do
      Call PutPin(Pin.RedLED, 0)
      Call Delay(0.5)
      Call PutPin(Pin.RedLED, 1)
      Call Delay(0.5)
Loop
```

This loop causes the red LED to turn on and off alternately for a half second each.

Compatibility

The BasicX documentation specifically indicates that Delay() will unlock a locked task. However, tests indicate that this only happens if the parameter to Delay() is non-zero. This implementation unlocks a task on any Delay() call.

See Also   DelayMicroseconds, DelayMilliseconds, DelayUntilClockTick, Pause, Sleep, Register.RTCStopWatch

DelayCycles

Type   Subroutine

Invocation   DelayCycles(count)

Parameter	Method	Type	Description
count	ByVal	constant integral	The number of CPU cycles to delay.

Discussion

This routine effects a delay for at least as long as the specified number of cycles. The actual delay may be longer if interrupts are enabled, i.e. the delay will be extended by the amount of time required to service any interrupt that occurs. Further, if a task switch occurs (which will occur only if interrupts are enabled), the delay will be extended further until the task containing the delay resumes execution.

The delay is effected by executing a series of instructions in a loop repeatedly and possibly executing one or more NOP instructions to consume at least the specified number of cycles.

Example

```
Do
      Call PutPin(pin, 0)
      Call DelayCycles(20)
      Call PutPin(pin, 1)
      Call DelayCycles(20)
Loop
```

This loop causes the pin to go low and then high alternately for approximately 20 cycles each.

Compatibility

This subroutine is only available for native mode devices and is not available in BasicX compatibility mode.

See Also   Delay, DelayMilliseconds, DelayUntilClockTick, Pause, Sleep, Register.RTCStopWatch

DelayMicroseconds

Type   Subroutine

Invocation   DelayMicroseconds(usDelay)

Parameter	Method	Type	Description
usDelay	ByVal	constant integral	The amount of time to delay, in microseconds.

Discussion

This routine effects a delay for at least as long as the specified time. The actual delay depends on the particular processor clock frequency and whether interrupts are enabled. If interrupts are enabled, the delay will be extended by the amount of time required to service any interrupt that occurs. Further, if a task switch occurs (which will occur only if interrupts are enabled), the delay will be extended further until the task containing the delay resumes execution.

The delay is effected by executing a series of instructions in a loop repeatedly and possibly executing one or more NOP instructions to consume at least the specified delay time. As an example, requesting a 10 microsecond delay on a device running at 14.7456MHz will result in a series of instructions that consume 149 cycles; the actual delay being slightly more than 10.1 microseconds.

The minimum delay incurred by calling this subroutine is 1 CPU cycle. The maximum delay obtainable is approximately $(2^{24} * 5)$ cycles; slightly more than 5.6 seconds at 14.7MHz.

Example

```
Do
      Call PutPin(Pin.RedLED, 0)
      Call DelayMicroseconds(500000)
      Call PutPin(Pin.RedLED, 1)
      Call DelayMicroseconds(500000)
Loop
```

This loop causes the red LED to turn on and off alternately for approximately one half second each.

Compatibility

This subroutine is only available for native mode devices and is not available in BasicX compatibility mode.

See Also   Delay, DelayCycles, DelayMilliseconds, DelayUntilClockTick, Pause, Sleep, Register.RTCStopWatch

DelayMilliseconds

Type	Subroutine		
Invocation	DelayMilliseconds(msDelay)		

Parameter	Method	Type	Description
msDelay	ByVal	constant integral	The amount of time to delay, in milliseconds.

Discussion

This routine effects a delay for at least as long as the specified time. The actual delay depends on the particular processor clock frequency and whether interrupts are enabled. If interrupts are enabled, the delay will be extended by the amount of time required to service any interrupt that occurs. Further, if a task switch occurs (which will occur only if interrupts are enabled), the delay will be extended further until the task containing the delay resumes execution.

The delay is effected by executing a series of instructions in a loop repeatedly and possibly executing one or more NOP instructions to consume at least the specified delay time. As an example, requesting a 1 millisecond delay on a device running at 14.7456MHz will result in a series of instructions that consume 14,746 cycles; the actual delay being slightly more than 1.0 miilliseconds.

The minimum delay incurred by calling this subroutine is 1 CPU cycle. The maximum delay obtainable is approximately (2^24 * 5) cycles; slightly more than 5.6 seconds at 14.7MHz.

Example

```
Do
     Call PutPin(Pin.RedLED, 0)
     Call DelayMilliseconds(500)
     Call PutPin(Pin.RedLED, 1)
     Call DelayMilliseconds(500)
Loop
```

This loop causes the red LED to turn on and off alternately for approximately one half second each.

Compatibility

This subroutine is only available for native mode devices and is not available in BasicX compatibility mode.

See Also   Delay, DelayCycles, DelayMicroseconds, DelayUntilClockTick, Pause, Sleep, Register.RTCStopWatch

DelayUntilClockTick

Type   Subroutine

Invocation   DelayUntilClockTick()

Discussion

This routine suspends the current task until at least the next tick of the RTC. The actual delay depends on what other tasks actually do that may run in the interim. It is possible that the task will be suspended indefinitely.

If no other tasks are ready to run, the actual delay could be between 0 and 1 RTC tick.

This routine is exactly equivalent to `Sleep(1)`. However, the RTC must be enabled in your application in order to use this subroutine.

See Also   Delay, Pause, Sleep

DisableInt

Type   Function returning Byte

Invocation   DisableInt()

Discussion

This routine disables interrupts, preventing any interrupt source from interrupting the current task. Most commonly, this function is used to temporarily disable interrupts thereby allowing a sequence of instructions to execute without interruption. Of course, interrupts should be disabled for the shortest possible time in order to avoid missing important interrupts (e.g. real time clock interrupts). If interrupts are disabled for longer than one period of the RTC fast tick (i.e. `1/F_RTC_FAST`) you run the risk of missing an RTC tick which will result in the RTC losing time.

The most common use for DisableInt() is to implement "atomic access" to variables. This should be done for any variable that occupies multiple bytes of memory (e.g. `Integer`, `Long`, etc.) or for a read-modify-write operation on any variable when there is a possibility that another task or interrupt handler might attempt to access the same variable.

The value returned by DisableInt() should be passed to EnableInt(). Doing so will allow proper nesting of DisableInt() and EnableInt() calls.

Note

The Atomic block construct (described in the ZBasic Language Reference Manual) is the preferred method for implementing atomic access.

Example

```
Dim iflag as Byte

iflag = DisableInt()
' place code here that must not be interrupted
Call EnableInt(iflag)
```

See Also   EnableInt, UpdateRTC, Yield

DrainQueue

Type   Subroutine

Invocation   DrainQueue(queue, count)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue to be drained.
count	ByVal	anyIntegral	The number of bytes to remove.

Discussion

This routine removed up to the specified number of bytes of data from the queue.

Note that before any queue operations are performed, the queue data structure must be initialized. See the discussion of `OpenQueue()` for more details.

Example

```
Dim inQueue(1 to 40) as Byte

Call OpenQueue(inQueue, SizeOf(inQueue))
Call PutQueueStr(inQueue, "Hello")
Call DrainQueue(inQueue, 3)
```

After the call to DrainQueue() the queue will contain only l and o.

Compatibility

This routine is not available in BasicX mode.

EnableInt

Type   Subroutine

Invocation   EnableInt(flag)

Parameter	Method	Type	Description
flag	ByVal	Byte	The value controlling re-enabling of interrupts.

Discussion

This routine conditionally re-enables interrupts depending on the value of the `flag` parameter. If the most significant bit of the `flag` parameter is a 1, interrupts will be re-enabled. Otherwise, the state of the interrupt enabling will not change. Passing the value returned from DisableInt() implements proper nesting of DisableInt() and EnableInt() calls so they are most often used in pairs as shown in the example below.

Note

The Atomic block construct (described in the ZBasic Language Reference Manual) is the preferred method for implementing atomic access.

Example

```
Dim iflag as Byte

iflag = DisableInt()
' place code here that must not be interrupted
Call EnableInt(iflag)
```

See Also   DisableInt**,** UpdateRTC**,** Yield

ExitTask

Type	Subroutine
Invocation	ExitTask(taskStack)
	ExitTask()

Parameter	Method	Type	Description
taskStack	ByRef	array of Byte	The stack for a task of interest.

Discussion

This routine attempts to terminate an active task. If no task stack is explicitly given, the task stack for the `Main()` routine is assumed.

If this routine is invoked using an array other than one that is or was being used for a task stack the result is undefined.

See the section on Task Management in the ZBasic Reference Manual for additional information regarding task management.

When a task exits, whether normally or via ExitTask(), that task's status is first set to 254 indicating that it is in the process of exiting but that it is still in the task list. The exiting task will remain in the task list until the task manager runs again. The task manager runs whenever a task switch is called for but you can force it to run by invoking Sleep() or Yield(). Once the task manager removes an exiting task from the task list, its status will change to 255 indicating that it is fully terminated.

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   ResumeTask, RunTask, StatusTask

Exp

Type   Function returning Single

Invocation   Exp(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The power of e to be computed.

Discussion

This function returns the `Single` value corresponding to the value e raised to the specified power. The transcendental value e, upon which the natural logarithm is based, is approximately 2.718. This function is the inverse of the `Log()` function.

See Also   Exp10, Log, Log10,Pow

Exp10

Type   Function returning Single

Invocation   Exp10(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The power of 10 to be computed.

Discussion

This function returns the `Single` value corresponding to the value 10 raised to the specified power. This function is the inverse of the `Log10()` function.

See Also   Exp, Log, Log10,Pow

FirstTime

Type   Function returning Boolean

Invocation   FirstTime()

Discussion

When called the first time after downloading a program, this function will return True. Thereafter, it will always return False even if the processor is powered down or reset. Subsequently downloading again will again cause the function to return True on the first call, etc.

Fix

Type   Function returning Single

Invocation   Fix(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value to be "fixed".

Discussion

This function returns the `Single` representation of the integer that is nearest the supplied value, rounding toward zero.

Example

```
Dim f as Single

f = Fix(1.5)      ' result is 1.0
f = Fix(-1.5)     ' result is -1.0
```

See Also   Ceiling, Floor, Fraction

FixB

Type   Function returning Byte

Invocation   FixB(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value to be changed to integral form.

Discussion

The supplied `Single` value is first converted to a signed 32-bit integer, rounding toward zero, and then the low 8 bits of that value is returned. The result isn't particularly useful if the provided `Single` value is negative or larger than 255.

Example

```
Dim b as Byte

b = FixB(100.5)          ' result is 100
```

FixI

Type   Function returning Integer

Invocation   FixI(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value to be changed to integral form.

Discussion

The supplied `Single` value is first converted to a signed 32-bit integer, rounding toward zero, and then the low 16 bits of that value is returned. The result isn't particularly useful if the provided `Single` value is outside the range –32768 to 32767, inclusive.

Example

```
Dim i as Integer

i = FixI(-100.5)          ' result is -100
```

Compatibility

For compatibility with BasicX, if the provided `Single` value is larger than 32767 this function returns 32767. Similarly, if the value is less than –32767 (not –32768 as one would expect) this function returns –32767.

FixL

Type   Function returning Long

Invocation   FixL(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value to be changed to integral form.

Discussion

The supplied `Single` value is converted to a signed 32-bit integer, rounding toward zero, and that value is returned. The result isn't particularly useful if the provided `Single` value is outside the range –2,147,485,648 to 2,147,485,647, inclusive.

Example

```
Dim l as Long

l = FixL(-100.5)          ' result is -100
```

FixUI

Type   Function returning UnsignedInteger

Invocation   FixUI(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value to be changed to integral form.

Discussion

The supplied `Single` value is first converted to a signed 32-bit integer, rounding toward zero, and then the low 16 bits of that value is returned. The result isn't particularly useful if the provided `Single` value is outside the range 0 to 65535, inclusive.

Example

```
Dim ui as UnsignedInteger

ui = FixUI(100.5)        ' result is 100
```

FixUL

Type   Function returning UnsignedLong

Invocation   FixUL(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value to be changed to integral form.

Discussion

The supplied `Single` value is converted to a signed 32-bit integer, rounding toward zero, and that value is returned. The result isn't particularly useful if the provided `Single` value is outside the range 0 to 4,294,967,295, inclusive.

Example

```
Dim ul as UnsignedLong

ul = FixUL(100.5)        ' result is 100
```

FlipBits

Type   Function returning Byte

Invocation   FlipBits(arg)

Parameter	Method	Type	Description
arg	ByVal	Byte	The value to be bit-wise reversed.

Discussion

This function reverses the order of the bits in the supplied value and returns the result. This is useful, for example, if you want to send data using `ShiftOut()` but you want the least significant bit to be sent first.

Example

```
Dim b as Byte

b = &B1011_0110
b = FlipBits(b)      ' result is &B0110_1101
```

Floor

Type   Function returning Single

Invocation   Floor(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value of which to compute the floor.

Discussion

This function returns a `Single` value that is equal to the largest integer that is less than or equal to the supplied value, effectively rounding down to the nearest integer.

Example

```
Dim flr as Single

flr = Floor(1.5)   ' result is 1.0
flr = Floor(-1.5)  ' result is -2.0
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   Ceiling, Fix

Fmt

Type	Function returning String

Invocation	Fmt(val, fracDigits)

Parameter	Method	Type	Description
val	ByVal	Single	The value to convert to a string.
fracDigits	ByVal	Byte	The number of digits to produce following the decimal point.

Discussion

This function returns a `String` that represents the value of the `val` parameter.   The string will have a number of digits following the decimal point as specified by the `fracDigits` parameter. The maximum number of digits to the right of the decimal point is 6. If the `fracDigits` parameter specifies a larger number, it will be ignored and 6 will be used.

For very large and very small values, the returned string may be in scientific notation form. Also, some special cases are detected resulting in the strings shown in the table below. See the System Library function `SngClass()` for more information about the special values.

Special Value	Result[†]
NaN	"*.**".
±Infinity	"&.&&"
Denormalized value	"#.##"

[†]The number of special characters following the decimal point will be the same as the number of fraction digits that would have been generated had the value been normal.

Compatibility

In BasicX, the maximum number of fraction digits is 3 and the valid range of the value parameter is –999.0 to +999.0. If either of those ranges is exceeded, BasicX produces a string containing a single asterisk. Moreover, no provision is made for detecting special values such as NaN.

Fraction

Type   Function returning Single

Invocation   Fraction(val)

Parameter	Method	Type	Description
val	ByVal	Single	The value from which the fractional part will be returned.

Discussion

This function returns the fractional portion of the supplied value. The sign of the returned value will be the same as that of the value provided.

Example

```
Dim frac as Single

frac = Fraction(1.5)     ' result is 0.5
frac = Fraction(-1.5)    ' result is -0.5
```

Compatibility

This function is not available in BasicX compatibility mode.

FreqOut

Type   Subroutine

Invocation   FreqOut(pin, freqA, freqB, duration)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin on which the signal will be created.
freqA	ByVal	Integer	The primary frequency, in Hertz.
freqB	ByVal	Integer	The secondary frequency, in Hertz.
duration	ByVal	Single or Integer	The duration of the signal, in seconds or units. See the discussion below for more details.

Discussion

This routine generates a signal on the specified pin that is a digital approximation of two superimposed sine waves having the specified frequencies. The method used to produce the signal is a pseudo-PWM technique similar to that used for `DACPin()`. The output signal is actually purely digital, consisting of a series of precisely timed pulses that have an average value approximating that of two superimposed sine waves. This signal must be filtered to get an analog approximation. Depending on what you want to do with the signal, it may need to be amplified as well.

The duration of the signal may be specified in seconds by providing a Single value. Alternately, the time may be specified in units of approximately 1 millisecond by giving duration as an Integer or UnsignedInteger value. In either case, the valid range is approximately 1ms to 32 seconds.

Before beginning the frequency generation, the specified pin will be made an output. When the routine returns, the pin will still be an output.

If the pin is invalid, or both frequencies are zero, or the duration is zero, this routine does nothing. The maximum frequency that can be produced is approximately 14.4KHz. Requesting higher frequencies will produce undefined results.

Resource Usage

This routine uses the I/O Timer and disables interrupts until the signal generation is completed. RTC ticks are accumulated during the process so long signal durations should not cause a loss in RTC accuracy.

Example

```
Call FreqOut(pin, 440, 880, 5.0)   ' play middle C/high C for 5 seconds
```

Because of the high frequency nature of the pulse train used to synthesize the waveform some filtering is required. The example circuit below may be used to couple the output to a high impedance speaker (> 40) or an amplifier. Note, however, that the signal is too large to be fed to the microphone input of an amplifier. Instead, the Auxiliary or Line input should be used.
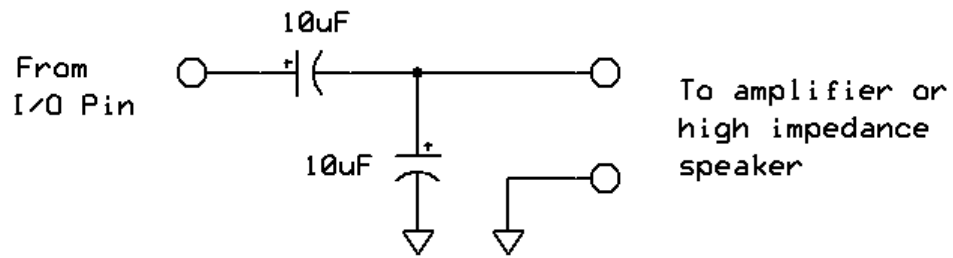
Compatibility

In BasicX, the RTC will lose time if the duration is longer than 1 millisecond. Also, the duration is documented as being limited to about 2.5 seconds

Get1Wire

Type   Function returning Byte

Invocation   Get1Wire(pin)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin to be used for 1-Wire I/O.

Discussion

This function retrieves a single bit using the 1-Wire protocol. To perform a 1-Wire operation, this function along with related 1-Wire routines must be used in the proper sequence. See the specifications of your 1-Wire device for more information.

The value returned will be either 0 or 1.

Resource Usage

This routine uses the I/O Timer and disables interrupts for approximately 100µS.

Example

```
Dim b as Byte

b = Get1Wire(12)
```

See Also   Get1WireByte, Get1WireData, Put1Wire,
                   Put1WireByte, Put1WireData, Reset1Wire

Get1WireByte

Type   Function returning Byte

Invocation   Get1WireByte(pin)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin to be used for 1-Wire I/O.

Discussion

This function reads a byte value (LSB first) using the 1-Wire protocol. It may be used instead of a series of calls to `Get1Wire()` in order to read a byte at a time. To perform a 1-Wire operation, this function along with related 1-Wire routines must be used in the proper sequence. See the specifications of your 1-Wire device for more information.

Resource Usage

This routine uses the I/O Timer and disables interrupts for about 100µS for each bit received.

Example

```
Dim b as Byte

b = Get1WireByte(12)
```

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   Get1Wire, Get1WireData, Put1Wire,
                  Put1WireByte, Put1WireData, Reset1Wire

Get1WireData

Type   Subroutine

Invocation   Get1WireData(pin, data, count)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin to be used for 1-Wire I/O.
data	ByRef	any type	The variable to receive the bytes read.
count	ByVal	Byte	The number of bytes to read.

Discussion

This function retrieves 1 or more bytes (each LSB first) using the 1-Wire protocol and writes them to the given variable. To perform a 1-Wire operation, this function along with related 1-Wire routines must be used in the proper sequence. See the specifications of your 1-Wire device for more information.

Caution

If the variable provided has fewer bytes than the given count, subsequent memory locations will be altered, usually with undesirable consequences.

Resource Usage

This routine uses the I/O Timer and disables interrupts for about 100µS for each bit received.

Example

```
Dim ba(1 to 10) as Byte

Call Get1WireData(12, ba, SizeOf(ba))
```

See Also   Get1Wire, Get1WireByte, Put1Wire,
Put1WireByte, Put1WireData, Reset1Wire

GetADC (subroutine form)

Type   Subroutine

Invocation   GetADC(pin, val)
GetADC(pin, val, fullScale)
GetADC(pin, val, fullScale, offset)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin from which to read an analog voltage.
Val	ByRef	Single	The variable in which to return the result.
fullScale	ByVal	Single	The full-scale voltage value.
offset	ByVal	Integral	The offset to apply before scaling.

Discussion

This function performs an analog-to-digital conversion on the signal present on the specified pin that must be one of the analog port pins (see Resource Usage below). The return value will be a 10-bit (for ATmega-based and ATtiny-based devices) or 12-bit (for ATxmega-based devices) digital approximation of the input voltage with a range from zero to the reference voltage (see below). For the first form, the returned value is scaled to the range 0.0 to 1.0 and for the remaining forms it is scaled to the range 0.0 to value of the *fullScale* parameter. For the third form, the value of the offset parameter (which could be negative) is added to the ADC value before scaling. This is useful, for example, for removing the effect of a non-zero offset voltage of the ADC.

You must make the pin an input before calling this routine.

For ATtiny and ATmega target devices, the conversion is performed using the AVcc reference voltage (connected internally to Vcc on the ZX-24, ZX-24a, ZX-24p, ZX-24n, ZX-24r, ZX-24s, ZX-24t, ZX-24e, ZX-24ae, ZX-24ne, ZX-24pe, ZX-24nu, ZX-24pu, ZX-24ru, ZX-24su, ZX-328nu, ZX-128e, ZX-128ne, ZX-1281e and ZX-1281ne). For ATxmega target devices, the conversion is performed using a reference voltage of Vcc/1.6.

Resource Usage

Only analog port pins may be used to perform an analog-to-digital conversion. The number and location of analog port pins vary depending on the ZBasic target device. See the section Analog-to-Digital Converters for more information.

Most ZBasic target devices contain a single analog-to-digital converter thus allowing only one conversion to be performed at a time (some have none at all). The conversion process takes approximately 220uS during which time the calling task will be awaiting conversion completion.

Compatibility

Although the BasicX manual indicates that that it is not necessary to configure the pin to be an input before calling, tests indicate that it is, in fact, necessary to do so. Consequently, the behavior of this implementation matches the actual behavior of the BasicX platform. The second and third forms are not available in BasicX mode.

GetADC (function form)

Type Function returning Integer

Invocation GetADC(pin)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin from which to read an analog voltage.

Discussion

This function performs an analog-to-digital conversion of the voltage present on the specified pin which must be one of the analog port pins (see Resource Usage below). The return value will be a 10-bit (for ATmega-based and ATtiny-based devices) or 12-bit (for ATxmega-based devices) digital approximation of the input voltage with a range from zero to the reference voltage (see below). The return value represents the measured voltage voltage according to the formula V_{ref} * adcVal / FS where V_{ref} is the reference voltage, adcVal is the value returned by GetADC(), and FS is 1024 for ATmega and ATtiny-based devices and 4096 for ATxmega-based devices.

You must make the specified pin an input before calling this routine.

For ATtiny and ATmega target devices, the conversion is performed using the AVcc reference voltage (connected internally to Vcc on the ZX-24, ZX-24a, ZX-24p, ZX-24n, ZX-24r, ZX-24s, ZX-24t, ZX-24e, ZX-24ae, ZX-24ne, ZX-24pe, ZX-24nu, ZX-24pu, ZX-24ru, ZX-24su, ZX-328nu, ZX-128e, ZX-128ne, ZX-1281e and ZX-1281ne).   For ATxmega target devices, the conversion is performed using a reference voltage of Vcc/1.6.

Resource Usage

Only analog port pins may be used to perform an analog-to-digital conversion. The number and location of analog port pins vary depending on the ZBasic target device. See the section Analog-to-Digital Converters for more information.

Most ZBasic target devices contain a single analog-to-digital converter thus allowing only one conversion to be performed at a time (some have none at all). The conversion process takes approximately 220uS during which time the calling task will be awaiting conversion completion.

Compatibility

Although the BasicX manual indicates that that it is not necessary to configure the pin to be an input before calling, tests indicate that it is, in fact, necessary to do so. Consequently, the behavior of this implementation matches the actual behavior of the BasicX platform.

GetBit

Type	Function returning Byte		

Invocation   GetBit(var, bitNumber)

Parameter	Method	Type	Description
var	ByRef	any type	The variable from which the bit will be read.
bitNumber	ByVal	int8/16	The bit number to read.

Discussion

This function extracts a single bit from memory beginning at the location of the specified variable. Bit numbers 0-7 are taken from the byte at the specified location, bit numbers 8-15 are taken from the subsequent byte, etc. In each case, the lower bit number corresponds to the least significant bit of the byte while the higher bit number corresponds to the most significant bit.

The return value will always be 0 or 1.

Compatibility

In BasicX compatibility mode the second parameter must be a `Byte` type.

See Also   PutBit

GetDate

Type	Subroutine
Invocation	GetDate(year, month, day)
	GetDate(year, month, day, dayNum)

Parameter	Method	Type	Description
year	ByRef	int16	The variable in which to place the year value (1999-2177).
month	ByRef	Byte	The variable in which to place the month value (1-12).
day	ByRef	Byte	The variable in which to place the day value (1-31).
dayNum	ByVal	integral	The day number to convert to year, month, day.

Discussion

This routine decomposes a day number into the corresponding year, month and day components. The month value of 1 corresponds to January while 12 corresponds to December. If the day number is omitted, the value of `Register.RTCDay` is used.

Note that `Register.RTCDay` is initialized to zero on power-up or reset. This day number corresponds to January 1, 1999.

Compatibility

This subroutine is not available if the RTC is not enabled in your application. Also, the second form of this subroutine is not available in BasicX compatibility mode.

See Also   GetDateValue, GetDayNumber, GetDayOfWeek, GetDayOfYear, PutDate

GetDateValue

Type   Function returning UnsignedInteger

Invocation   GetDateValue()
GetDateValue(dayNum)

Parameter	Method	Type	Description
dayNum	ByVal	integral	The day number to convert to year, month, day.

Discussion

This function decomposes a day number into the corresponding year, month and day components and packs them into a 16-bit value as shown in the table below. If the day number is omitted, the value of `Register.RTCDay` is used.

Note that `Register.RTCDay` is initialized to zero on power-up or reset. This day number corresponds to January 1, 1999.

Date Value Fields

Bits	Position Mask	Description
15-9	&Hfe00	Year relative to 1999 (0 to 127)
8-5	&H01e0	Month (1 to 12)
4-0	&H001f	Day (1 to 31)

Compatibility

This subroutine is not available if the RTC is not enabled in your application. Also, the second form of this subroutine is not available in BasicX compatibility mode.

See Also   GetDate, GetTime, GetTimeValue

GetDayNumber

Type   Function returning UnsignedInteger

Invocation   GetDayNumber(dayOfYear, year)
                GetDayNumber(year, month, day)

Parameter	Method	Type	Description
dayofYear	ByVal	integral	The ordinal day number of the year (Jan 1 = 1).
year	ByVal	integral	The year (1999 to 2178).
month	ByVal	integral	The month (1 to 12).
day	ByVal	integral	The day (1 to 31).

Discussion

This routine computes the day number corresponding to the day of the year specified by the parameters. Day number 0 is January 1, 1999. The first form is used when you have a day number and year. (The days in a year are numbered beginning with 1.) The second form is used when you have the year, month and day.

Examples

```
Dim dayNum as UnsignedInteger

dayNum = GetDayNumber(59, 2005)
dayNum = GetDayNumber(2006, 3, 20)
```

Compatibility

This function is not available if the RTC is not enabled in your application. Also, it is not available in BasicX compatibility mode.

See Also   GetDate, GetDayOfWeek, GetDayOfYear, PutDate

GetDayOfWeek

Type	Function returning Byte

Invocation	GetDayOfWeek()
	GetDayOfWeek(dayNum)

Parameter	Method	Type	Description
dayNum	ByVal	integral	The day number to convert to year, month, day.

Discussion

This routine computes the day of the week corresponding to a day number. If the day number is omitted, the value of `Register.RTCDay` is used.. A return value of 1 corresponds to Sunday and a value of 7 corresponds to Saturday with the remaining days falling in order in between. There are built-in constants that represent the day numbers as shown in the table below.

Day of Week Constants

Constant	Value
zxSunday	1
zxMonday	2
zxTuesday	3
zxWednesday	4
zxThursday	5
zxFriday	6
zxSaturday	7

Note that `Register.RTCDay` is initialized to zero on power-up or reset. This day number corresponds to Friday, January 1, 1999.

Compatibility

This function is not available if the RTC is not enabled in your application.

See Also   GetDate, GetDayNumber, GetDayOfYear

GetDayOfYear

Type   Function returning UnsignedInteger

Invocation   GetDayOfYear(dayNum)
                  GetDayOfYear(dayNum, year)

Parameter	Method	Type	Description
dayNum	ByVal	integral	The day number to convert to day of year and year.
year	ByRef	int16	The variable in which the year will be stored.

Discussion

This routine computes the day of the year and the year corresponding to a day number (such as represented by `Register.RTCDay`). The first day of the year is numbered 1. If the second parameter is present, the variable to which it refers will receive the year value.

Example

```
Dim dayOfYear as UnsignedInteger
Dim year as UnsignedInteger

dayOfYear = GetDayOfYear(Register.RTCDay, year)
```

Compatibility

This function is not available if the RTC is not enabled in your application. Also, it is not available in BasicX compatibility mode.

See Also   GetDate, GetDayNumber, GetDayOfWeek

GetEEPROM

Type   Subroutine

Invocation   GetEEPROM(addr, var, count)

Parameter	Method	Type	Description
addr	ByVal	Long	The Program Memory address from which to begin reading.
var	ByRef	any type	The variable in which to place the data read.
count	ByVal	int16	The number of bytes to read.

Discussion

This routine is provided for compatibility with BasicX. The more aptly named GetProgMem() should be used by new applications.

See Also   GetProgMem, PutProgMem

GetElapsedMicroTime

Type   Function returning UnsignedLong

Invocation GetElapsedMicroTime(timeBuf)
                GetElapsedMicroTime(timeBuf, timeBuf2)

Parameter	Method	Type	Description
timeBuf	ByRef	Microtime_t structure or array of Byte	The earlier time data.
timeBuf2	ByRef	Microtime_t structure or array of Byte	The later time data.

Discussion

This function is useful for implementing higher precision timing than can be obtained using
Register.RTCTick. It calculates the elapsed time between an earlier instant in time (as captured by
GetMicroTime()) and a later instant in time. If the second parameter is not provided, the later instant is
represented by the RTC time data at the time of the call.

The pre-defined structure, Microtime_t, can be incorporated in your application using the directive
Option Include Microtime_t. Using this structure instead of an array of bytes is preferable not
least because it automatically adapts if you change the target device.

The return value has units of the period of the frequency at which the TCNT register of the RTC timer
changes, i.e. $1/F\_RTC\_TIMER$ (typically about 4.34uS for ZX devices). The value of
Register.RTCTimerFrequency may be useful for converting the return value to seconds.

The array must contain at least 5 bytes (6 bytes for xmega devices), populated by a previous call to
GetMicroTime(). The return value will range from 0 to the equivalent of about 15,000 seconds. A
return value of &HFFFFFFFF indicates that an overflow has occurred, i.e. an elapsed time that is too
large to represent.

Although this function does not take into account the value of the "day" counter of the RTC, it does
properly handle an elapsed time that spans one midnight rollover.

Example

```
Dim t0(1 to 5) as Byte  ' must be 6 bytes for xmega devices
Call GetMicroTime(t0)
<other code>
Dim delta as UnsignedLong
delta = GetElapsedMicroTime(t0)
```

Compatibility

This function is not available on ZX models that are based on the ATmega32 processor (e.g. the ZX-24)
nor is it available if the RTC is not enabled in your application. Moreover, it is not available in BasicX
compatibility mode.

See Also   GetMicroTime

GetMicroTime

Type   Subroutine

Invocation   GetMicroTime(timeBuf)

Parameter	Method	Type	Description
timeBuf	ByRef	Microtime_t structure or array of Byte	A buffer to be populated with time data.

Discussion

This routine populates the provided buffer, which must be at least 5 bytes long (6 bytes for xmega devices), with high resolution timing data. This information is most useful in conjunction with a subsequent call to `GetElapsedMicroTime()` to compute an elapsed time.

The data in the time buffer comprises of the value of the TCNT register of the RTC Timer at the moment of the call followed by the RTC tick value converted to "fast ticks" (that typically occur at 1024Hz for ZX devices).

The pre-defined structure, Microtime_t, can be incorporated in your application using the directive `Option Include Microtime_t`. Using this structure instead of an array of bytes is preferable not least because it automatically adapts if you change the target device.

Example

```
Dim start(1 to 5) as Byte  ' must be 6 bytes for xmega devices
Dim mt0 as Microtime_t
Call GetMicroTime(start)
Call GetMicroTime(mt0)
```

Compatibility

This subroutine is not available on ZX models that are based on the ATmega32 processor (e.g. the ZX-24) nor is it available if the RTC is not enabled in your application. Moreover, it is not available in BasicX compatibility mode.

See Also   GetElapsedMicroTime

GetNibble

Type   Function returning Nibble

Invocation   GetNibble(var, nibbleNumber)

Parameter	Method	Type	Description
var	ByRef	any type	The variable from which the nibble will be read.
nibbleNumber	ByVal	int8/16	The nibble number to read.

Discussion

This function extracts a nibble value from memory beginning at the location of the specified variable. Nibble numbers 0-1 are taken from the byte at the specified location, nibble numbers 2-3 are taken from the subsequent byte, etc. In each case, the lower nibble number corresponds to the least significant four bits of the byte while the higher nibble number corresponds to the most significant four bits of the byte.

The return value will always be in the range 0 to 15.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   PutNibble

GetPersistent

Type	Subroutine	
Invocation	GetPersistent(addr, var, count)	

Parameter	Method	Type	Description
addr	ByVal	int16	The address in Persistent Memory from which to read.
var	ByRef	any type	The variable in which to place the data read.
count	ByVal	int8/16	The number of bytes to read.

Discussion

This routine reads one or more bytes from Persistent Memory and places them in RAM beginning at the location of the specified variable. Note that if a number of bytes is specified that is larger than the given variable, adjacent memory will be overwritten, possibly with detrimental results.

The DataAddress property is useful to get the address of a Persistent Memory data item.

Example

```
Dim pvar(1 to 10) as PersistentByte
Dim var(1 to 10) as Byte

Call GetPersistent(pvar.DataAddress, var, SizeOf(pvar))
```

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   PutPersistent

GetPin

Type   Function returning Byte

Invocation   GetPin(pin)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin to read.

Discussion

If the specified pin is configured to be an input, this function reads the state of the pin and returns the value 0 or 1 corresponding to logic zero and logic one. If the pin number is invalid the result is undefined. If the pin is configured to be an output, it is reconfigured to be an input in tri-state mode before reading the input value.

Compatibility

The BasicX documentation says that the result is undefined if `GetPin()` is called for a pin that is configured as an output. Tests show that the pin is actually reconfigured to be an input in tri-state mode. The ZBasic implementation of `GetPin()` does the same.

See Also   PinRead**,** PutPin

GetProgMem

Type   Subroutine

Invocation GetProgMem(addr, var, count)

Parameter	Method	Type	Description
addr	ByVal	Long	The Program Memory address from which to begin reading.
var	ByRef	any type	The variable in which to place the data read.
count	ByVal	int16	The number of bytes to read.

Discussion

This routine reads one or more bytes from Program Memory (where the user program is stored) and places them in RAM beginning at the location of the specified variable. Note that if a number of bytes is specified that is larger than the given variable, adjacent memory will be overwritten, possibly with detrimental results.

See Also PutProgMem

GetQueue

Type   Subroutine

Invocation   GetQueue(queue, var, count)
                      GetQueue(queue, var, count, timeLimit, timeoutFlag)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue from which to read data.
var	ByRef	any type	The variable to which to write the data from the queue.
count	ByVal	int16	The number of bytes to read from the queue.
timeLimit	ByVal	Single	The amount of time to wait for data availability, in seconds.
timeoutFlag	ByRef	Boolean	A variable to indicate if the call timed out.

Discussion

This routine has two forms. The first form simply attempts to read the given number of bytes from the specified queue and place them in RAM beginning at the location of the given variable. In this case, the routine will not return until requested number of bytes is available. If not enough data is placed in the queue, the routine will never return. Note that if the calling task is locked and the queue contains insufficient space for the data to be written data when this routine is called, the task will be unlocked to allow other tasks to run.

The second form specifies, additionally, a `timeLimit` and a `flag` variable. In this case, if the requested number of bytes does not become available within the specified time, the routine will return, having transferred zero bytes, and the `flag` variable will be set to `True` indicating that the routine timed out. If the requested number of bytes does become available before the specified time expires, that number of bytes will be removed from the queue and transferred to the specified memory location and the `flag` variable will be set to `False` indicating that the transfer did not time out. The resolution of the timeout value is the same as the RTC tick, approximately 1.95mS.

In either case, if data is removed from the queue it is written to RAM beginning at the location of the specified variable. Note that if the count specifies a number of bytes larger than the variable, the additional bytes will be written to subsequent RAM locations. This may have exactly the effect that you intended but depending on the function of those subsequent bytes it may have a deleterious effect on your program.

Note that before any queue operations are performed, the queue data structure must be initialized. See the discussion of `OpenQueue()` for more details. Also, attempting to retrieve data from a queue that has been assigned to a Com port as the transmit queue will produce undefined results.

Although this subroutine will accept a String variable as the second parameter it is generally not useful to do so because the control bytes at the beginning of the string will be overwritten. If you want to populate a string using data from a queue the alternatives are:

1) Build up the string by retrieving individual characters one by one and appending them to a string.
2) Retrieve a group of bytes to a Byte array and use the `MakeString()` function to create a string from the constituent bytes.
3) Use the `GetQueueStr()` function to obtain a string containing characters from the queue.

Example

```
Dim inQueue(1 to 40) as Byte
Dim lval as Long

Call OpenQueue(inQueue, SizeOf(inQueue))
```

```
Call GetQueue(inQueue, lval, SizeOf(lval))
```

Alternately,

```
Dim inQueue(1 to 40) as Byte
Dim lval as Long
Dim timeOut as Boolean

Call OpenQueue(inQueue, SizeOf(inQueue))
Call GetQueue(inQueue, lval, SizeOf(lval), 1.0, timeOut)
```

Compatibility

BasicX allows any type for the first parameter. The ZBasic implementation requires that it be an array of `Byte`.

The BasicX manual indicates that the range of values for the timeLimit parameter is 0.0 to 65.536 seconds implying a 1ms resolution. This implementation has a 1.95ms resolution and a range of 0.0 to about 127.0 seconds.

See Also   GetQueueStr, OpenQueue

GetQueueBufferSize

Type	Function returning Integer
Invocation	GetQueueBufferSize(queue)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue of interest.

Discussion

This function returns the number of bytes of data space in a queue that has been properly initialized using `OpenQueue()`. Note that the data space in a queue is somewhat less than the number of bytes in the byte array comprising the queue due to space required for queue management information. See `OpenQueue()` for more details.

Compatibility

Basic X allows any type for the first parameter. The ZBasic implementation requires that it be an array of `Byte`.

See Also GetQueueCount, GetQueueSpace

GetQueueCount

Type   Function returning Integer

Invocation   GetQueueCount(queue)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue of interest.

Discussion

This function returns the number of bytes of data currently in the specified queue. It is useful to note that this value subtracted from that returned by `GetQueueBufferSize()` indicates the remaining available data space in the queue.

Note that before any queue operations are performed, the queue data structure must be initialized. See the discussion of `OpenQueue()` for more details.

Compatibility

BasicX allows any type for the first parameter. The ZBasic implementation requires that it be an array of `Byte`.

See Also   GetQueueBufferSize, GetQueueSpace

GetQueueSpace

Type   Function returning Integer

Invocation   GetQueueSpace(queue)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue of interest.

Discussion

This function returns the number of bytes of space remaining in the specified queue, effectively the same result as the expression `GetQueueBufferSize() – GetQueueCount()`.

Note that before any queue operations are performed, the queue data structure must be initialized. See the discussion of `OpenQueue()` for more details.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   GetQueueBufferSize, GetQueueCount

GetQueueStr

Type   Function returning String

Invocation   GetQueueStr(queue) or
                  GetQueueStr(queue, maxChars)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue of interest.
maxChars	ByVal	integral	The maximum number of characters to retrieve.

Discussion

This function extracts a number of characters from the specified queue and returns a string populated with those characters. The number of characters is limited to the lesser of 1) the number of characters in the queue at the time of the call, 2) the value of `maxChars` (if specified), and 3) the maximum number of characters allowed in a string.

Note that before any queue operations are performed, the queue data structure must be initialized. See the discussion of `OpenQueue()` for more details. Also, attempting to retrieve data from a queue that has been assigned to a Com port as the transmit queue will produce undefined results.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   GetQueue, OpenQueue

GetTime

Type   Subroutine

Invocation   GetTime(hour, minute, seconds)
                GetTime(hour, minute, seconds, tick)

Parameter	Method	Type	Description
hour	ByRef	Byte	The variable in which to place the hour value (0-23).
minute	ByRef	Byte	The variable in which to place the minutes value (0-59).
seconds	ByRef	Single	The variable in which to place the seconds value.
tick	ByVal	integral	The tick count to decompose.

Discussion

This routine decomposes a tick count into the equivalent hour, minute and second components. If the tick count is omitted, the value of `Register.RTCTick` is used. The resolution of the `seconds` value is `1/F_RTC_TICK` (typically 1.95ms for ZX devices).

Note that `Register.RTCTick` is initialized to zero on power-up or reset. This corresponds to 0:00:00.

Compatibility

This subroutine is not available if the RTC is not enabled in your application. Also, explicitly specifying the tick count to use (fourth parameter) is not supported in BasicX compatibility mode.

See Also   GetDate, GetTimestamp, GetTimeValue

GetTimestamp

Type   Subroutine

Invocation GetTimestamp(year, month, day, hour, minute, seconds)

Parameter	Method	Type	Description
year	ByRef	int16	The variable in which to place the year value (1999-2177).
month	ByRef	Byte	The variable in which to place the month value (1-12).
day	ByRef	Byte	The variable in which to place the day value (1-31).
hour	ByRef	Byte	The variable in which to place the hour value (0-23).
minute	ByRef	Byte	The variable in which to place the minutes value (0-59).
seconds	ByRef	Single	The variable in which to place the seconds value.

Discussion

This routine decomposes the value of `Register.RTCDay` and `Register.RTCTick` into year, month, day, hour, minute and second components. See `GetDate()` and `GetTime()` for more details.

Compatibility

This subroutine is not available if the RTC is not enabled in your application.

GetTimeValue

Type   Function returning UnsignedLong

Invocation   GetTimeValue()
                  GetTimeValue(tick)

Parameter	Method	Type	Description
tick	ByVal	integral	The tick count to decompose.

Discussion

This function decomposes a tick count into the equivalent hour, minute, second and fractional second components and packs them into a 32-bit value as shown below. If the tick count is omitted, the value of `Register.RTCTick` is used.

Note that `Register.RTCTick` is initialized to zero on power-up or reset. This corresponds to 0:00:00.

Time Value Fields

Bits	Position Mask	Description
31-27	&Hf8000000	Hour (0 to 23)
26-21	&H07e00000	Minute (0 to 59)
20-15	&H001f8000	Second (0 to 59)
14-0	&H00007fff	Fractional Second (0 to F_RTC_TICK-1)

The 'fractional second' field represents the accumulation of RTC ticks between each second. The floating point equivalent of the seconds value represented by the converted time value can be calculated using code like that shown in the example below.

Example

```
Dim time as UnsignedLong
Dim seconds as Single

' compute the equivalent full and fractional seconds value
time = GetTimeValue()
seconds = CSng(Shr(time, 15) And &H3f) + _
    CSng(time And &H7fff) / CSng(Register.RTCTickFrequency)
```

Compatibility

The first form of the function is not available if the RTC is not enabled in your application. Neither form is supported in BasicX compatibility mode or on VM mode ZX devices.

See Also   GetDate, GetDateValue, GetTime, GetTimestamp

HiByte

Type   Function returning Byte

Invocation HiByte(val)

Parameter	Method	Type	Description
val	ByVal	numeric	The value of which the high byte is desired.

Discussion

This function returns the most significant byte of the specified value except that if the specified value is a Byte value, the result will be zero.

Compatibility

This function is not available in BasicX compatibility mode.

See Also HiWord, LoByte, LoWord, MidWord

HiWord

Type   Function returning UnsignedInteger

Invocation   HiWord(val)

Parameter	Method	Type	Description
val	ByVal	numeric	The value of which the high word is desired.

Discussion

This function returns the most significant word of the specified value except that if the specified value is a Byte, Integer or UnsignedInteger value, the result will be zero.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   HiByte, LoByte, LoWord, MidWord

I2CCmd

Type	Function returning Integer

Invocation	I2CCmd(channel, slaveID, writeCnt, writeData, readCnt, readData)

Parameter	Method	Type	Description
channel	ByVal	Byte	The I2C channel number (0-4).
slaveID	ByVal	Byte	The identifier of the I2C slave device (in the 7 high order bits).
writeCnt	ByVal	integral	The number of bytes to write (0 – 65535).
writeData	ByRef	any type	The variable containing the data to write.
readCnt	ByVal	integral	The number of bytes to read (0 – 65535).
readData	ByRef	any type	The variable in which to place the data read.

Discussion

The routine allows you to send and/or receive data from an I2C device. The specified channel must have been previously opened with a call to `OpenI2C()`. If the channel has not been opened, the results are undefined. If an invalid channel is specified or if both `writeCnt` and `readCnt` are zero, the function returns immediately without doing anything and the return value is zero. You may specify the value 0 for `writeData` or `readData` if no data is being provided for writing or reading, respectively. If you do this, the corresponding data count parameter must also be zero or the compiler will issue an error message.

The execution of the I2C command sequence begins by issuing an I2C start condition on the SDA and SCL lines. Next, if `writeCnt` is non-zero the given `slaveID` value is transmitted (with the least significant bit being zero) followed by the specified number of bytes taken from `writeData`. Then, if `readCnt` is non-zero the `slaveID` value is transmitted again but with the least significant bit being one and the specified number of bytes is read from the slave and placed in `readData`. Finally, an I2C stop condition is issued followed by both the SDA and SCL lines returning to the idle state.

The return value may be negative, zero or positive. If the return value is negative it signifies that the slave failed to positively acknowledge one of the transmitted bytes. The value is the negative of the number of bytes that were not successfully transmitted. If the slave fails to positively acknowledge either the slave ID or the first data byte, the return value will be the negative of the `writeCnt` parameter value. If the return value is non-negative it represents the number of data bytes read from the slave and placed in `readData`.

Example

```
Dim odata(1 to 2) as Byte, idata(1 to 10) as Byte
Dim ival as Integer

Call OpenI2C (1, 12, 13)
odata(1) = &H06
odata(2) = &H00
ival = I2CCmd(1, &H7e, 2, odata(1), 10, idata(1))
```

Resource Usage

This function uses the I/O Timer for channels 1 to 4. If the timer is already in use, the result and the return value are both undefined. Interrupts are disabled for periods of about 9 times the selected I2C bit time plus additional amounts due to slave clock stretching for each byte sent and received (interrupts are reenabled between bytes). However, RTC ticks are accumulated during the process so the RTC should not lose time.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   OpenI2C, I2CGetByte, I2CPutByte, I2CStart, I2CStop, CloseI2C

I2CGetByte

Type	Function returning Byte		

Invocation	I2CGetByte(channel, ackValue)		

Parameter	Method	Type	Description
channel	ByVal	Byte	The I2C channel number (0-4).
ackValue	ByVal	Boolean	The value to send to the slave in acknowledgement of the data byte.

Discussion

This function retrieves a data value from an I2C slave and responds to the receipt of that data by sending back the specified acknowledgement value. The value returned by this function is the data byte received from the slave.

This function can be used in conjunction with `I2CStart()`, `I2CPutByte()` and `I2CStop()` to perform a lower level interaction with an I2C slave device. Knowledge of the I2C protocol and the specifications of the particular I2C device are required in order to use this function.

If the specified I2C channel has not been properly prepared using `OpenI2C()`, the results are undefined. If an invalid channel number is specified, the function returns immediately without doing anything.

Resource Usage

This function uses the I/O Timer for channels 1 to 4. If the timer is already in use, the function will do nothing and the return value is undefined. Interrupts are disabled for about 9 times the selected I2C bit time plus additional amounts due to slave clock stretching. However, RTC ticks are accumulated during the process so the RTC should not lose time.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   OpenI2C, CloseI2C, I2CPutByte, I2CStart, I2CStop, I2CCmd

I2CPutByte

Type   Function return Boolean

Invocation   I2CPutByte(channel, dataVal)

Parameter	Method	Type	Description
channel	ByVal	Byte	The I2C channel number (0-4).
dataVal	ByVal	Byte	The data byte to send to the slave.

Discussion

This function transmits a data value to an I2C slave and reads the acknowledgement bit returned by the slave. The value returned by this function is the value of the acknowledge bit received from the slave device – a positive acknowledgement results in a True value being returned.

This function can be used in conjunction with `I2CStart()`, `I2CGetByte()` and `I2CStop()` to perform a lower level interaction with an I2C slave device. Knowledge of the I2C protocol and the specifications of the particular I2C device are required in order to use this function.

If the specified I2C channel has not been properly prepared using `OpenI2C()`, the results are undefined. If an invalid channel number is specified, the function returns immediately without doing anything.

Resource Usage

This function uses the for channels 1 to 4. If the timer is already in use, the function will do nothing and the return value is undefined. Interrupts are disabled for about 9 times the selected I2C bit time plus additional amounts due to slave clock stretching. However, RTC ticks are accumulated during the process so the RTC should not lose time.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   OpenI2C, CloseI2C, I2CGetByte, I2CStart, I2CStop, I2CCmd

I2CStart

Type	Subroutine
Invocation	I2CStart(channel)

Parameter	Method	Type	Description
channel	ByVal	Byte	The I2C channel number (0-4).

Discussion

This subroutine initiates an I2C bus cycle by implementing the proper sequence of transitions on the SDA and SCL lines.

This subroutine can be used in conjunction with `I2CGetByte()`, `I2CPutByte()` and `I2CStop()` to perform a lower level interaction with an I2C slave device. Knowledge of the I2C protocol and the specifications of the particular I2C device are required in order to use this function.

If the specified I2C channel has not been properly prepared using `OpenI2C()`, the results are undefined. If an invalid channel number is specified, the function returns immediately without doing anything.

Compatibility

This subroutine is not available in Basic X compatibility mode.

See Also   OpenI2C, CloseI2C, I2CGetByte, I2CPutByte, I2CStop, I2CCmd

I2CStop

Type   Subroutine

Invocation   I2CStop(channel)

Parameter	Method	Type	Description
channel	ByVal	Byte	The I2C channel number (0-4).

Discussion

This subroutine terminates an I2C bus cycle by implementing the proper sequence of transitions on the SDA and SCL lines.

This subroutine can be used in conjunction with `I2CStart()`, `I2CGetByte()` and `I2CPutByte()` to perform a lower level interaction with an I2C slave device. Knowledge of the I2C protocol and the specifications of the particular I2C device are required in order to use this function.

If the specified I2C channel has not been properly prepared using `OpenI2C()`, the results are undefined. If an invalid channel number is specified, the function returns immediately without doing anything.

Compatibility

This subroutine is not available in Basic X compatibility mode.

See Also   OpenI2C, CloseI2C, I2CGetByte, I2CPutByte, I2CStart, I2CCmd

IIf

Type	Function returning the same type as the second parameter

Invocation	IIf(testExpr, trueExpr, falseExpr)

Parameter	Method	Type	Description
testExpr	ByVal	Boolean	The expression to evaluate, the result of which determine which expression value will be returned.
trueExpr	ByVal	any type	The value to return if *testExpr* evaluates to True.
falseExpr	ByVal	any type	The value to return if *testExpr* evaluates to False.

Discussion

This function is adapted from VB6 where it is sometimes called "Immediate If". It is used to select one of two values based on the result of a test. Employing this function will generally result in less code than an equivalent If-Then-Else structure. On the other hand, the execution of this function does use more stack space than an equivalent If-Then-Else structure. Also, it is important to note that using this function is not exactly the same as an If-Then-Else because both the *trueExpr* and the *falseExpr* are always evaluated. This difference is only significant if the evaluation of one or both of these expressions has side effects.

Note that *trueExpr* and *falseExpr* must have the same type or be of compatible types.

Examples

```
Dim a as Byte
Dim b as UnsignedInteger
Dim u as UnsignedInteger

u = IIf(a > 3, 5, b)

Debug.Print IIf(a = 5, "Hello", "Goodbye")
```

Compatibility

This function is not available in BasicX compatibility mode. Also, it is only supported by ZX firmware v1.1.0 or later.

InputCapture

Type   Subroutine

Invocation   InputCapture(data, count, flags)
                  InputCapture(data, count, flags, timeout)

Parameter	Method	Type	Description
data	ByRef	array of UnsignedInteger	The array in which pulse width information will be stored.
count	ByVal	int16	The number of pulse widths to store. This should be no larger than the number of entries in the passed array.
flags	ByVal	Byte	A value of zero requests that a falling edge begin the capture process while a value of 1 indicates a rising edge. All other values are reserved.
timeout	ByVal	Integral	If non-zero, this parameter specifies a timeout value that, if exceeded, will terminate the input capture process.

Discussion

Invoking this routine is equivalent to the call `InputCaptureEx(pin, data, count, flags)` or `InputCaptureEx(pin, data, count, flags, timeout)` where `pin` is the default input capture pin for the device as shown in the table below. See the description of InputCaptureEx() for more detailed information. Also, see the section Input Capture Timers for information on the default input capture pin used by this subroutine.

The stored values represent the number of I/O Timer ticks (i.e. $1/F\_CPU$ or about 67.8ns for 14.7MHz devices) measured for each segment of the pulse train. However, the value of `Register.TimerSpeed1` may be changed to allow longer pulse widths to be measured. See the section on Timers for more information.

Example

```
Dim pd(1 to 5) as UnsignedInteger

Call PutPin(12, zxInputTriState)
Call InputCapture(pd, UBound(pd), 1)
```

Compatibility

The BasicX compiler erroneously allows any variable for the first parameter. This implementation requires the data type to be `UnsignedInteger` or `Integer` although it needn't be an array. For practical purposes, an array will almost always be used.

In BasicX compatibility mode, the use of the optional fourth parameter is not supported. Also, because the processor runs at twice the speed of the BX-24 processor, the default time unit is one half of that provided for by BasicX.

InputCaptureEx

Type	Subroutine		

Invocation   InputCaptureEx(pin, data, count, flags)
InputCaptureEx(pin, data, count, flags, timeout)

Parameter	Method	Type	Description
pin	ByVal	Byte	The input capture pin to use.
data	ByRef	array of UnsignedInteger	The array in which pulse width information will be stored.
count	ByVal	int16	The number of pulse widths to store. This should be no larger than the number of entries in the passed array.
flags	ByVal	Byte	A value of zero requests that a falling edge begin the capture process while a value of 1 indicates a rising edge. All other values are reserved.
timeout	ByVal	integral	If non-zero, this parameter specifies a timeout value that, if exceeded, will terminate the input capture process.

Discussion

This routine collects timing data from a pulse train applied to the specified input capture pin and stores it in the specified array. The stored data reflects the width of the successive high and low portions of the pulse train. If any segment is longer than can be represented in a 16-bit value, the stored value will be 65535 (&Hffff) and the immediately following value, if any, will be meaningless.

Prior to commencing the input capture process all of the elements of the data array are initialized with the value 65534 (&Hfffe). This fact can be used to determine the actual number of timing data stored in the array during input capture.

The stored values represent the number of I/O Timer ticks (i.e. $1/F\_CPU$ or about 67.8ns for 14.7MHz devices) measured for each segment of the pulse train. However, the value of `Register.TimerSpeed1` may be changed to allow longer pulse widths to be measured. See the section on Timers for more information.

Due to the overhead of servicing the input capture interrupt and possible RTC interrupts the shortest interval (high or low segment) that can be reliably measured corresponds to about 300 CPU cycles (about 21µs for 14.7MHz devices). If an input waveform had a 50% duty cycle this would correspond to about 24KHz. Additional interrupt sources may increase the minimum interval that can be measured reliably.

If the optional `timeout` parameter is specified and is non-zero, the Input Capture process will be terminated if N * 65536 I/O Timer ticks occur (where N is the value of the `timeout` parameter) before the specified number of datapoints has been stored. This gives a range of possible timeout values from about 4.5mS to 290 seconds with a resolution of 4.5mS (using the default value of `Register.TimerSpeed1`) for 14.7MHz devices.

The calling task will be suspended until the specified number of datapoints has been stored, the timeout value is exceeded or the task is resumed using ResumeTask(). Other tasks will be allowed to run but you must be careful to not call any routines that may disable interrupts for long periods of time because that could interfere with the accuracy of the input capture timing.

Resource Usage

This routine utilizes a timer to collect the timing information of the pulse train. See the section Input Capture Timers for information on the valid input capture pins and the timer associated with each and the ISRs utilized for native mode devices.

Example

```
Dim pd(1 to 5) as UnsignedInteger

Call PutPin(D.6, zxInputTriState)
Call InputCaptureEx(D.6, pd, UBound(pd), 1)
```

Compatibility

This routine is not available in BasicX compatibility mode.

LBound

Type	Function returning an integral value		

Invocation	LBound(array) or		
	LBound(array, dimension)		

Parameter	Method	Type	Description
array	ByRef	any array	The array about which the bound information is desired.
dimension	ByVal	int16	The dimension of interest. See the description for more details.

Discussion

This function returns the lower bound of a dimension of the specified array. There are two forms. The first requires only the array to be specified. In this case, the lower bound of the first dimension of the array is returned. The second form specifies a dimension number (which must be a constant value), the valid range of which is 1 to the number of dimensions of the array. The array may be located in RAM, Program Memory or Persistent Memory.

Note that the use of this function instead of hard-coding values makes your code easier to maintain because it automatically adapts if the definition of an array changes.

Example

```
Dim ba(1 to 20) as Byte
Dim ma(3 to 5, -6 to 7) as Byte
Dim i as Integer

i = LBound(ba)          ' the result is 1
i = LBound(ma)          ' the result is 3
i = LBound(ma, 1)       ' the result is 3
i = LBound(ma, 2)       ' the result is -6
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   UBound, Span

LCase

Type   Function returning String

Invocation LCase(str)

Parameter	Method	Type	Description
str	ByVal	String	The string to be changed to lower case.

Discussion

This function returns a new string containing the same characters as the passed string except that all upper case characters will be replaced with lower case characters.

Example

```
Dim s as String, s1 as String
s = "Hello, world!"
s2 = LCase(s)            ' the result will be "hello, world!"
```

See Also UCase

Left

Type	Function returning String		

Invocation   Left(str, length)

Parameter	Method	Type	Description
Str	ByVal	String	The string from which to extract characters.
length	ByVal	int8/16	The number of characters to extract from the string.

Discussion

This function returns a string consisting of the leftmost characters of the given string. The maximum number of characters in the returned string is the smaller of 1) the number of characters in the string passed as the first parameter and 2) the value of the second parameter. Internally, the length is interpereted as a 16-bit signed value and negative values are treated as zero.

This function produces the same result as `Mid(str, 1, length)`.

Example

```
Dim s as String, s2 as String

s = "Hello, world!"
s2 = Left(s, 5)          ' the result will be "Hello"
```

See Also   Mid, Right, Trim

Len

Type   Function returning Integer

Invocation   Len(str)

Parameter	Method	Type	Description
str	ByVal	String	The string of which the length is to be determined.

Discussion

This function returns the length of the given string, in bytes. Note that the length may be zero.

Example

```
Dim s as String
Dim i as Integer

s = "Hello, world!"
i = Len(s)        ' the result will be 13
```

LoByte

Type   Function returning Byte

Invocation   LoByte(val)

Parameter	Method	Type	Description
val	ByVal	numeric	The value of which the low byte is desired.

Discussion

This function returns the least significant byte of the specified value.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   HiByte, HiWord, LoWord, MidWord

LockTask

Type　　　　Subroutine

Invocation　　LockTask()

Discussion

This routine causes the running task to become locked so that no other task can run. The one exception to this is a task that is awaiting an external interrupt or an interval interrupt. Note that a task may explicitly unlock itself by calling `UnlockTask()`. A task will also become unlocked if it calls any of the sleep or delay routines.

Note that multiple calls to `LockTask()` have the same effect as a single call to `LockTask()` assuming that no other calls are made that implicitly unlock the task.

Compatibility

The BasicX documentation indicates that a locked task will yield to a task that is awaiting an interrupt when the interrupt occurs. However, testing indicates that this is, in fact, not the case. This implementation allows an interrupt task to have priority over a locked task.

See Also　　　UnlockTask, Delay, Sleep, WaitForInterrupt, WaitForInterval

Log

Type   Function returning Single

Invocation   Log(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value of which the natural log is to be computed.

Discussion

This function returns the `Single` value corresponding to natural logarithm (base e) of the value provided. The transcendental value e, upon which the natural logarithm is based, is approximately 2.71828. This function is the inverse of the `Exp()` function.

If the value of the argument provided is zero, the result is positive infinity. If the argument value is negative, the result is NaN.

See Also   Exp, Exp10, Log10

Log10

Type	Function returning Single		

Invocation	Log10(arg)		

Parameter	Method	Type	Description
arg	ByVal	Single	The value of which the common log is to be computed.

Discussion

This function returns the `Single` value corresponding to the common logarithm (base 10) of the value provided. This function is the inverse of the `Exp10()` function.

If the value of the argument provided is zero, the result is positive infinity. If the argument value is negative, the result is NaN.

See Also   Exp, Exp10, Log

LongJmp

Type   Subroutine

Invocation   LongJmp(jmpbuf, val)

Parameter	Method	Type	Description
jmpbuf	ByRef	Array of Byte	A buffer holding the return context, see description below.
val	ByVal	int16	The value to be returned to the original SetJmp() caller.

Discussion

This subroutine, in conjunction with SetJmp(), provides a way to circumvent the normal call-return structure and return directly to a distant caller. It is the equivalent of a non-local Goto function and can be used, among other purposes, to handle exceptions in your programs. The first parameter specifies a Byte array that has been previously initialized by a call to SetJmp(). The second parameter specifies a value that will be seen by the original SetJmp() caller as the return value. This value, which should be non-zero, can indicate the nature of the condition that led to the LongJmp() call. See the section on Exception Handling in the ZBasic Reference Manual for more details.

Caution

Passing a jump buffer that has not been prepared by a call to SetJmp(), one that has been modified after the SetJmp()call, or one that was prepared by a subroutine/function that is no longer active will have unpredictable and almost certainly undesirable effects.

Compatibility

This routine is not available in BasicX compatibility mode. Also, this routine should not be used in applications that use ZBasic objects because it bypasses the execution of destructors that are necessary for proper object management.

See Also   SetJmp

LoWord

Type	Function returning UnsignedInteger

Invocation   LoWord(val)

Parameter	Method	Type	Description
val	ByVal	numeric	The value of which the low word is desired.

Discussion

This function returns the least significant word of the specified value. If the specified value is a Byte the return value will have zero in the high byte.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   HiByte, HiWord, LoByte, MidWord

MakeDword

Type   Function returning UnsignedLong

Invocation   MakeDword(loWord, hiWord)

Parameter	Method	Type	Description
loWord	ByVal	int16	The value for the low word of the double word value.
hiWord	ByVal	int16	The value for the high word of the double word value.

Discussion

This function returns a value composed of the two word values.

Example

```
Dim w1 as UnsignedInteger, w2 as UnsignedInteger
Dim ul as UnsignedLong

ul = MakeDword(w1, w2)
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   MakeWord

MakeString

Type   Function returning String

Invocation   MakeString(address, length)

Parameter	Method	Type	Description
address	ByVal	integral	The address of bytes with which to populate the string.
length	ByVal	int8/16	The number of characters to place in the string.

Discussion

This function populates a string with an arbitrary byte stream. It is most useful for composing or modifying strings but may have other uses as well.

Example

```
Dim ba(1 to 10) as Byte
Dim i as Integer
Dim s as String

For i = LBound(ba) to UBound(ba)
     ba(i) = &H60 + CByte(i)
Next i
s = MakeString(MemAddress(ba), SizeOf(ba))
```

Compatibility

This function is not available in BasicX compatibility mode.

MakeWord

Type   Function returning UnsignedInteger

Invocation   MakeWord(loByte, hiByte)

Parameter	Method	Type	Description
loByte	ByVal	Byte	The value for the low byte of the word value.
hiByte	ByVal	Byte	The value for the high byte of the word value.

Discussion

This function returns a value composed of the two byte values.

Example

```
Dim b1 as Byte, b2 as Byte
Dim u as UnsignedInteger

u = MakeWord(b1, b2)
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   MakeDword

Max

Type   Function (see discussion for the return type)

Invocation   Max(val1, val2)

Parameter	Method	Type	Description
val1	ByVal	numeric	One of two values of which the largest is desired.
val2	ByVal	numeric	One of two values of which the largest is desired.

Discussion

This function returns the larger of the two supplied values, both of which must be of the same type. If the supplied values are signed, the determination of which is largest takes the sign of the values into account. The return value is the same type as the parameters.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   Min

MemAddress

Type	Function returning Integer		
Invocation	MemAddress(var)		

Parameter	Method	Type	Description
var	ByRef	any variable	The variable of which the address is desired.

Discussion

This function returns the `Integer` representation of the RAM address of the specified variable. Note that for arrays, you may also specify subscript expressions for all of the array dimensions to yield the address of an individual array element. Without the subscript expressions, the resulting value will be the address of the first element of the array.

This function is useful for deriving the address to pass to the several functions that require a RAM address, e.g. `BitCopy()`, `RamPeek()`, `RamPoke()`, etc.

The address of any variable can also be obtained using the `DataAddress` property. For RAM-based variables, the `DataAddress` property is of type `UnsignedInteger`.

Example

```
Dim addr as Integer
Dim ba(1 to 20) as Byte
Dim fval as Single

addr = MemAddress(fval)
addr = MemAddress(ba)
addr = MemAddress(ba(2))
addr = fval.DataAddress
addr = ba.DataAddress
addr = ba.DataAddress(2)
```

Compatibility

BasicX only supports the DataAddress property for Program Memory data items.

See Also   MemAddressU, VarPtr

MemAddressU

Type	Function returning UnsignedInteger
Invocation	MemAddressU(var)

Parameter	Method	Type	Description
var	ByRef	any variable	The variable of which the address is desired.

Discussion

This function returns the `UnsignedInteger` representation of the RAM address of the specified variable. Note that for arrays, you may also specify subscript expressions for all of the array dimensions to yield the address of an individual array element. Without the subscript expressions, the resulting value will be the address of the first element of the array.

This function is useful for deriving the address to pass to the several functions that require a RAM address, e.g. `BitCopy()`, `RamPeek()`, `RamPoke()`, etc.

The `DataAddress` property may also be used to determine the address of a variable (except in BasicX compatibility mode). The type of the resulting value is `UnsignedInteger`. See the examples below.

Examples

```
Dim addr as UnsignedInteger
Dim ba(1 to 20) as Byte
Dim fval as Single

addr = MemAddressU(fval)
addr = MemAddressU(ba)
addr = MemAddressU(ba(2))
addr = ba.DataAddress
addr = ba.DataAddress(2)
```

See Also   MemAddress, VarPtr

MemCmp

Type   Function returning Integer

Invocation   MemCmp(addr1, addr2, count)

Parameter	Method	Type	Description
addr1	ByVal	integral	The address of the first block of memory to be compared.
addr2	ByVal	integral	The address of the second block of memory to be compared.
count	ByVal	integral	The number of bytes to compare.

Discussion

This function can be used to compare two arbitrary sequences of data in RAM. If all of the bytes in the two blocks are the same (over the given number of bytes to compare) the value zero is returned. Otherwise, the return value will be greater than zero if at the position of the first mismatch the byte in the first block is greater than the corresponding byte in the second block. If the converse is true, the return value will be less than zero.

All three parameters are converted internally to `UnsignedInteger`.

Example

```
Dim a1(1 to 10) as Byte
Dim a2(1 to 10) as Byte
Dim ival as Integer

ival = MemCmp(a1.DataAddress, a2.DataAddress, SizeOf(a1))
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   MemCopy, MemSet

MemCopy

Type	Subroutine		
Invocation	MemCopy(destination, source, count)		

Parameter	Method	Type	Description
destination	ByVal	integral	The address to which to begin copying.
source	ByVal	integral	The address from which to begin copying.
count	ByVal	integral	The number of bytes to copy.

Discussion

This subroutine can be used to copy a block of data from one location in RAM to another location. An overlapping copy (when the destination is in the midst of the data being copied) is handled correctly so that the data to be copied is not overwritten.

All three parameters are converted internally to `UnsignedInteger`. Note that `MemCopy()` has the same functionality as `BlockMove()` but has a different parameter order; one that you may be accustomed to.

Caution

This subroutine should be used with care because it is possible to overwrite important data on the stack or other areas of memory which may cause your program to malfunction.

Example

```
Dim ba(1 to 10) as Byte
Dim ival as Integer

ba(3) = &H48
ba(4) = &H55
Call MemCopy(ival.DataAddress, ba(3).DataAddress, SizeOf(ival))
```

After execution, `ival` will have the value &H5548. Note the use of the `SizeOf()` function. This is a better programming practice than using a specific value because it makes the code easier to maintain.

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   BitCopy, MemCmp, MemSet

MemFind

Type	Function returning UnsignedInteger		

Invocation   MemFind(dataAddr, dataLen, val)
MemFind(dataAddr, dataLen, val, ignoreCase)

Parameter	Method	Type	Description
dataAddr	ByVal	integral	The address in RAM of the block to search.
dataLen	ByVal	integral	The length of the block to search.
val	ByVal	Byte	The byte value for which to search.
ignoreCase	ByVal	Boolean	A flag controlling whether alphabetic case is significant.

Discussion

This function attempts to find the first occurrence of the byte specified by the `val` parameter in a block of RAM beginning at the specified address. If it is found, the return value gives the 1-based index where the sought byte was found within the block. If the sought byte is not found, zero is returned. If the optional `ignoreCase` parameter is not given, the search is performed observing alphabetic case differences, otherwise alphabetic case differences are significant or not depending on the value specified for `ignoreCase`. For the purposes of this parameter only the characters A-Z and a-z (&H41 to &H5a and &H61 to &H7a) are considered to be alphabetic.

Example

```
Dim buf(1 to 40) as Byte
Dim idx as UnsignedInteger

' search for a carriage return
idx = MemFind(buf.DataAddress, Ubound(buf), &H0d)
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   ProgMemFind, StrFind

MemSet

Type	Subroutine
Invocation	MemSet(addr, count, val)

Parameter	Method	Type	Description
addr	ByVal	integral	The address of a block to initialize.
count	ByVal	int8/16	The number of bytes to initialize.
val	ByVal	Byte	The initialization value.

Discussion

This routine is useful for initializing arrays, buffers, etc. that reside in RAM.

Example

```
Dim ba(1 to 20) as Byte

Call MemSet(MemAddress(ba), Sizeof(ba), &H55)
Call MemSet(ba.DataAddress, Sizeof(ba), 0)
```

Caution

Using this routine to initialize data other than your own program variables may have detrimental effects.

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   MemCmp, MemCopy

Mid

Type	Function returning String		

Invocation	Mid(str, pos, length) or		
	Mid(str, pos)		

Parameter	Method	Type	Description
str	ByVal*	String	The string from which to extract or modify a substring.
pos	ByVal	int8/16	The position of the first character of the substring.
length	ByVal	int8/16	The length of the substring to extract or modify.

* When used on the left hand side of an assignment, this parameter is passed ByRef.

Discussion

This function can be used to extract a portion of a string or to modify a portion of a string, depending on how it is used. When it appears in the context of a function call, it returns a new string extracted from the string provided. The first character of the extracted substring will be the character at the position given by `pos` (where the first character of the string is position 1). The length of the returned string will be the number of characters in the source string beginning at the starting index through the end of the string or the specified length (if present), whichever is less. If the starting position is beyond the end of the string or if the specified length is less than or equal to zero, the returned string will be of zero length.

When used on the left hand side of an assignment operator, the `Mid()` function replaces a sequence of characters in a string with characters from the string value on the right hand side of the assignment operator.

```
Dim s as String
s = "abcdef"
Mid(str, 3) = "##"        ' result is "ab##ef"
```

Note that when used in this way the first parameter is passed by reference so it cannot be a literal string or any other entity than cannot be passed by reference. Also, the length of the target string will never be changed. The number of characters overwritten in the destination string will be the lesser of a) the number of characters in the string on the right hand side of the assignment, b) the number of characters specified in the third parameter (if present), and c) the number of characters in the target string beginning at the position specified by the second parameter through the end of the string.

Compatibility

In BasicX, the first parameter is pass-by-reference. This disallows any use of a string literal for the first parameter. Also, in BasicX the third parameter must always be provided.

The BasicX documentation suggests that using `Mid()` on the left hand side of an assignment might result in a change in the string length. Tests indicate that this is not the case. Moreover, execution of the code fragment below actually results in a garbage character being placed in the third character position.

```
Dim s as String
s = "abc"
Mid(s, 2, 2) = "!"        ' result is "a!@" (@ is an indeterminate character)
```

See Also   Left, Right, Trim

MidWord

Type   Function returning UnsignedInteger

Invocation   MidWord(val)

Parameter	Method	Type	Description
val	ByVal	numeric	The value of which the middle word is desired.

Discussion

This function returns the middle two bytes of a 4-byte value. If the specified value is a Byte the return value will be zero. If the specified value is contained in two bytes, the return value will have zero in the high byte.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   HiByte, HiWord, LoByte, LoWord

Min

Type   Function (see discussion for the return type)

Invocation   Min(val1, val2)

Parameter	Method	Type	Description
val1	ByVal	numeric	One of two values of which the smallest is desired.
val2	ByVal	numeric	One of two values of which the smallest is desired.

Discussion

This function returns the smaller of the two supplied values, both of which must be of the same type. If the supplied values are signed, the determination of which is smallest takes the sign of the values into account. The return value is the same type as the parameters.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   Max

NoOp

Type   Subroutine

Invocation NoOp()

Discussion

This subroutine implements a delay of one CPU cycle, typically about 68nS.

Compatibility

This function is only available for native code targets, e.g. the ZX-24n.

OpenCom

Type   Subroutine

Invocation   OpenCom(channel, baud, inQueue, outQueue)

Parameter	Method	Type	Description
channel	ByVal	Byte	The serial channel to open.
baud	ByVal	Long	The desired baud rate.
inQueue	ByRef	array of Byte	The queue for incoming characters.
outQueue	ByRef	array of Byte	The queue for outgoing characters.

Discussion

This subroutine prepares a serial channel for use. If the specified channel number is invalid, the call has no effect. Serial channels are either implemented in hardware (using an onboard UART) or in software. Depending on the device one, two or four hardware-based serial channels are supported, denoted by the channel numbers 1, 2, 7, 8, etc. (Com1, Com2, Com7, Com8, etc., respectively). All ZBasic devices can support as many as four software-based serial channels, denoted by the channel numbers 3-6 (Com3, Com4, Com5 and Com6). Note, however, that you must have previously called `ComChannels()` in order to use channels 4-6.

The supported baud rates for the hardware-based channels are the standard rates from 300 to 460,800. Tthe supported baud rates for software-based channels are listed in the table below. However, if `ComChannels()` has been invoked, the maximum rate for channels 3-6 will be limited to that specified in the description of `ComChannels()`. Moreover, for channels 3-6 the baud rate for any given channel must be an integral divisor of the maximum rate. Also, for ZX devices running at 7.37MHz, the maximum software-based channel baud rate is 9600. For generic target devices, the set of desired software-based channel baud rates must be explicitly specified as part of the device configuration and will be a subset of the rates in the table below; the baud rates that are attainable with a specified accuracy are dependent on the operating frequency.

Supported Baud Rates for Channels 3-6

300	600	1200	2400	4800	9600	19200

The transmit and receive queues specified for the channel each must have been previously initialized by calling `OpenQueue()`. If you set up a transmit-only or receive-only serial channel you may use the value 0 for the unused queue. If you provide the value 0 for both queues, the channel will not be opened.

After opening the channel, flow control may be configured for either the transmit side, the receive side or both. See the description of the ControlCom() subroutine for more information.

Example

```
Dim outQueue(1 to 40) as Byte

Call OpenQueue(outQueue, SizeOf(outQueue))
Call ComChannels(2, 9600)
Call DefineCom(4, 0, 12, &H08)
Call OpenCom(4, 9600, 0, outQueue)
```

The code above prepares Com4 as a transmit-only serial channel. If you wanted reception as well, you would have to declare and initialize a second queue and define the receive pin.

Resource Usage

See the resource usage sub-section UARTs for information on which UART is assigned to each available serial channel, the transmit and receive pins, and the ISRs utilized for native mode devices.

The software-based serial channels are implemented using the Serial Timer. See the resource usage sub-section Timers for information on which timer is used for a particular target device

Compatibility

In BasicX, the supported channel numbers are 1 to 3, depending on the particular target chip. Also, BasicX doesn't support the use of zero to indicate that no queue is being supplied.

See Also   ComChannels, CloseCom, ControlCom, DefineCom, StatusCom

OpenDAC

Type   Subroutine

Invocation   OpenDAC(channel, mode)
                OpenDAC(channel, mode, stat)

Parameter	Method	Type	Description
channel	ByVal	Byte	The channel to use for DAC generation.
mode	ByVal	integral	The desired DAC mode (see discussion below).
stat	ByRef	Boolean	The variable to receive the status code.

Discussion

This subroutine prepares a DAC channel for generating an analog voltage level. The pins on which an analog level may be generated depends on the target device. See the Resource Usage sub-section Digital-to-Analog Converters for details on the available analog output pins.

DAC Mode Constituent Values

Function	Hex Value	Bit Mask
Dual Output	&H8000	1x xx xx xx xx xx xx xx
Single Output	&H0000	0x xx xx xx xx xx xx xx
Automatic Refresh	&H4000	x1 xx xx xx xx xx xx xx
Manual Refresh	&H0000	x0 xx xx xx xx xx xx xx
Internal 1-volt Reference	&H0000	xx xx xx xx xx xx xx 00
AVcc Reference	&H0001	xx xx xx xx xx xx xx 01
PortA Aref Reference	&H0002	xx xx xx xx xx xx xx 10
PortB Aref Reference	&H0003	xx xx xx xx xx xx xx 11

It is important to note that the `mode` parameter value is only used for the first OpenDAC() call for each channel pair. That is to say, if one channel of a pair is already open when OpenDAC() is called, the `mode` parameter is ignored.

The shaded portion of the table above applies only to non-USB xmega devices. For these devices, to used dual output mode, the DAC values must be updated at least every 30uS. This will be done automatically if the Automatic Refresh bit is set in the `mode` parameter. Otherwise, your application will need to ensure that the DAC values are updated frequently enough to prevent drooping of the DAC output.

The analog value output by the DAC will be approximately equal to the 12-bit digital value set for each channel (see the DAC() subroutine) divided by 4095 and multiplied by the reference voltage. The choice of four reference voltages available is made by the least significant two bits of the `mode` parameter value. For the PortA and PortB Aref Reference, the table below indicates the pin to which the desired reference voltage should be applied.

DAC Reference Voltage Pins

ZBasic Target	Aref A	Aref B
ZX-24x, ZX-24u	20, A.0	7, B.0
ZX-32a4	40, A.0	4. B.0
ZX-128a1	95, A.0	5, B.0
ZX-24xu	36, A.0	28, B.0
xmegaA1, xmegaA1U	95, A.0	5, B.0
xmegaA3, xmegaA3U, xmegaA3B, xmegaA3BU, xmegaD3	62, A.0	6. B.0
xmegaA4, xmegaA4U	40, A.0	4. B.0
xmegaD4	40, A.0	-

The `status` parameter, if supplied, receives a value to indicate success or failure of the call.

Example

```
Call OpenDAC(1, &H01) ' prepare for DAC output using AVcc reference
Call DAC(1, 300)      ' set the DAC level to 300/4095*AVcc
```

Compatibility

This subroutine is only available for xmega target devices and is not available in BasicX compatibility mode.

See Also   CloseDAC, DAC

OpenI2C

Type   Subroutine

Invocation   OpenI2C(channel, sdaPin, sclPin) or
               OpenI2C(channel, sdaPin, sclPin, bitRate)

Parameter	Method	Type	Description
channel	ByVal	Byte	The I2C channel to open (0-4).
sdaPin	ByVal	Byte	The pin for the I2C data (SDA) signal.
sclPin	ByVal	Byte	The pin for the I2C clock (SCL) signal.
bitRate	ByVal	integral	The optional clock speed designation, see discussion.

Discussion

This subroutine prepares an I2C channel for use. Five channels are supported, numbered 0 through 4. Channel zero uses the onboard hardware I2C controller (if available) while channels 1 through 4 are generally implemented in software. However, on devices with multiple I2C controllers (e.g. xmega-based devices) channels 1 to 4 can be used for the additional hardware I2C controllers by specifying the SCL and SDA pins as zero. The I2C implementation does not support multi-master arbitration when operating in Master mode. Slave clock stretching is supported on both hardware and software channels.

For channel 0, the `sdaPin` and `sclPin` parameters are ignored since the hardware uses specific pins for the SDA and SCL signals (e.g. Port C, bits 1 and 0, respectively). For channels 1-4 in software mode, the `sdaPin and sclPin` parameters specify the pins to use for the data and clock signals, respectively. In both cases, the clock and data pins are automatically configured for I2C operation. The I2C protocol requires pullup resistors on both of the lines, the value of which depends on characteristics of your system. A typical value is in the range of 1.5K to 4.7K.

The optional `bitRate` parameter allows you to control the speed of the data interchange. If the parameter is not given, the default speed is 100KHz. Each I2C device has a maximum clock rate at which it will operate reliably; check the datasheet of your selected device to determine the maximum rate.

The interpretation of the value of the `bitRate` parameter differs for channel 0 and channels 1-4. The tables below specify the values to use for several common clock speeds.

I2C Hardware Channel Clock Speeds

bitRate Value	Approximate Clock Speed	Notes
140	50KHz	
66	100KHz	Standard Low Speed, default speed
29	200KHz	
11	388KHz	Closest to Standard High Speed (400KHz)
10	410KHz	Highest supported speed

I2C Software Channel Clock Speeds[1]

bitRate Value	Approximate Clock Speed	Notes
295	50KHz	
148	100KHz	Standard Low Speed, default speed
74	200KHz	
59	250KHz	Highest supported speed

[1] The values given assume the default setting of `Register.TimerSpeed1`.

For hardware channels, the `bitRate` parameter controls the hardware bit rate. For ATmega-based devices, the parameter is a composite of two values: the value in the lower 8 bits is known as BR and is

written to the TWBR register of the I2C controller. The low two bits of the high byte select a clock divisor according to the table below. The clock speed of the hardware channel is given by the formula F_CPU / (16 + 2 * BR * Divisor) where F_CPU is the device's operating frequency. If the `bitRate` parameter is omitted or is zero the value of 66 is used by default.

Channel 0 Prescaler Selector Value

Value	Divisor
0	1
1	4
2	16
3	64

For ATxmega-based devices, the I2C bit rate is given by the formula `F_CPU / 2 / (5 + rateVal)` where `rateVal` is the low 8 bits of the `bitRate` parameter. Rearranging this formula gives an equation for the `bitRate` parameter: `bitRate = (F_CPU / 2 / F_I2C) – 5` where `F_I2C` is the desired I2C clock frequency.

For software channels the `bitRate` parameter is interpreted as the number of I/O Timer ticks per bit. For I2C operations, the I/O Timer uses the prescaler specified by `Register.TimerSpeed1`. With the default prescaler of 1, each I/O Timer tick represents approximately 68nS with a main clock frequency of 14.7MHz. If the `bitRate` parameter is omitted or is zero the value of 74 is used by default. Due to processing overhead, the minimum attainable bit time is approximately 60 CPU cycles (4µS at 14.7MHz).

See the Resource Usage subsection "I2C Controllers" for information on the available I2C hardware channels and the corresponding clock and data pins.

Examples

```
Call OpenI2C(0, 0, 0)                ' open the hardware channel at 100KHz
Call OpenI2C(2, 19, 20)              ' open channel 2 using pins 19, 20
Call OpenI2C(1, C.3, A.1, 74)        ' open channel 1 at 200KHz
```


Resource Usage

The I2C routines utilize the I/O Timer to regulate the bit timing for the software channels. While sending or receiving I2C data, the corresponding timer busy flag will be True indicating that the I/O Timer is in use.

Compatibility

This subroutine is not available in BasicX compatibility mode.

See Also   CloseI2C, I2CGetByte, I2CPutByte, I2CStart, I2CStop, I2CCmd, OpenI2CSlave

OpenI2CSlave

Type   Subroutine

Invocation OpenI2CSlave(slaveAddr)
                OpenI2CSlave(slaveAddr, channel)

Parameter	Method	Type	Description
slaveAddr	ByVal	Byte	The I2C slave address to which to respond.
channel	ByVal	Byte	The I2C channel to open (0-4).

Discussion

This subroutine immediately activates the I2C controller in slave mode. If the optional `channel` parameter is not given, channel 0 is assumed.   Note that use of channels 1-4 are supported only on devices that have multiple I2C controllers (e.g. ATxmega devices). See the description of `OpenI2C` for more details about the correspondence between channel numbers and hardware controllers.

If you activate slave mode, you must also provide an interrupt handler for the `TWI` vector (aka the `I2C` vector). While slave mode is active, calls to CmdI2C() and the low level I2C commands are ineffective for the I2C channel in use. Slave mode can be canceled by calling `CloseI2C()`, specifying the channel number specified or implied in the call to `OpenI2Cslave()`.

While slave mode is active, the device will respond to reads and writes on the I2C bus referring to its slave address which is the value of the `slaveAddr` parameter with the least significant bit set to zero. If the least significant bit of the `slaveAddr` parameter is set, the slave can respond also to "general call" traffic on the bus.

See the Resource Usage subsection I2C Controllers for information on the available I2C hardware channels and the corresponding clock and data pins.

Example

```
Call OpenI2CSlave(&H50) ' activate I2C slave mode with address &H50
```

Resource Usage

The I2C hardware channel in use cannot be opened by OpenI2C() while slave mode is active. On the ZX-24n, ZX-24s, and ZX-24t, I2C slave mode cannot be used while Com2 is open since pin 11 is shared by the SDA signal and TxD for Com2.

Compatibility

This subroutine is only available for native mode devices.

See Also   CloseI2C, OpenI2C

OpenPWM

Type	Subroutine

Invocation   OpenPWM(channel, frequency, mode)
                    OpenPWM(channel, frequency, mode, stat)

Parameter	Method	Type	Description
channel	ByVal	Byte	The channel to use for PWM generation.
frequency	ByVal	Single	The desired PWM frequency.
mode	ByVal	Byte	The desired PWM mode (see discussion below).
stat	ByRef	Boolean	The variable to receive the status code.

Discussion

This subroutine prepares a PWM channel for generating a pulse width modulated (PWM) signal. PWM generation is performed using one of the CPU's 16-bit timers, the number of which varies depending on the ZBasic device. See the Resource Usage sub-section 16-Bit PWM Timers for details of the available channels and the corresponding timer and output pin used. See the description of PWM() for additional details on the PWM channels.

The `frequency` parameter specifies the PWM base frequency in Hertz. Since the same frequency and generation mode will be used for all PWM channels based on the same timer, it is only necessary to call `OpenPWM()` once to prepare the timer for all of the PWM channels that are based on a given timer.

The `mode` parameter specifies the PWM generation mode. Two modes are supported: Fast PWM mode and Phase/Frequency Correct mode. The constants `zxFastPWM` and `zxCorrectPWM`, having the values 0 and 1 respectively, may be used to specify the mode. The Fast PWM mode has a maximum frequency of one-half of the CPU clock frequency and is intended for fixed-frequency applications. The Phase/Frequency Correct PWM mode has a maximum frequency of one-quarter of the CPU clock frequency and may be used when the PWM frequency will be changed in the midst of PWM signal generation. Frequency changes are effected by making additional calls to `OpenPWM()` and the change is synchronized so that it takes effect at the beginning of a cycle.

The `status` parameter, if supplied, receives a value to indicate success or failure of the call.

A side effect of calling `OpenPWM()` is that the timer busy flag for the underlying timer (e.g. `Register.Timer1Busy`) will be set to `True` irrespective of its prior state. It is recommended that the initial call to `OpenPWM()` be preceded by a call to acquire the semaphore for the timer. This will ensure that an existing timer operation will not be disturbed.

It is important to note that the call to OpenPWM() doesn't affect the configuration the PWM output pin. When a call is eventually made to PWM(), the PWM pin will be made an output and be actively driven. If your application needs to have the PWM pin in a particular state prior to PWM beginning you must configure the pin in your code.

Example

```
Call OpenPWM(1, 50.0, zxFastPWM)  'prepare for 50Hz Fast PWM using channel 1
```

Compatibility

This subroutine is not available in BasicX compatibility mode.

See Also   ClosePWM, PWM

OpenPWM8

Type   Subroutine

Invocation   OpenPWM8(channel, frequency, mode)
                  OpenPWM8(channel, frequency, mode, stat)

Parameter	Method	Type	Description
channel	ByVal	Byte	The channel to use for PWM generation.
frequency	ByVal	Single	The desired PWM frequency.
mode	ByVal	Byte	The desired PWM mode (see discussion below).
stat	ByRef	Boolean	The variable to receive the status code.

Discussion

This subroutine prepares a PWM channel for generating a pulse width modulated (PWM) signal using one of the CPU's 8-bit timers. The table below indicates the available channels and the corresponding timer used. See the Resource Usage sub-section 8-Bit PWM Timers for details of the available channels and the corresponding timer and output pin used. See the description of PWM8() for additional details on the PWM channels. Note that ZBasic devices based on ATxmega processors don't have any 8-bit timers so 8-bit PWM is not supported on those devices.

It is important to note that the timer used for 8-bit PWM generation is the same one used for generating the timing for the software UARTs (Com3-Com6). Consequently, these two features cannot be used at the same time.

The `frequency` parameter specifies the desired PWM base frequency in Hertz. Since the same frequency and generation mode will be used for all PWM channels based on the same timer, it is only necessary to call `OpenPWM8()` once to prepare the timer for all of the PWM channels that are based on a that timer.

The `mode` parameter specifies the PWM generation mode. Two modes are supported: Fast PWM mode and Phase/Frequency Correct mode. The constants `zxFastPWM` and `zxCorrectPWM`, having the values 0 and 1 respectively, may be used to specify the mode

The `status` parameter, if supplied, receives a value to indicate success or failure of the call.

A side effect of calling `OpenPWM8()` is that the timer busy flag for the underlying timer (e.g. `Register.Timer2Busy`) will be set to `True` irrespective of its prior state. It is recommended that the initial call to `OpenPWM()` be preceded by a call to acquire the semaphore for the timer. This will ensure that an existing timer operation will not be disturbed.

It is important to note that the call to OpenPWM8() doesn't affect the configuration the PWM output pin. When a call is eventually made to PWM8(), the PWM pin will be made an output and be actively driven. If your application needs to have the PWM pin in a particular state prior to PWM beginning you must configure the pin in your code.

The actual PWM frequency used will be the closest of the available frequencies as shown in the table below for ZBasic devices operating at 14.7MHz. For ZBasic devices operating at a different frequency, the available PWM frequencies will be proportionally higher or lower and can be computed by the formulae given in the table headings

Available 8-bit PWM Frequencies at 14.7MHz

Prescaler Divisor	Fast PWM Frequency F_CPU / Div / 256	Phase Correct PWM Frequency F_CPU / Div / 510
1	57,600.0	28,912.9
2	28,800.0	14,156.8
4	1.440	7,228.2
8	7,200.0	3,614.1
16	3.600.0	1,807.1
32	1,800.0	903.5
64	900.0	451.8
128	450.0	225.9
256	225.0	112.9
1024	56.3	28.2

In the table above, the frequencies in the shaded rows are not available on some ZBasic devices due to the set of available prescaler divisors on the 8-bit PWM timer. The table below gives the set of prescaler divisors for each target device.

Available Prescaler Divisors for the 8-bit PWM Timer

Target Devices	Prescaler Divisors
tiny48, tiny88, tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A, tiny441, tiny841	1, 8, 64, 256, 1024
tiny87, tiny167, tiny1634	1, 8, 32, 64, 128, 256, 1024
tiny2313, tiny2313A, tiny4313, tiny828	1, 8, 64, 256, 1024
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	1, 8, 32, 64, 128, 256, 1024
mega16, mega16A, mega32, mega32A , mega644, mega644A , mega164A, mega164P, mega164PA, mega324P, mega324PA, mega644P, mega644PA, mega1284P	1, 8, 32, 64, 128, 256, 1024
mega8515, mega64, mega64A, mega128, mega128A, AT90CAN32, AT90CAN64, AT90CAN128	1, 8, 64, 256, 1024
mega1281, mega2561, mega640, mega1280, mega2560	1, 8, 64, 256, 1024
mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162, AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	1, 8, 64, 256, 1024
mega16U4, mega32U4	1, 2, 4, 8, 16, 32, 64, 128, 256, 1024
mega8535, mega161, mega162, mega163, mega323	1, 8, 32, 64, 128, 256, 1024
mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P	1, 8, 32, 64, 128, 256, 1024
mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	1, 8, 32, 64, 128, 256, 1024
all xmega	n/a

Example

```
Call OpenPWM8(1, 50.0, zxFastPWM)   'prepare for 50Hz Fast PWM using channel 1
```

Compatibility

This subroutine is not available in BasicX compatibility mode nor is it available on ATxmega-based ZBasic devices.

See Also   ClosePWM8, PWM8

OpenQueue

Type	Subroutine		

Invocation   OpenQueue(queue, size)
                          OpenQueue(queue)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue to be initialized.
size	ByVal	int16	The size of the array, in bytes.

Discussion

This routine prepares a queue for use by initializing the management information contained in the queue data structure. The number of bytes of space available for data in a queue is the specified size less the queue management overhead (9 bytes). It may be convenient to use the built-in constant `System.MinQueueSize` in the definition of an array intended to hold a queue.

If the compiler can deduce the size of the array element, e.g. an explicitly dimensioned Byte array is specified, the second parameter may be omitted. In this case, the compiler utilizes the size of the array as the size parameter. Otherwise, the compiler will issue an error message indicating that the size must be explicitly specified.

Caution

If you specify a size parameter that is larger than the actual size of the array, data following the array may be overwritten, usually with undesirable consequences. For this reason, it is recommended that you use the `SizeOf()` function to specify the queue size so that it will automatically track any changes that you make to the actual queue size. See the example below.

`OpenQueue()` should only be called for a queue that is not in use. Invoking it for a queue that is in use has undefined results.

Example

```
Dim inQueue(1 to System.MinQueueSize + 20) as Byte

Call OpenQueue(inQueue, SizeOf(inQueue))
```

After the call to OpenQueue() the queue will ready to be used.

Compatibility

BasicX allows any type for the first parameter. The ZBasic implementation requires that it be an array of `Byte`. The second parameter must always be supplied in BasicX mode.

OpenSPI

Type   Subroutine

Invocation   OpenSPI(channel, flags, csPin)
                    OpenSPI(channel, flags, csPin, rxDelay)

Parameter	Method	Type	Description
channel	ByVal	Byte	The SPI channel to open (1-4).
flags	ByVal	integral	Flags controlling the SPI communication.
csPin	ByVal	Byte	The pin for the chip select signal to the device.
rxDelay	ByVal	Byte	The delay time prior to each received byte.

Discussion

This subroutine prepares an SPI channel for use as a master. By default, the hardware SPI controller is used to implement the SPI protocol but on some devices, a bit-bang implementation can be enabled (see DefineSPI).

Four channels are supported, numbered 1 through 4. It does not matter if the particular channel has been previously opened. The `flags` parameter specifies the characteristics of the SPI communication as shown in the table below. They must be set to be compatible with the device with which you want to communicate. See the table below for details. The `csPin` parameter specifies the pin number that you wish to control the device's chip select input. The pin will be made an output and set to the inactive state (as specified by bit 6 of the `flags` parameter). Any general purpose I/O pin of the device may be used as the slave select pin. Note, however, that if the pin dedicated SS pin is not used to select remote slaves, it must be configured as an output or remain at the logic high state for the duration of an SPI operation.   Also, for ZX devices that use an external SPI EEPROM for program storage (e.g. ZX-24, ZX-24a), the SS pin is used to select the SPI EEPROM and therefore cannot be used for any other purpose.

The table below describes the function of the bits of the *flags* parameter. The shaded entries do not apply to the software SPI implementation.

SPI Channel Control Bits

Function	Hex Value	Binary Value
Bit Rate f/4	&H00	xx xx xx 00
Bit Rate f/16	&H01	xx xx xx 01
Bit Rate f/64	&H02	xx xx xx 10
Bit Rate f/128	&H03	xx xx xx 11
Clock Phase False	&H00	xx xx x0 xx
Clock Phase True	&H04	xx xx x1 xx
Clock Low at Idle	&H00	xx xx 0x xx
Clock High at Idle	&H08	xx xx 1x xx
Use Hardware SPI	&H00	xx x0 xx xx
Use Software SPI	&H10	xx x1 xx xx
Bit Order – MSB first	&H00	xx 0x xx xx
Bit Order – LSB first	&H20	xx 1x xx xx
Active Low Chip Select	&H00	x0 xx xx xx
Active High Chip Select	&H40	x1 xx xx xx
Double Speed	&H80	1x xx xx xx

The remaining bits are currently undefined but they may be employed in the future. Because of this possibility, the undefined flag bits should be zero. Bits 3 and 2 taken together specify the SPI mode 0-3, e.g. xx xx 00 xx specifies mode 0. When using the hardware SPI controller, if the Double Speed bit is set, the SPI channel will run at twice the frequency specified by the two low order flag bits. The value of `f` for

the bit rate selector is the CPU frequency (F_CPU, typically 14.7456MHz for ZX devices). For the software SPI implementation, the number of cycles per bit is a minimum of about 50 so the implementation runs at full speed with either the f/4 or f/16 speed settings.

For devices that have multiple SPI controllers (e.g. xmega-based devices), the most significant byte of the `flags` parameter specifies the index of the SPI controller to use (0=PortD, 1=PortC, 2=PortE, 3=PortF). See the Resource Usage sub-section SPI Controllers for information about the available hardware SPI controllers for the various ZBasic devices and the control and data pins for each.

The `rxDelay` parameter, which defaults to zero if not present, specifies the amount of time to delay before beginning the SPI cycle for each byte received, if any, during the second half of the `SPICmd()` process. See the description of `SPICmd()` for more details.

Caution

For ZX devices that use an external SPI EEPROM for user program storage, you must avoid doing anything that will interfere with the SPI commands to that device. SPI communication by direct manipulation of the processor SPI control registers is not supported and may cause your program to malfunction.

Compatibility

BasicX does not support the double speed option, the active high chip select, the optional rxDelay parameter, or the bit-bang mode. The same is true for ZX devices based on the ATmega32 processor.

See Also   CloseSPI, DefineSPI, OpenSPISlave, SPICmd, SPIGetByte, SPIPutByte, SPIGetData, SPIPutData, SPIStart, SPIStop

OpenSPISlave

Type   Subroutine

Invocation   OpenSPISlave(flags)

Parameter	Method	Type	Description
flags	ByVal	integral	Flags controlling the SPI communication.

Discussion

This subroutine, available only for native mode devices, immediately activates the hardware SPI controller in slave mode. The `flags` parameter specifies the characteristics of the SPI communication. They must be set to be compatible with the SPI master with which you want to communicate. See the table below for details.

<div align="center">

SPI Slave Mode Configuration Bits

Function	Hex Value	Bit Mask
Clock Phase False	&H00	xx xx x0 xx
Clock Phase True	&H04	xx xx x1 xx
Clock Low at Idle	&H00	xx xx 0x xx
Clock High at Idle	&H08	xx xx 1x xx
Bit Order – MSB first	&H00	xx 0x xx xx
Bit Order – LSB first	&H20	xx 1x xx xx

</div>

For devices that have multiple SPI controllers (e.g. xmega-based devices), the most significant byte of the `flags` parameter specifies the index of the SPI controller to use (0=PortD, 1=PortC, 2=PortE, 3=PortF). See the tables below for information about which pins of each port are used for the SPI control/data pins.

The chip select pin for an SPI slave is a dedicated pin. See the Resource Usage sub-section SPI Controllers for information about the available hardware SPI controllers for the various ZBasic devices and the chip select, control and data pins for each. If you activate slave mode, you must also provide an interrupt handler for the corresponding interrupt vector. While slave mode is active, `SPICmd()` calls are ineffective for that channel. Slave mode can be canceled by calling `CloseSPI()`.

See OpenSPI() for information about which pins are used for the data and control signals for each SPI controller.

Note that the SPI master sets the SPI clock speed. The highest SPI clock speed that can be used reliably is one quarter of the CPU clock speed of a ZX slave device. Depending on how much computation the slave must perform to prepare data for sending back to the master, a substantially slower SPI clock may need to be used. If a ZBasic device is being used as the master, it may be useful to set the `rxDelay` parameter on calls to `OpenSPI()` on the master to allow additional processing time.

Compatibility

This subroutine is only supported for native mode devices.

See Also   CloseSPI, OpenSPI

OpenWatchDog

Type   Subroutine

Invocation   OpenWatchDog(timeout)

Parameter	Method	Type	Description
timeout	ByVal	Byte	Specifies a timeout value (see discussion).

Discussion

This subroutine prepares the watchdog timer for use. Once it is opened, the `WatchDog()` routine must be called from time to time. If the period between `WatchDog()` calls exceeds the timeout value, the system will be reset.

The approximate timeout value is T times 2 to the power N where T is the Timeout Base value and N is the value of the `timeout` parameter limited to the range shown in the table below. Note that the timeout value varies with processor voltage, being slightly longer at a lower operating voltage. Consult the Atmel documentation for more specific information.

WatchDog Timeout Parameter Range For ZX Devices

ZX Devices	Timeout Base	Value Range	Max. Time
ZX-24, ZX-40, ZX-44	16mS	0-7	2 sec
ZX-24a, ZX-40a, ZX-44a, ZX-24p, ZX-40p, ZX-44p, ZX-24n, ZX-40n, ZX-44n, ZX-24r, ZX-40r, ZX-44r, ZX-24s, ZX-24t, ZX-40s, ZX-44s, ZX-40t, ZX-44t	16mS	0-9	8 sec
ZX-24x, ZX-24u, ZX-32a4, ZX-128a1, ZX-24xu	8mS	0-10	8 sec
ZX-328n, ZX-328l, ZX-32n, ZX-32l, ZX-1281, ZX-1281n, ZX-1280, ZX-1280n	16mS	0-9	8 sec
ZX-24e, ZX-128e, ZX-128ne	16mS	0-7	2 sec
ZX-24ae, ZX-24ne, ZX-24pe, ZX-24nu, ZX-24pu, ZX-24ru, ZX-24su, ZX-1281e, ZX-1281ne, ZX-328nu	16mS	0-9	8 sec

WatchDog Timeout Parameter Range For Generic Devices

Target Device	Timeout Base	Value Range	Max. Time
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A, tiny441, tiny841, tiny48, tiny88, tiny87, tiny167, tiny2313, tiny2313A, tiny4313, tiny828, tiny1634	16mS	0-9	8 sec
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB, mega164A, mega164P, mega164PA, mega324P, mega324PA, mega644, mega644A, mega644P, mega644PA, mega1284P	16mS	0-9	8 sec
mega8, mega8A, mega16, mega16A, mega32, mega32A, mega8515, mega8535, mega162	16mS	0-7	2 sec
mega64, mega64A, mega128, mega128A	14mS	0-7	2 sec

mega161, mega163, mega323, mega165, mega165A, mega165P, mega165PA	15mS	0-7	2 sec
mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P, mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P	16mS	0-7	2 sec
mega1281, mega2561, mega640, mega1280, mega2560	16mS	0-9	8 sec
mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162	32mS	0-9	16 sec
mega16U4, mega32U4, AT90CAN32, AT90CAN64, AT90CAN128, AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	16mS	0-7	2 sec
all xmega	8mS	0-10	8 sec

When the processor is reset, the register value `Register.ResetFlags` contains bit flags indicating the source of the reset. It is important to note that the occurrence of a system fault (e.g. a stack overflow) will also cause a WatchDog reset as will calling `ResetProcessor()`. See the section on Run Time Stack Checking in the ZBasic Reference Manual for more information on stack overflow detection.

The watchdog timer can be turned off using `CloseWatchDog`.

Compatibility

BasicX doesn't support `Register.ResetFlags` or `CloseWatchDog`.

See Also   WatchDog, CloseWatchDog, ResetProcessor

OpenX10

Type   Subroutine

Invocation   OpenX10(channel, inQueue, outQueue)

Parameter	Method	Type	Description
channel	ByVal	Byte	The X-10 communication channel to open.
inQueue	ByRef	array of Byte	The queue for incoming X-10 data.
outQueue	ByRef	array of Byte	The queue for outgoing X-10 data.

Discussion

This subroutine prepares an X-10 communication channel for use. After the channel is opened you can send arbitrary X-10 command bit streams, which you must create in low-level form, by simply adding the constitutent bytes to the outgoing queue. Similarly, the incoming queue will receive raw X-10 data which you must decode. Each X-10 command begins with the bit sequence 1110 which is followed by additional bit pairs. The bit pair 01 represents a logic zero while the bit pair 10 represents a logic one. The bit pair 11 is invalid and the bit pair 00 signifies the end of a command bit stream and also represents the idle condition. Additional information on X-10 commands may be found in various places on the Internet.

If the specified channel is already open or if the channel number is invalid, the call has no effect. The supported channel numbers are 1-2. The channel must have been previously configured by a call to `DefineX10()`. Also, the queues specified for the receive and transmit channels each must have been previously initialized by calling `OpenQueue()`. If you set up a transmit-only or receive-only serial channel you may use the value 0 for the unused queue. If you provide the value 0 for both queues, the channel will not be opened.

For three phase mode, additional queues may be specified for the transmit and receive operations on phases 2 and 3 using SetQueueX10().

Example

```
Dim outQueue(1 to 40) as Byte

Call OpenQueue(outQueue, SizeOf(outQueue))
Call DefineX10(1, 0, 12, &H08)
Call OpenX10(1, 0, outQueue)
```

The code above prepares channel 1 for transmit-only operation. If you wanted reception as well, you would have to declare and initialize a second queue and define the receive pin.

Resource Usage

X-10 communication requires the use of a zero-crossing signal input to the ZX as noted in the table below. When one or more of the X-10 channels are open the zero-crossing input pin may not be used for any other purpose. When all X-10 channels are closed, zero-crossing input pin will again be available for other uses. Note, however, that the ability to await an external interrupt (e.g. INT0) on the zero-crossing pin is unavailable when the low-level X-10 functionality is included in an application, even if the X-10 channel is closed.

For devices based on the ATtiny and ATmega chips, the default zero-crossing interrupt is INT0. If desired, an alternate interrupt may be specified using the Option X10Interrupt directive described in the ZBasic Language Reference Manual.

For native mode devices, the ISRs noted in the table below are automatically included. The notation `INT#` in that table indicates the default or specified zero-crossing interrupt input, e.g. INT0, INT1, etc. The timing for the X-10 signaling is derived from the RTC timer using a second output compare register (OCRnB). Target devices that do not have a second compare register consequently do not support the low-level X-10 functionality. Further, an application must include the RTC functionality (present by default in ZX devices, not so in generic target devices) in order to support the low-level X-10 functionality.

Resources Required for Low-level X-10 Functionality

Target Device	Zero-Crossing Input	ISRs Included
tiny87, tiny167	n/a	n/a
tiny24, tiny24A, tiny44, tiny44A, tiny84, tiny84A, tiny441, tiny841, tiny48, tiny88, tiny2313, tiny2313A, tiny4313, tiny828, tiny1634	INT#	Timer0_CompB, INT#
mega8, mega8A, mega16, mega16A, mega32, mega32A, mega64, mega64A, mega128, mega128A, mega8515, mega8535, mega161, mega162, mega163, mega323, mega165, mega165, mega165A, mega165P, mega165PA, mega325, mega325P, mega645, mega645A, mega645P, mega169, mega169A, mega169P, mega169PA, mega329, mega329P, mega329PA, mega649, mega649A, mega649P, mega3250, mega3250P, mega6450, mega6450A, mega6450P, mega3290, mega3290P, mega6490, mega6490A, mega6490P, AT90CAN32, AT90CAN64, AT90CAN128	n/a	n/a
mega48, mega48A, mega48P, mega48PA, mega48PB, mega88, mega88A, mega88P, mega88PA, mega88PB, mega168, mega168A, mega168P, mega168PA, mega168PB, mega328, mega328P, mega328PB	INT#	Timer0_CompB, INT#
mega164A, mega164P, mega164PA, mega324P, mega324PA, mega644, mega644A, mega644P, mega644PA, mega1284P	INT#	Timer0_CompB, INT#
mega640, mega1280, mega2560, mega1281, mega2561	INT#	Timer2_CompB, INT#
mega8U2, mega16U2, mega32U2, AT90USB82, AT90USB162	INT#	Timer0_CompB, INT#
AT90USB162, AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	INT#	Timer2_CompB, INT#
all xmega	A.5	TCC1_CCB, ACA_AC0

Compatibility

If the RTC is not enabled in your application or if the target device does not have a second output compare register on the RTC timer, this routine will not be available. Moreover, it is not available in BasicX compatibility mode.

See Also   CloseX10, DefineX10, SetQueueX10, StatusX10

OutputCapture

Type	Subroutine		

Invocation   OutputCapture(intevals, count, flags)

Parameter	Method	Type	Description
intervals	ByRef	array of int16	The lengths of successive segments of the output waveform.
count	ByVal	int16	The number of entries in the value array.
flags	ByVal	Byte	Configuration bits controlling the generation process.

Discussion

This subroutine produces a series of precisely timed logic levels on the OutputCapture pin (see table below) allowing you to produce an arbitrary waveform. Each entry in the `intervals` array specifies a time interval, in units of the I/O Timer period (i.e. `1/F_CPU`, about 67.8ns for devices running at 14.7MHz), for each segment of the waveform. When called, the OutputCapture pin will be made an output and will be set to its initial state (the complement of the least significant bit of the `flags` parameter).

When waveform generation is begun, the OutputCapture pin will be changed to the opposite state for the interval specified by the first `intervals` element, changed to the opposite state again for the interval specified by the second `intervals` element, etc. for as many elements as specified. The final state of OutputCapture pin depends on whether the `count` parameter is odd or even. If it is odd the final state will be the complement of the least significant bit of the `flags` parameter; if it is even the final state will be the same as the least significant bit of the `flags` parameter.

The calling task will be suspended during the waveform generation process. If another task disables interrupts the accuracy of the generated waveform may suffer.

Due to processing overhead, the smallest pulse width that can be accommodated is about 90 CPU cycles (6µS at 14.7MHz). This corresponds to a value of about 88 in the data array at the default timer speed. If the system has a heavy interrupt load (e.g. serial channels 3-6 are open) the minimum pulse width for reliable operation may be significantly larger. The maximum pulse width using the default timer speed is about 4.4mS at 14.7MHz. If you need to generate longer pulse widths, you may set the value of `Register.TimerSpeed1` so that a slower clock rate is used.

To avoid unwanted logic transitions on the OutputCapture pin during preparation for waveform generation, the OutputCapture pin should be configured as an input prior to the call. You'll probably need to employ a pullup or pulldown resistor on the pin to guarantee the desired logic state prior to the commencement of waveform generation.

Resource Usage

See the Resource Usage sub-section Output Capture Timers for information about the output pin and, for native mode devices, the ISRs that will be included in the application.

Compatibility

For ZX devices running at 14.7MHz, since the CPU runs at twice the rate as the BasicX CPU, the units of the pulse width are half as long. If you need to generate longer pulse widths, you may set the value of `Register.TimerSpeed1` so that a slower timer clock rate is used. Also, the BasicX documentation indicates that if the I/O Timer is already in use, that use will be terminated and the waveform generation will be performed.

See Also   OutputCaptureEx

OutputCaptureEx

Type   Subroutine

Invocation   OutputCaptureEx(pin, intervals, count, flags)
            OutputCaptureEx(pin, intervals, count, flags, repeatCount)

Parameter	Method	Type	Description
pin	ByVal	Byte	Specifies the waveform output pin.
intervals	ByRef	array of int16	The lengths of successive segments of the output waveform.
count	ByVal	any int	The number of entries in the `intervals` array (1-65535).
flags	ByVal	Byte	Configuration bits controlling the generation process.
repeatCount	ByVal	any int	The number of times to repeat the pattern (1-65535).

Discussion

This subroutine produces a series of precisely timed logic levels on the specified pin allowing you to produce an arbitrary waveform. Each entry in the `intervals` array specifies a time interval, in units of the I/O Timer clock period using the TimerSpeed 1 prescaler setting (i.e. `1/F_TS1`, by default about 67.8ns for devices running at 14.7MHz), for each segment of the waveform. When called, the specified pin will be made an output and will be set to its initial state (the complement of the least significant bit of the `flags` parameter).

When waveform generation is begun, the specified pin will be changed to the opposite state for the interval specified by the first `intervals` element, changed to the opposite state again for the interval specified by the second `intervals` element, etc. for as many elements as specified. The final state of the output pin depends on whether the `count` parameter is odd or even. If it is odd the final state will be the complement of the least significant bit of the `flags` parameter; if it is even the final state will be the same as the least significant bit of the `flags` parameter.

If the optional `repeatCount` parameter is not given a repeat count of 1 is assumed. If the repeat count is 1 the `intervals` array should generally have an odd number of values. This allows the output to end in the same state as it started. If the repeat count is greater than one the `intervals` array should generally have an even number of values. This allows the output waveform to repeat at the same logic levels. Also, when the waveform is repeated the last interval of the last cycle is omitted so that the output ends up in the same state as it started.

The calling task will be suspended during the waveform generation process. If another task disables interrupts, the accuracy of the generated waveform will suffer.

Due to processing overhead, the smallest pulse width that can be accommodated is equivalent to about 100 CPU cycles (6.8μS at 14.7MHz). This corresponds to a value of about 100 in the data array at the default timer speed. If the system has a heavy interrupt load (e.g. serial channels 3-6 are open) the minimum pulse width for reliable operation may be significantly larger. The maximum pulse width using the default timer speed is about 4.4mS. If you need to generate longer pulse widths, you may set the value of `Register.TimerSpeed1` so that a slower clock rate is used.

To avoid unwanted logic transitions on the output pin during preparation for waveform generation, the output pin should either be configured as an input or as an output in the desired starting state prior to the call. If you configure it as an input you'll probably need to employ a pullup or pulldown resistor on the pin to guarantee the desired logic state prior to the commencement of waveform generation.

Although this subroutine can be invoked specifying a specific hardware OutputCapture pin (see the tables in the Resource Usage sub-section Output Capture Timers) or a general I/O pin, the behavior when using a general I/O pin may be slightly different than when using the specific hardware OutputCapture pin. The hardware OutputCapture pin uses features of the hardware to toggle the I/O pin while for general I/O pins

the pin is toggled in software by directly setting the corresponding PORTx bit. During periods of high interrupt load the hardware toggling will be more accurate.

Some ZBasic target devices support ability to use the output capture waveform to modulate a carrier waveform produced by the Serial Timer; such devices are listed in the table below and also in the Resource Usage sub-section Output Capture Timers where a second output capture pin for Timer1 is shown.

Output Capture Modulation Carrier Timer and Output Pin

Target Device	Serial Timer Compare Output	Output Pin
ZX-1281, ZX-1281n	OC0A	17, B.7
ZX-1280, ZX-1280n	OC0A	26, B.7
ZX-128e, ZX-128ne	OC2	21, B.7
ZX-1281e, ZX-1281ne	OC0A	21, B.7
mega640, mega1280, mega2560	OC0A	26, B.7
mega64, mega64A, mega128, mega128A	OC2	17, B.7
mega1281, mega2561, AT90CAN32, AT90CAN64, AT90CAN128, AT90USB646, AT90USB647, AT90USB1286, AT90USB1287	OC0A	17, B.7

To implement Output Capture modulation, the Serial Timer must be set up to generate the desired carrier frequency and duty cycle prior to calling OutputCaptureEx. Note, however, that the actual output from the serial timer should not be enabled – this will be done automatically when OutputCaptureEx is called. Further, the TimerBusy flag for the Serial Timer must be set to indicate that it is active. Finally, when calling OutputCaptureEx, the value &H02 should be added to the `flags` parameter to request output capture modulation.

Resource Usage

See the Resource Usage sub-section Output Capture Timers for information about the timers, output pins and (for native mode devices) the ISRs that will be included in the application.   If the timer is already in use the routine will return immediately without performing the waveform generation. Note that when performing an output capture on a general I/O pin, any available 16-bit timer may be used to generate the required timing.

Compatibility

This routine is not available in BasicX compatibility mode.

ParityCheck

Type	Function returning Boolean
Invocation	ParityCheck(data, oddParity)

Parameter	Method	Type	Description
data	ByVal	Byte	The data value for which to check the parity.
oddParity	ByVal	Boolean	The desired parity: True -> odd parity, False -> even parity

Discussion

This function computes the parity over the eight bits of the provided data value and compares that result to the desired result indicated by the oddParity parameter. The return value is a pass/fail indicator where True means that the parity matched the desired parity.

The data value has even parity if the number of one bits in the value is even.

Example

```
Dim b as Byte

If Not ParityCheck(b, False) Then
      Debug.Print "Even parity check failed"
End If
```

Compatibility

This routine is not available in BasicX compatibility mode.

Pause

Type	Subroutine
Invocation	Pause(time)

Parameter	Method	Type	Description
time	ByVal	Single or int16	The amount of time to pause, in seconds (Single) or ticks (int16)

Discussion

This routine suspends execution of the current task for approximately the period of time specified. When provided with an Int16 parameter, the units will be will be the period of the rate of change RTC timer (1/F_RTC_TIMER or 4.34µS for most ZX devices) if the RTC is included in the application. If the RTC is not included the units are 1uS intervals. The maximum pause duration is 65535 units

No other task is allowed to run during the pause period. Note that the accuracy of the pause may be affected by the time required for the processor to service interrupts (RTC, serial channel, etc.). Also note that the resolution of the pause is similar to the minimum execution time for user instructions. This means that timing using `Pause()` calls of less than 20 to 50 units or so will be affected significantly by the succeeding instructions.

This routine should be used instead of `Sleep()` or `Delay()` when higher resolution timing is required or you don't want a task switch to occur. If you need longer pauses than can be produced by this routine, you can implement them using `Register.RTCStopWatch`.

Example

```
Do
     Call PutPin(12, 0)
     Call Pause(0.010)       ' a 10 millisecond delay
     Call PutPin(12, 1)
     Call Pause(2304)        ' a 10 millisecond delay
Loop
```

This loop produces a square wave signal on pin 12 at approximately 50Hz (with some jitter due to handling interrupts).

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   Delay, DelayUntilClockTick, Sleep, WaitForInterval

PeekQueue

Type   Subroutine

Invocation   PeekQueue(queue, var, count)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue from which to retrieve data.
var	ByRef	any type	The variable to receive the retrieved data.
count	ByVal	int16	The number of bytes to retrieve.

Discussion

This routine will copy the specified number of bytes from the queue to the indicated variable but it does not remove them from the queue. The routine will not return until it can copy the entire number of bytes specified. Because of this, you should usually check the number of bytes available in the queue using `GetQueueCount()` before calling `PeekQueue()`.

Note that if the calling task is locked and the queue contains insufficient data when this routine is called, the task will be unlocked to allow other tasks to run.

Caution

If the requested number of bytes is larger than the queue capacity, the routine will never return. Likewise, if not enough data is placed in the queue, the routine will never return. Also, if the variable to receive the data is smaller than the number of bytes indicated, adjacent memory will be overwritten, usually with undesirable results.

Example

Compatibility

BasicX allows any type for the first parameter. The ZBasic implementation requires that it be an array of `Byte`.

PersistentPeek

Type   Function returning Byte

Invocation   PersistentPeek(address)

Parameter	Method	Type	Description
address	ByVal	int16	The persistent memory address from which to read.

Discussion

This function will return the content of the specified persistent memory address.

The address of any persistent variable can also be obtained using the `DataAddress` property. For persistent variables, the `DataAddress` property is of type `UnsignedInteger`.

Example

```
Dim pi as PersistentInteger
Dim b as Byte
b = PersistentPeek(1000)
b = PersistentPeek(pi.DataAddress + 1)
```

The second use of `PersistentPeek()` demonstrates how you can use the `DataAddress` property to read a byte value from any part of a persistent variable of any type.

Compatibility

BasicX does not support the use of the `DataAddress` property for persistent items.

The BasicX system has only 512 bytes of persistent memory. In ZBasic, the amount of persistent memory available depends on the particular target device; the first 32 bytes of persistent memory are reserved for system use.

See Also   PersistentPoke

PersistentPoke

Type   Subroutine

Invocation   PersistentPoke(value, address)

Parameter	Method	Type	Description
value	ByVal	Byte	The to write to persistent memory.
address	ByVal	int16	The persistent memory address to which to write.

Discussion

This routine will write the given value to the specified persistent memory address.

The address of any persistent variable can also be obtained using the `DataAddress` property. For persistent variables, the `DataAddress` property is of type `UnsignedInteger`.

Caution

The first 32 bytes of persistent memory are reserved for the system. Modifying any of them may produce unpredictable results.

The persistent memory (on-board EEPROM) has a limit specified by the manufacturer of a million write cycles. When this limit is exceeded the memory may become unreliable.

Example

```
Dim pi as PersistentInteger
Call PersistentPoke(&H55, 1000)
Call PersistentPoke(&H55, pi.DataAddress + 1)
```

The second use of `PersistentPoke()` demonstrates how you can use the `DataAddress` property to write a byte value to any part of a persistent variable of any type.

Compatibility

BasicX does not support the use of the `DataAddress` property for persistent items.

The BasicX system has only 512 bytes of persistent memory. In ZBasic, the amount of persistent memory available depends on the particular target device; the first 32 bytes of persistent memory are reserved for system use.

See Also   PersistentPeek

PinHigh

Type   Subroutine

Invocation   PinHigh(pin)

Parameter	Method	Type	Description
pin	ByVal	integral	A pin number (value must be known at compile time).

Discussion

This subroutine sets an output pin to the high state. If the value of the `pin` parameter is not known at compile time (e.g. a constant) a compile error will result. If the pin has not been previously configured to be an output, the effect of invoking PinHigh on that pin is undefined (varies by target device).

The advantage of using this subroutine instead of using PutPin() to achieve the same result is that the call to this subroutine results in much smaller/faster code. The disadvantage is that the pin number must be known at compile time.

Compatibility

This subroutine is only available for native mode devices (ZX or generic target devices) and is not available in BasicX compatibility mode.

See Also   PinLow, PinOutput, PinToggle, PutPin

PinInput

Type	Subroutine
Invocation	PinInput(pin, flags)
	PinInput(pin)

Parameter	Method	Type	Description
pin	ByVal	integral	A pin number (must be known at compile time).
flags	ByVal	integral	Flag bits giving additional configuration information.

Discussion

This subroutine configures a pin as an input. If the value of the `pin` parameter is not known at compile time (e.g. a constant) a compile error will result. The second form of invocation is equivalent to the first form with `flags` set to zero. The `flags` parameter gives additional configuration information as shown in the table below.

Flags Bits	Description
xxxx xxx0	Pullup disabled.
xxxx xxx1	Pullup enabled.

All other bits are reserved for future use and should be set to zero for compatibility with such changes.

The advantage of using this subroutine instead of using PutPin() to achieve the same result is that the call to this subroutine results in much smaller/faster code. The disadvantage is that the pin number must be known at compile time.

Compatibility

This subroutine is only available for native mode devices (ZX or generic target devices) and is not available in BasicX compatibility mode.

See Also   PinOutput, PutPin

PinLow

Type   Subroutine

Invocation   PinLow(pin)

Parameter	Method	Type	Description
pin	ByVal	integral	A pin number (value must be known at compile time).

Discussion

This subroutine sets an output pin to the low state. If the value of the `pin` parameter is not known at compile time (e.g. a constant) a compile error will result. If the pin has not been previously configured to be an output, the effect of invoking PinLow on that pin is undefined (varies by target device).

The advantage of using this subroutine instead of using PutPin() to achieve the same result is that the call to this subroutine results in much smaller/faster code. The disadvantage is that the pin number must be known at compile time.

Compatibility

This subroutine is only available for native mode devices (ZX or generic target devices) and is not available in BasicX compatibility mode.

See Also   PinHigh, PinOutput, PinToggle, PutPin

PinOutput

Type   Subroutine

Invocation   PinOutput(pin, flags)
                    PinOutput(pin)

Parameter	Method	Type	Description
pin	ByVal	integral	A pin number (value must be known at compile time).
flags	ByVal	integral	Flag bits giving additional configuration information.

Discussion

This subroutine configures a pin as an output. If the value of the `pin` parameter is not known at compile time (e.g. a constant) a compile error will result. The second form of invocation is equivalent to the first form with `flags` set to zero. The `flags` parameter gives additional configuration information as shown in the table below.

Flags Bits	Description
xxxx xxx0	Initial state is low.
xxxx xxx1	Initial state is high.

All other bits are reserved for future use and should be set to zero for compatibility with such changes.

The advantage of using this subroutine instead of using PutPin() to achieve the same result is that the call to this subroutine results in much smaller/faster code. The disadvantage is that the pin number must be known at compile time.

Compatibility

This subroutine is only available for native mode devices (ZX or generic target devices) and is not available in BasicX compatibility mode.

See Also   PinHigh, PinInput, PinLow, PinToggle, PutPin

PinRead

Type   Function returning Byte

Invocation   PinRead(pin)

Parameter	Method	Type	Description
pin	ByVal	integral	A pin number (value must be known at compile time).

Discussion

This function returns an indicator of the logic level present on the specified pin. A return value of zero indicates a logic low and a non-zero return value indicates a logic high. If the value of the `pin` parameter is not known at compile time (e.g. a constant) a compile error will result. Note that the specified pin may have previously been configured as either an input or an ouput.

The advantage of using this function instead of using GetPin() to achieve the same result is that the call to this function results in much smaller/faster code and that this function does not reconfigure the pin to be an input. The disadvantage is that the pin number must be known at compile time.

Compatibility

This function is only available for native mode devices (ZX or generic target devices) and is not available in BasicX compatibility mode.

See Also   GetPin

PinToggle

Type   Subroutine

Invocation   PinToggle(pin)

Parameter	Method	Type	Description
pin	ByVal	integral	A pin number (value must be known at compile time).

Discussion

This subroutine changes the state of an output pin to the opposite of its current state. If the value of the `pin` parameter is not known at compile time (e.g. a constant) a compile error will result. If the pin has not been previously configured to be an output, the effect of invoking PinToggle on that pin is undefined (varies by target device).

The advantage of using this subroutine instead of using PutPin() to achieve the same result is that the call to this subroutine results in much smaller/faster code. The disadvantage is that the pin number must be known at compile time.

Compatibility

This subroutine is only available for native mode devices (ZX or generic target devices) and is not available in BasicX compatibility mode.

See Also   PinHigh, PinLow, PinOutput, PutPin

PlaySound

Type　　　　　Subroutine

Invocation　　PlaySound(pin, address, length, rate, repeat)

Parameter	Method	Type	Description
pin	ByVal	Byte	The output pin.
address	ByVal	int16	The Program Memory address of the sound data.
length	ByVal	int16	The number of bytes of sound data.
rate	ByVal	int16	The sample rate for the sound data.
repeat	ByVal	int16	The number of times to repeat the sound.

Discussion

This routine uses a pseudo-PWM technique to create an approximation to a sine wave on the specified output pin. The frequency of the sine wave is given by successive bytes in Program Memory beginning at the specified address and continuing for the given length. The `rate` parameter specifies the rate at which the data elements will be utilized. It is equivalent to the sampling rate at which an original analog sound might have been digitized. Lastly, the repeat parameter tells how many times to repeat the production of the output using the supplied data. If zero is specified, the sound will be repeated 65,536 times.

The minimum supported sample rate is 250Hz. If a smaller value is specified, 250Hz will be used instead.

The actual output will be a pulse stream that has an average value that approximates the target analog signal. Because of the high frequency nature of the pulse train used to synthesize the waveform some filtering is required. The example circuit below may be used to couple the output to a high impedance speaker (> 40) or an amplifier. Note, however, that the signal is too large to be fed to the microphone input of an amplifier. Instead, the Auxiliary or Line input should be used.
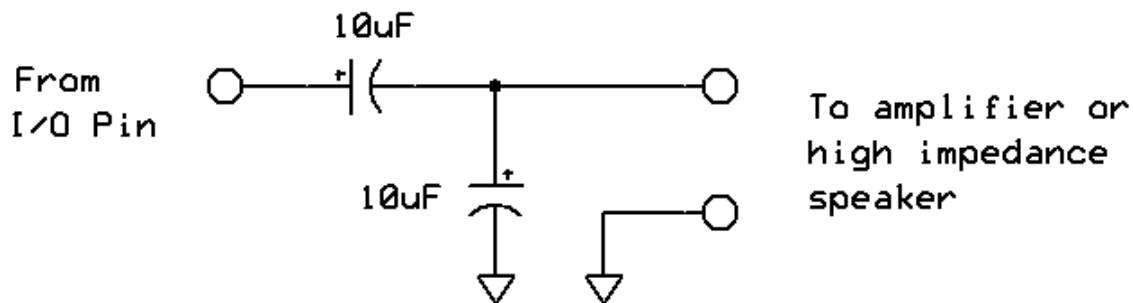
Resource Usage

This routine uses the I/O Timer and disables interrupts during the generation process. In particular, this means that serial input that arrives during the generation will likely be missed and serial output on channels 3-6 will be disrupted.

Task switching is suspended and other interrupts are disabled while the sound is being produced. However, RTC ticks are accumulated during the process and the RTC is updated when the process has completed so that the RTC does not lose time.

Example

```
Dim music as ByteVectorData("sound.txt")

Call PlaySound(12, LoWord(music.DataAddress), UBound(music), 11025, 1)
```

This example assumes that you have prepared the file "sound.txt" to contain the digitized music, sampled at 11025Hz.

Compatibility

The BasicX documentation for `PlaySound()` does not explicitly indicate that a zero repeat count will result in 65,536 iterations. However, experimental evidence indicates that it does.

In the BasicX implementation the RTC will lose time if the duration is too long. It is not known if the BasicX implementation has a minimum sample rate.

PortBit

Type	Function returning Byte
Invocation	PortBit(portIdx, bitIdx) PortBit(pin)

Parameter	Method	Type	Description
portIdx	ByVal	integral	The I/O port designator (A=0, B=1, etc.)
bitIdx	ByVal	integral	The bit designator (0-7)
pin	ByVal	integral	A pin number

Discussion

This function returns a composite value that describes a specific bit in a specific I/O port. The fields of the Byte value are as shown in the table below.

Bit(s)	Description
7	Always 1
6-3	The I/O port designator (A=0, B=1, etc.)
2-0	The bit designator (0-7)

When invoked in the first form with the parameter values 2 and 6 (representing Port C, bit 6) the return value will have the bit pattern `&B10010110`.

The second form of invocation converts a physical pin number to the composite value representing the port and bit corresponding to that pin. When passed an invalid pin, the return value is zero.

Values returned by the PortBit() function may be used anywhere that a pin number may be used, e.g. as the first parameter to PutPin(). The primary advantage to using the composite port/bit designator is that the same value may be used unchanged on any ZBasic device having the referenced pin.

Note that the special port/bit designators like `C.2` are converted by the compiler to the same type of composite port/bit designator described here if the compiler directive `Option PortPinEncoding On` is specified.

Compatibility

This function is not available in BasicX compatibility mode.

PortMask

Type   Function returning Byte

Invocation   PortMask(pin)

Parameter	Method	Type	Description
pin	ByVal	integral	A pin number

Discussion

This function returns a bit mask for the port with which the specified pin is associated. The resulting bit mask will have at most one bit set if the pin is valid and will be zero for an invalid pin. The bit mask can be used for directly manipulating the I/O registers associated with a pin.

Note that the value of this function is a compile-time constant if the compiler can determine the value of the pin parameter at compile-time.

For further information about how to use this function, see the discussion of `Register.Port()` in the ZBasic Reference Manual.

Example

```
Dim mask as Byte

mask = PortMask(C.2)     ' the result will be &H04
```

Compatibility

This function is not available in BasicX compatibility mode.

Pow

Type   Function returning Single

Invocation   Pow(mantissa, exponent)

Parameter	Method	Type	Description
mantissa	ByVal	Single	The value to be raised to the power given by the exponent.
exponent	ByVal	Single	The exponent value.

Discussion

This function returns the value of the first parameter raised to the power given by the second parameter. This is the same functionality as provided by the exponentiation operator ^.

Certain special cases are detected as shown in the table below.

Mantissa	Exponent	Result
any value	0.0	1.0
negative	non-integral value	NaN
0.0	Negative	+Infinity

Example

```
Dim r as Single, f as Single

f = 10.0
r = Pow(f, 2.0)    ' result is 100.0
```

See Also   Exp, Exp10

ProgMemFind

Type	Function returning UnsignedInteger		

Invocation ProgMemFind(dataAddr, dataLen, val)
ProgMemFind(dataAddr, dataLen, val, ignoreCase)

Parameter	Method	Type	Description
dataAddr	ByVal	Long	The address in Program Memory of the block to search.
dataLen	ByVal	integral	The length of the block to search.
val	ByVal	Byte	The byte value for which to search.
ignoreCase	ByVal	Boolean	A flag controlling whether alphabetic case is significant.

Discussion

This function attempts to find the first occurrence of the byte specified by the val parameter in a block of data in Program Memory beginning at the specified address. If it is found, the return value gives the 1-based index where the sought byte was found within the block. If the sought byte is not found, zero is returned. If the optional ignoreCase parameter is not given, the search is performed observing alphabetic case differences, otherwise alphabetic case differences are significant or not depending on the value specified for ignoreCase. For the purposes of this parameter only the characters A-Z and a-z (&H41 to &H5a and &H61 to &H7a) are considered to be alphabetic.

Example

```
Dim charSet as ByteVectorData ({ ".$%'-_@~`!(){}^#&" })
Dim inCharSet as Boolean

If (ProgMemFind(charSet.DataAddress, SizeOf(charSet), c) <> 0) Then
     inCharSet = True
End If
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also MemFind, StrFind

PulseIn (subroutine form)

Type   Subroutine

Invocation   PulseIn(pin, level, var)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin on which a pulse width will be measured.
level	ByVal	Byte	The expected pulse logic value (high = 1).
var	ByRef	Single	The variable to receive the pulse width value.

Discussion

This routine waits for the input pin to be in the idle state (the opposite of that specified by the `level` parameter), waits for it to change to the specified logic level and then measures the time that it stays at that level. The pulse width is stored in the specified variable and has units of seconds with a default resolution as shown in the table below.

PulseIn Resolution

Target	I/O Scaling	Resolution
ZX devices running at 14.7456MHz	True	1.085 µS
	False	0.542 µS
all other targets	n/a	1/F_TS2

The pin is made an input if it is not already so. If the awaited logic transition never occurs or if the pulse width exceeds the maximum representable width the stored result will be zero.

The timing resolution may be adjusted using `Register.TimerSpeed2`. However, if this is done, the resulting pulse width value will need to be scaled proportionally.

Resource Usage

This routine uses the I/O Timer and interrupts are disabled during the pulse measurement. However, RTC ticks will be accumulated during the pulse measurement and the RTC will be updated when the process is complete.

Example

```
Dim width as Single

Call PulseIn(12, 1,width)      ' measure a positive-going pulse
```

Compatibility

The BasicX implementation does not support adjustable timing resolution.

PulseIn (function form)

Type   Function returning Integer

Invocation   PulseIn(pin, level)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin on which a pulse width will be measured.
level	ByVal	Byte	The expected pulse logic value (high = 1).

Discussion

This routine waits for the input pin to be in the idle state (the opposite of that specified by the `level` parameter), waits for it to change to the specified logic level and then measures the time that it stays at that level. The width of the pulse is returned by the function, the units of which are shown in the table below.

PulseIn Units		
Target	**I/O Scaling**	**Resolution**
ZX devices running at 14.7456MHz	True	1.085 µS
	False	0.542 µS
all other targets	n/a	1/F_TS2

The pin is made an input if it is not already so. If the awaited logic transition never occurs or if the pulse width exceeds the maximum representable width the returned value will be zero.

The timing resolution may be adjusted using `Register.TimerSpeed2`.

Resource Usage

This routine uses the I/O Timer and interrupts are disabled during the pulse measurement. However, RTC ticks will be accumulated during the pulse measurement and the RTC will be updated when the process is complete.

Example

```
Dim width as Integer

i = PulseIn(12, 1)      ' measure a positive pulse
```

Compatibility

The BasicX implementation does not support adjustable timing resolution.

PulseOut

Type   Subroutine

Invocation   PulseOut(pin, duration, level)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin on which a pulse width will be generated.
duration	ByVal	int16 or Single	The width of the generated pulse.
level	ByVal	Byte	The desired pulse logic value (low = 0, high = 1).

Discussion

This routine first makes the specified pin an output. (However, for practical purposes, you should generally make the pin an output and set it to the desired state before calling this routine.) Then it sets the pin to the active state (as indicated by the `level` parameter), waits the specified time and then sets the pin back to the inactive state. The pin will be left configured as an output.

The pulse width may be specified by a Single value with units of seconds and a resolution as shown in the table below. Note, however, that due to processing overhead, the shortest pulse that can be generated is approximately 200 CPU cycles (13us at 14.7MHz). Alternately, the pulse width may be specified by an Integer or UnsignedInteger value with units as shown in the table below. Note, however, that Register.TimerSpeed2 may be modified to adjust the I/O Timer tick rate. If this is done, the Single value will have to be scaled proportionally.

PulseOut Resolution

Target	I/O Scaling	Resolution
ZX devices running at 14.7456MHz	True	1.085 µS
	False	0.542 µS
all other targets	n/a	1/F_TS2

If the output pin is specified as zero, this routine does not generate a pulse but will delay for approximately the specified period of time. This may be useful for generating a delay with better precision than can be obtained by using `Delay()` or `Sleep()`. Moreover, generating a delay in this manner does not cause the task to lose control.

Resource Usage

This routine uses the I/O Timer and interrupts are disabled during the pulse generation. However, RTC ticks will be accumulated during the pulse generation and the RTC will be updated when the process is complete. If the pulse is too long characters being sent or received on serial channels 3-6 may be garbled.

Example

```
Dim width as Integer

Call PutPin(12, zxOutputLow)
Call PulseOut(12, 2, 1)    ' generate a positive pulse about 2µS long
Call PulseOut(0, 1e-5, 0)  ' generate a delay of about 10µS
```

Compatibility

In the BasicX implementation, the RTC will lose time if the pulse is too long. Also, the BasicX implementation does not support adjustable timing resolution.

Put1Wire

Type   Subroutine

Invocation Put1Wire(pin, value)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin to be used for 1-Wire I/O.
value	ByVal	Byte	The bit value to write.

Discussion

This routine sends the LSB of the given value using the 1-Wire protocol. To perform a 1-Wire operation, this function along with related 1-Wire routines must be used in the proper sequence. See the specifications of your 1-Wire device for more information.

Resource Usage

This routine uses the I/O Timer and disables interrupts for about 100µS.

Example

```
Call Put1Wire(12, 1)
```

See Also   Get1Wire, Get1WireByte, Get1WireData,
Put1WireByte, Put1WireData, Reset1Wire

Put1WireByte

Type   Subroutine

Invocation   Put1WireByte(pin, value)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin to be used for 1-Wire I/O.
value	ByVal	Byte	The value to write.

Discussion

This routine sends a byte (LSB first) using the 1-Wire protocol. To perform a 1-Wire operation, this function along with related 1-Wire routines must be used in the proper sequence. See the specifications of your 1-Wire device for more information.

Example

```
Call Put1WireByte(12, &H55)
```

Resource Usage

This routine uses the I/O Timer and disables interrupts for about 100µS for each bit sent.

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   Get1Wire, Get1WireByte, Get1WireData,
Put1Wire, Put1WireData, Reset1Wire

Put1WireData

Type   Subroutine

Invocation   Put1WireData(pin, data, count)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin to be used for 1-Wire I/O.
data	ByRef	any type	A variable holding the bytes to write.
count	ByVal	Byte	The number of bytes to write.

Discussion

This routine sends 1 or more bytes of data (each LSB first) using the 1-Wire protocol. To perform a 1-Wire operation, this function along with related 1-Wire routines must be used in the proper sequence. See the specifications of your 1-Wire device for more information.

Example

```
Dim d(1 to 10) As Byte

Call Put1WireData(12, d, 5)
```

Resource Usage

This routine uses the I/O Timer and disables interrupts for about 100µS for each bit sent.

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   Get1Wire, Get1WireByte, Get1WireData,
Put1Wire, Put1WireByte, Reset1Wire

PutBit

Type   Subroutine

Invocation PutBit(var, bitNumber, val)

Parameter	Method	Type	Description
var	ByRef	any type	The variable to which the bit will be written.
bitNumber	ByVal	int8/16	The bit number to write.
val	ByVal	Byte	The bit value.

Discussion

This routine writes a single bit to memory beginning at the location of the specified variable. Bit numbers 0-7 are written to the byte at the specified location, bit numbers 8-15 are written to the subsequent byte, etc. In each case, the lower bit number corresponds to the least significant bit of the byte while the highest bit number corresponds to the most significant bit of a byte.

Only the least significant bit of the `val` parameter is used; the remaining bits are ignored.

Caution

If you specify a bit number beyond the number of bits in the specified variable, a byte in memory following the variable will be modified, perhaps with undesirable results.

Compatibility

In BasicX compatibility mode, the `bitNumber` parameter may only be specified using a `Byte` value.

See Also   GetBit

PutDAC

Type	Subroutine	

Invocation	PutDAC(pin, dacValue, dacAccumulator)
	PutDAC(pin, dacValue, dacAccumulator, cycles)

Parameter	Method	Type	Description
pin	ByVal	Byte	The output pin.
dacValue	ByVal	numeric	The desired output value. See discussion below.
dacAccumulator	ByRef	Byte	A value used in the DAC process. See the discussion below.
cycles	ByVal	Byte	The number of PWM cycles to perform.

Discussion

This routine creates a digital approximation of an analog signal on the specified pin using a pseudo-PWM technique. When called, the specified pin is made an output, a pulse train is generated having an average value equal to the `dacValue` parameter and then, after a fixed number of iterations, the pin is placed in the high impedance input state. If the output is filtered with a low pass filter, the voltage will, immediately after the process is completed, be at a level between zero and the processor voltage (usually +5 volts). However, the voltage will begin to decay at a rate dependent on the load presented to the filter. The voltage can be refreshed from time to time by calling `PutDAC()` again.

The `dacValue` parameter may be specified by a `Single` value or an integral value. If a `Single` value is supplied, it should be in the range 0.0 to 1.0 corresponding to the output range of 0 to the processor voltage (usually +5 volts). If an integral value is supplied, it should be in the range of 0 to 255 corresponding to the same output voltage range as above.

The `dacAccumulator` parameter is required to ensure continuity between successive calls to `PutDAC()`. The value of the parameter after the call should not be modified and the same parameter should be supplied on the next call. The initial value of the parameter is of no consequence. If your application uses `PutDAC()` to create an analog voltage on more than one pin at a time, a separate accumulator value must be used for each one.

If the `cycles` parameter is not specified, a single PWM cycle is performed. Each cycle will generate a burst of pulses for about 3000 CPU cycles (200µS at 14.7MHz) during which time interrupts will be disabled. At the end of each cycle, the pin is put in high impedance mode and interrupts are re-enabled. The process is then repeated if the cycle count is greater than one. A cycle count of zero causes no cycles to be performed.

The selection of components for the required filter depends on several factors. A larger capacitor will allow the voltage to hold longer but also takes longer to bring up to the proper voltage. As a rule of thumb, the product of the resistance (in ohms) and the capacitance (in farads) should be on the order of the number of cycles times one-fourth of the cycle time specified above. For example, with a 100 resistor and a 1µF capacitor, the cycle count should probably be 2 in order to bring the capacitor up to the desired voltage level.

For best results, you should probably follow the filter with a high impedance buffer such as a unity gain op amp circuit, an example of which is shown below. The op amp chosen is not particularly critical, nearly any will do the job.

For ZBasic devices based on the ATxmega, a hardware DAC is available. In most applications requiring a DAC, using the hardware DAC will produce much better results.
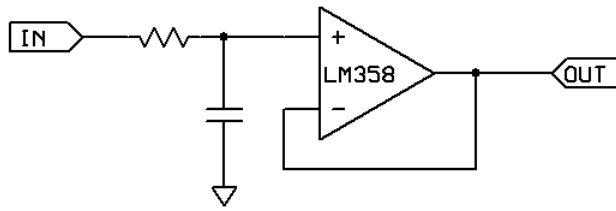
Examples

```
Dim acc as Byte

Call PutDAC(12, 0.5, acc)

Call PutDAC(12, 128, acc, 5)
```

Compatibility

In BasicX compatibility mode, the `dacValue` parameter may only be specified using a `Single` value. Also, the fourth parameter is not supported.

Resource Usage

This routine disables interrupts for about 3000 CPU cycles (200µS at 14.7MHz) during the generation process. Interrupts are reenabled between each successive cycle.

See Also   DAC, DACPin, OpenDAC

PutDate

Type   Subroutine

Invocation   PutDate(year, month, day)

Parameter	Method	Type	Description
year	ByVal	int16	The year value (1999-2177).
month	ByVal	Byte	The month value (1-12).
day	ByVal	Byte	The day value (1-31).

Discussion

This routine composes a new value for `Register.RTCDay` using the provided parameters. The month value of 1 corresponds to January while 12 corresponds to December. If the year or month is invalid or if the day number is invalid for the specified month and year, `Register.RTCDay` will be set to zero.

Note that `Register.RTCDay` is initialized to zero on power-up or reset. This corresponds to January 1, 1999.

Compatibility

This subroutine is not available if the RTC is not enabled in your application.

See Also   GetDate

PutEEPROM

Type   Subroutine

Invocation   PutEEPROM(addr, var, count)

Parameter	Method	Type	Description
addr	ByVal	Long	The Program Memory address at which to begin writing.
var	ByRef	any type	The variable from which the data to be written will be taken.
count	ByVal	int16	The number of bytes to write.

Discussion

This routine is provided for compatibility with BasicX. The more aptly named PutProgMem() should be used by new applications.

See Also   GetProgMem, PutProgMem

PutNibble

Type   Subroutine

Invocation   PutNibble(var, nibbleNumber, val)

Parameter	Method	Type	Description
var	ByRef	any type	The variable to which the nibble will be written.
nibbleNumber	ByVal	int8/16	The nibble number to write.
val	ByVal	Byte	The nibble value.

Discussion

This routine writes a single nibble (four bits) to memory beginning at the location of the specified variable. Nibble numbers 0-1 are written to the byte at the specified location, nibble numbers 2-3 are written to the subsequent byte, etc. In each case, the lower nibble number corresponds to the least significant four bits of the byte while the higher nibble number corresponds to the most significant four bits of the byte.

Only the least significant four bits of the `val` parameter is used; the remaining bits are ignored.

Caution

If you specify a nibble number beyond the number of nibbles in the specified variable, a byte in memory following the variable will be modified, perhaps with undesirable results.

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   GetNibble

PutPersistent

Type	Subroutine		

Invocation	PutPersistent(addr, var, count)		

Parameter	Method	Type	Description
addr	ByVal	int16	The Persistent Memory address to which to write.
var	ByRef	any type	The variable from which data will be taken.
count	ByVal	int8/16	The number of bytes to write.

Discussion

This routine reads one or more bytes from RAM and writes them to Persistent Memory beginning at the address given.

Caution

Persistent Memory has a write cycle limit of approximately a million writes. Writing to a particular address in excess of this limit may cause the memory to become unreliable.

A block of Persistent Memory starting at address zero is reserved for system use. When the compiler assigns addresses to persistent variables defined in your program, the lowest address used is the first address above this reserved block. The .map file generated by the compiler contains a section indicating the addresses assigned to persistent variables defined in your program. The built-in values `Register.PersistentStart`, `Register.PersistentSize` and `Register.PersistentUsed` may be useful for determining the allocated an unallocated portions of Persistent Memory.

This routine will write to any address in Persistent Memory. Generally, you should avoid writing to the reserved area of Persistent Memory.

Example

```
Dim pvar(1 to 10) as PersistentByte
Dim var(1 to 10) as Byte

Call PutPersistent(pvar.DataAddress, var, SizeOf(pvar))
```

Compatibility

This routine is not available in BasicX compatibility mode.

See Also	GetPersistent

PutPin

Type   Subroutine

Invocation   PutPin(pin, mode)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin to configure.
mode	ByVal	Byte	The configuration mode (see below).

Discussion

This routine is used to configure a pin to be an input or an output or to effect a change in the output logic level. If the pin is configured as an input, it may be configured to be in "tri-state" mode or "pull-up" mode. If the pin is configured to be an output, the output level may be set to zero or 1. The table below gives the values for each of the possible modes. If an invalid mode is specified or an invalid pin is specified, the routine has no effect.

Values for the `mode` Parameter

Value	Built-in Constant	Description
0	zxOutputLow	The pin is an output at logic zero.
1	zxOutputHigh	The pin is an output at logic one.
2	zxInputTriState	The pin is an input with the pull-up/pull-down resistors disabled.
3	zxInputPullUp	The pin is an input with the pull-up resistor enabled.
4	zxOutputToggle	Change the logic level of the output.
5	zxOutputPulse	Pulse the output.
6	zxInputPullDown	The pin is an input with the pull-down resistor enabled.
7	zxInvertIO	Input and output levels are inverted.
8	zxNormalIO	Input and output levels are normal (non-inverted).

Note that for modes 4 and 5 to be useful, the pin must have been previously set to be an output. Mode 4 (`zxOutputToggle`) will change the output to the opposite logic level. Mode 5 (`zxOutputPulse`) will change the output to the opposite level for a short period of time and then change it back to the original level. The duration of the pulse will be about 8 CPU cycles (approximately 0.5uS at 14.7456MHz).

Modes 6, 7 and 8 are only supported on xmega targets. Modes 7 and 8 are to be used in conjunction with the other modes (in separate calls, of course) to achieve the desired configuration.

Example

```
Call PutPin(12, zxOutputLow)  ' pin 12 will be at logic zero
```

Compatibility

In BasicX compatibility mode, *mode* values higher than 3 are not supported.

See Also   GetPin**,** PinHigh, PinInput, PinLow, PinOutput, PinToggle, PutPin

PutProgMem

Type   Subroutine

Invocation   PutProgMem(addr, var, count)

Parameter	Method	Type	Description
addr	ByVal	Long	The Program Memory address to which to begin writing.
var	ByRef	any type	The variable from which the data to be written will be taken.
count	ByVal	int16	The number of bytes to write.

Discussion

This routine writes one or more bytes to Program Memory (where the user program is stored) taking the data from RAM beginning at the location of the specified variable. Note that if a number of bytes is specified that is larger than the given variable, adjacent memory will be read.

Caution

Program Memory has a write cycle limit specified by the manufacturer of a million cycles. Writing to a particular address in excess of this limit may result in unreliable operation.

See Also   GetProgMem

PutQueue

Type   Subroutine

Invocation   PutQueue(queue, var, count)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue to which to write data.
var	ByRef	any type	The variable from which to read data to be written to the queue.
count	ByVal	int16	The number of bytes to write to the queue.

Discussion

This routine reads data from the variable and writes it to the specified queue. If there is insufficient space in the queue, the calling task will suspend until space becomes available. Note, particularly, that no data will be written until there is room for all the data to be written. This has two important ramifications. Firstly, if the number of bytes to be written is larger than the data capacity of the queue, the write will never complete. Secondly, if data is never taken out of the queue thus making room for the additional data, the write will also never complete.

Note that the number of bytes to write may be larger than the named variable. If this is the case, data will be taken from subsequent memory locations until the write count is satisfied. This may or may not be what you intended to occur.

Note, also, that before any queue operations are performed, the queue data structure must be initialized. See the discussion of `OpenQueue()` for more details. Also, attempting to put data in a queue that has been assigned to a Com port as the receive queue will produce undefined results.

Example

```
Dim outQueue(1 to 40) as Byte
Dim lval as Long

Call OpenQueue(outQueue, SizeOf(outQueue))
lval = &H55aa
Call PutQueue(outQueue, lval, SizeOf(lval))
```

Compatibility

BasicX allows any type for the first parameter. The ZBasic implementation requires that it be an array of `Byte`.

See Also   PutQueueByte, PutQueueStr

PutQueueByte

Type   Subroutine

Invocation   PutQueueByte(queue, val)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue to which to write data.
val	ByVal	Byte	The byte value to be written to the queue.

Discussion

This routine writes the given byte value to the specified queue. If there is insufficient space in the queue, the calling task will suspend until space becomes available. This means that if data is never taken out of the queue thus making room for additional data, the process will never complete.

Note that before any queue operations are performed, the queue data structure must be initialized. See the discussion of `OpenQueue()` for more details. Also, attempting to put data in a queue that has been assigned to a Com port as the receive queue will produce undefined results.

Example

```
Dim outQueue(1 to 40) as Byte

Call OpenQueue(outQueue, SizeOf(outQueue))
Call PutQueueByte(outQueue, &H55)
```

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   OpenQueue, PutQueue, PutQueueStr

PutQueueStr

Type   Subroutine

Invocation   PutQueueStr(queue, str)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue to which to write data.
str	ByVal	String	The string to be written to the queue.

Discussion

This routine writes the characters from the string to the specified queue. If there is insufficient space in the queue, the calling task will suspend until space becomes available. Note, particularly, that no data will be written until there is room for all the data to be written. This has two important ramifications. Firstly, if the number of bytes to be written is larger than the data capacity of the queue, the write will never complete. Secondly, if data is never taken out of the queue thus making room for the additional data, the write will also never complete.

Note that before any queue operations are performed, the queue data structure must be initialized. See the discussion of `OpenQueue()` for more details. Also, attempting to put data in a queue that has been assigned to a Com port as the receive queue will produce undefined results.

Example

```
Dim outQueue(1 to 40) as Byte

Call OpenQueue(outQueue, SizeOf(outQueue))
Call PutQueueStr(outQueue, "Hello, world!")
```

Compatibility

BasicX allows any type for the first parameter. The ZBasic implementation requires that it be an array of `Byte`.

See Also   PutQueueByte, PutQueue, OpenQueue

PutTime

Type Subroutine

Invocation PutTime(hour, minute, seconds)

Parameter	Method	Type	Description
hour	ByVal	Byte	The hour value (0-23).
minute	ByVal	Byte	The minutes value (0-59).
seconds	ByVal	Single	The seconds value (0.0 to 59.999)

Discussion

This routine combines the given values into the corresponding RTC tick count and stores the result in `Register.RTCTick`. Each parameter that is outside its corresponding legal range is considered to be zero.

Note that `Register.RTCTick` is initialized to zero on power-up or reset. This corresponds to 0:00:00.

See Also GetTime

PutTimeStamp

Type   Subroutine

Invocation   PutTimeStamp(year, month, day, hour, minute, seconds)

Parameter	Method	Type	Description
year	ByVal	int16	The year value (1999-2177).
month	ByVal	Byte	The month value (1-12).
day	ByVal	Byte	The day value (1-31).
hour	ByVal	Byte	The hour value (0-23).
minute	ByVal	Byte	The minutes value (0-59).
seconds	ByVal	Single	The seconds value.

Discussion

This routine combines the given date values into the corresponding `Register.RTCDay` value and combines the given time values into the corresponding RTC tick count and stores the result in `Register.RTCTick`. The effect is the same as if `PutDate()` and `PutTime()` had been called with their respective parameters.

Note that `Register.RTCDay` and `Register.RTCTick` are initialized to zero on power-up or reset.

PWM

Type	Subroutine		

Invocation   PWM(channel, dutyCycle)
PWM(channel, dutyCycle, status)

Parameter	Method	Type	Description
channel	ByVal	Byte	The channel to use for PWM generation.
dutyCycle	ByVal	Single or integral	The desired duty cycle.
status	ByRef	Boolean	The variable to receive the status value.

Discussion

This subroutine begins or modifies the generation of a 16-bit PWM signal on the specified channel. The channel must have been previously prepared for PWM generation by calling `OpenPWM()`. PWM generation is performed using one of the CPU's 16-bit timers, the number of which varies depending on the ZBasic device. See the Resource Usage sub-section 16-Bit PWM Timers for details of the available channels and the corresponding timer and output pin used.

The `dutyCycle` parameter specifies the desired duty cycle of the generated signal, expressing the percentage of time that the PWM signal will be at the logic 1 state. If the supplied parameter is of type `Single`, the value is in percent with a resolution of 0.01%. If the supplied parameter is integral, the units are percent, i.e., the value 100 means 100%. Specifying a `Single` value that is negative or any value greater than 100 will have an undefined effect.

The `status` parameter, if supplied, receives a value to indicate success or failure of the call.

If this subroutine is called without a preceding call to `OpenPWM()` to prepare the timer, the call will have no effect. This subroutine may be called multiple times to effect changes to the PWM signal's duty cycle while the signal is being generated. The change in duty cycle is synchronized so that it takes effect at the beginning of the next PWM pulse.

Example

```
Call OpenPWM(2, 50.0, zxFastPWM)' prepare for 50Hz Fast PWM using channel 2
Call PWM(2, 7.5)                 ' generate PWM with 7.5% duty cycle (1.5mS)
Call Delay(1.0)
Call PWM(2, 6.25)               ' generate PWM with 6.25% duty cycle (1.25mS)
```

Compatibility

This subroutine is not available in BasicX compatibility mode.

See Also   ClosePWM, OpenPWM

PWM8

Type	Subroutine		

Invocation	PWM8(channel, dutyCycle)		
	PWM8(channel, dutyCycle, status)		

Parameter	Method	Type	Description
channel	ByVal	Byte	The channel to use for 8-bit PWM generation.
dutyCycle	ByVal	Single or integral	The desired duty cycle.
status	ByRef	Boolean	The variable to receive the status value.

Discussion

This subroutine begins or modifies the generation of an 8-bit PWM signal on the specified channel. The channel must have been previously prepared for PWM generation by calling `OpenPWM8()`. Eight-bit PWM generation is performed using one of the CPU's 8-bit timers, the number of which varies depending on the ZBasic device. See the Resource Usage sub-section 8-Bit PWM Timers for details of the available channels and the corresponding timer and output pin used. Note that ZBasic devices based on ATxmega processors don't have any 8-bit timers so 8-bit PWM is not supported on those devices. The table below indicates the output pin for each PWM supported channel.

The `dutyCycle` parameter specifies the desired duty cycle of the generated signal, expressing the percentage of time that the PWM signal will be at the logic 1 state. If the supplied parameter is of type `Single`, the value is in percent with a resolution of 0.01%. If the supplied parameter is integral, the units are percent, i.e., the value 100 means 100%. Specifying a `Single` value that is negative or any value greater than 100 will have an undefined effect.

The `status` parameter, if supplied, receives a value to indicate success or failure of the call.

If this subroutine is called without a preceding call to `OpenPWM8()` to prepare the timer, the call will have no effect. This subroutine may be called multiple times to effect changes to the PWM signal's duty cycle while the signal is being generated. The change in duty cycle is synchronized so that it takes effect at the beginning of the next PWM pulse.

Example

```
Call OpenPWM8(1, 50.0, zxFastPWM) ' prepare for 50Hz Fast PWM
Call PWM8(1, 50.0)                ' generate PWM with 50% duty cycle
```

Compatibility

This subroutine is not available in BasicX compatibility mode nor is it available on ATxmega-based ZBasic devices.

See Also   ClosePWM8, OpenPWM8

RadToDeg

Type   Function returning Single

Invocation   RadToDeg(angle)

Parameter	Method	Type	Description
angle	ByVal	Single	The angle, in radians, to convert to degrees.

Discussion

The trigonometric functions in the System Library all use radian angle measure. Depending on the programming task, it is sometimes more convenient to think of angles in terms of degrees. This function and its inverst DegToRad() facilitate the conversion between the two systems.

Depending on optimization settings, if the parameter supplied to this function is known to be constant at compile time, the compiler will convert the value at compile time.   Otherwise, code is generated to perform the conversion (multiplication by a conversion factor) at run time.

Example

```
Dim f as Single
Dim theta as Single     ' the angle in degrees

theta = RadToDeg(Asin(f))
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   DegToRad

RamPeek

Type   Function returning Byte

Invocation   RamPeek(address)

Parameter	Method	Type	Description
address	ByVal	integral	The RAM address from which to read.

Discussion

This function will return the content of the specified RAM address.

Example

```
Dim b as Byte
Dim i as Integer

b = RamPeek(MemAddress(i))
b = RamPeek(i.DataAddress)
```

See Also   RamPeekDword, RamPeekWord

RamPeekDword

Type   Function returning UnsignedLong

Invocation   RamPeekDword(address)

Parameter	Method	Type	Description
address	ByVal	integral	The RAM address from which to read.

Discussion

This function will return the 4-byte value at the specified RAM address. The first byte will be the low order byte and the last will be the high order byte.

Example

```
Dim ul as UnsignedLong

ul = RamPeekDWord(200)
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   RamPeek, RamPeekWord

RamPeekWord

Type   Function returning UnsignedInteger

Invocation   RamPeekWord(address)

Parameter	Method	Type	Description
address	ByVal	integral	The RAM address from which to read.

Discussion

This function will return the 2-byte value at the specified RAM address. The first byte will be the low order byte and the following will be the high order byte.

Example

```
Dim u as UnsignedInteger

u = RamPeekWord(200)
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   RamPeek, RamPeekDword

RamPoke

Type   Subroutine

Invocation   RamPoke(value, address)

Parameter	Method	Type	Description
value	ByVal	Byte	The value to write to RAM.
address	ByVal	integral	The RAM address to which to write.

Discussion

This routine will write the given value to the specified RAM address.

Caution

Modifying user variables in this way may cause your program to malfunction. Writing to areas of RAM used by the system may cause your program to malfunction.

Examples

```
Dim b as Byte

Call RamPoke(&H55, MemAddress(b))
Call RamPoke(&H55, b.DataAddress)
```

See Also   RamPokeDword, RamPokeWord

RamPokeDword

Type   Subroutine

Invocation   RamPokeDword(value, address)

Parameter	Method	Type	Description
value	ByVal	any 32-bit	The value to write to RAM.
address	ByVal	integral	The RAM address to which to write.

Discussion

This routine will write the given value to the four bytes at the specified RAM address, least significant byte first.

Caution

Modifying user variables in this way may cause your program to malfunction. Writing to areas of RAM used by the system may cause your program to malfunction.

Example

```
Dim ul as UnsignedLong

Call RamPokeDword(&H117355aa, MemAddress(ul))
Call RamPokeDword(&H117355aa, ul.DataAddress)
```

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   RamPoke, RamPokeWord

RamPokeWord

Type   Subroutine

Invocation   RamPokeWord(value, address)

Parameter	Method	Type	Description
value	ByVal	int16	The value to write to RAM.
address	ByVal	integral	The RAM address to which to write.

Discussion

This routine will write the given value to the two bytes at the specified RAM address, least significant byte first.

Caution

Modifying user variables in this way may cause your program to malfunction. Writing to areas of RAM used by the system may cause your program to malfunction.

Example

```
Dim u as UnsignedInteger

Call RamPokeWord(&H55aa, MemAddress(u))
```

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   RamPoke, RamPokeDword

Randomize

Type   Subroutine

Invocation   Randomize()

Discussion

This routine seeds the random number generator with the value of Register.RTCTick. This is can be used to introduce some randomness into the sequence of values returned by `Rnd()` especially if the time that Randomize() gets called has some uncertainty due to external events, e.g. the time that a user takes to press a key.

See Also   Rnd

RCTime (subroutine form)

Type   Subroutine

Invocation   RCTime(pin, level, interval)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin to use.
level	ByVal	Byte	The expected initial logic level of the pin.
interval	ByRef	Single	The variable in which to return the charge/discharge interval.

Discussion

This routine measures how long the specified pin stays at the given logic level after it is made a tri-state input. The return value is expressed in seconds with a default resolution as shown in the table below this can be changed using `Register.TimerSpeed2`. If the maximum time elapses (32,767 units times the resolution) and the pin has not changed logic levels, the return value will be zero. If the pin is not at the specified level when called, the routine returns immediately with a value of approximately one unit of resolution. The pin will be left in the input tri-state mode.

RCTime Resolution

Target	I/O Scaling	Resolution
ZX devices running at 14.7456MHz	True	1.085 µS
	False	0.542 µS
all other targets	n/a	1/F_TS2

This function can be used with an external resistor-capacitor circuit to measure the value of one element when the other one is known. The charge/discharge time depends on the values of R and C as well as the initial and final voltages. Before calling this routine, you should make the specified pin an output and set it to the level specified.

Resource Usage

This routine uses the I/O Timer. If the timer is already in use when this routine is called, it will return immediately with a zero value. The same is true if the specified pin is invalid.

Task switching is suspended and interrupts are disabled while the charge/discharge time is being measured. However, RTC ticks are accumulated during the process and the RTC is updated when the process has completed so that the RTC does not lose time.

Example

See the function form of this routine for more information.

Compatibility

In BasicX, the ability to change the resolution using `Register.TimerSpeed2` is not supported.

The BasicX documentation indicates that the maximum value that can be returned is about 71ms. In this implementation, the maximum value that can be returned is about 32,767 units of resolution.

The BasicX implementation will miss RTC ticks if the charge/discharge time is too long.

RCTime (function form)

Type Function returning Integer

Invocation RCTime(pin, level)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin to use.
level	ByVal	Byte	The expected initial logic level of the pin.

Discussion

This function measures how long the specified pin stays at the given logic level after it is made a tri-state input. The return value has units as shown in the table below by default but this can be changed using `Register.TimerSpeed2`. If the maximum time elapses (32,767 units) and the pin has not changed logic levels, the return value will be zero. If the pin is not at the specified level when called, the routine returns immediately with a value of 1. The pin will be left in the input tri-state mode.

RCTime Units		
Target	**I/O Scaling**	**Resolution**
ZX devices running at 14.7456MHz	True	1.085 µS
	False	0.542 µS
all other targets	n/a	1/F_TS2

As an example, this function can be used with an external resistor-capacitor circuit to measure the value of one element when the other one is known. The charge/discharge time depends on the values of R and C as well as the initial and final voltages. Before calling this routine, you should make the specified pin an output and set it to the level specified.

Example

```
Const pin as Byte = 12

Call PutPin(pin, 1)  ' make the pin an output high to start charging
Call Delay(1.4e-4)   ' delay a bit to allow nearly full charging
i = RCTime(pin, 1)   ' measure the time to reach logic zero level
```

Resource Usage

This routine uses the I/O Timer. If the timer is already in use when this routine is called, it will return immediately with a zero value. The same is true if the specified pin is invalid.

Task switching is suspended and interrupts are disabled while the charge/discharge time is being measured. However, RTC ticks are accumulated during the process and the RTC is updated when the process has completed so that the RTC does not lose time.

Compatibility

In BasicX, the ability to change the resolution using `Register.TimerSpeed2` is not supported.

The BasicX implementation will miss RTC ticks if the charge/discharge time is too long.

Reset1Wire

Type   Function returning Byte

Invocation   Reset1Wire(pin)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin to be used for 1-Wire I/O.

Discussion

This function generates a reset signal on the given pin using the 1-Wire protocol. The return value is the "presence" bit sent by the attached 1-Wire device(s), if any. It will be zero if a 1-Wire device responded, 1 otherwise.

To perform a 1-Wire operation, this function along with related 1-Wire routines must be used in the proper sequence. See the specifications of your 1-Wire device for more information.

Resource Usage

This routine uses the I/O Timer and disables interrupts for approximately 1mS.

Example

```
Dim b as Byte

b = Reset1Wire(12)
```

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   Get1Wire, Get1WireByte, Get1WireData,
                  Put1Wire, Put1WireByte, Put1WireData

ResetProcessor

Type   Subroutine

Invocation   ResetProcessor()

Discussion

Calling this routine will cause a WatchDog reset of the processor within approximately 40ms. When the processor begins running again, the value of `Register.ResetFlags` will indicate that a WatchDog reset has occurred. If you need to be able to distinguish between an actual WatchDog reset and a call to `ResetProcessor()` it is recommended that you define a persistent variable and set its value to indicate the source of the reset.

Compatibility

BasicX does not support `Register.ResetFlags`.

ResetX10

Type   Subroutine

Invocation   ResetX10(chan, mask)

Parameter	Method	Type	Description
chan	ByVal	Byte	The X-10 communication channel of interest.
mask	ByVal	Byte	A mask value indicating which state flags to clear.

Discussion

Calling this routine will clear some of the flags that are returned by the StatusX10() function. The mask parameter should contain a value with a 1 in the bit positions corresponding to the state flags that you want to be cleared. Note that only a subset of the flags can be reset; asserted bits in the other bit positions are ignored. See the description of StatusX10() for more information.

Compatibility

This subroutine is supported only for native mode devices and is not available in BasicX compatibility mode.

See Also   StatusX10

ResumeTask

Type   Subroutine

Invocation   ResumeTask(taskStack)
ResumeTask()

Parameter	Method	Type	Description
taskStack	ByRef	array of Byte	The stack for a task of interest.

Discussion

This routine attempts to change the status of a task to a ready-to-run state. If no task stack is explicitly given, the task stack for the `Main()` routine is assumed. The table below shows the effect for various task states (as returned by `StatusTask()`).

Effect of Resuming a Task in Various States

Status	State	Effect
0	Ready to run.	None, the task is already ready to run.
1	Sleeping.	The task is awakened.
2	Awaiting `InputCapture()`.	The task resumes as if the `InputCapture()` had completed.
3	Awaiting interrupt 0.	The task resumes as if the interrupt had occurred.
4	Awaiting interrupt 1.	The task resumes as if the interrupt had occurred.
5	Awaiting interrupt 2.	The task resumes as if the interrupt had occurred.
6	Awaiting interval expiration.	The task resumes as if the interval had expired.
7	Awaiting analog compare.	The task resumes as if the comparison interrupt had occurred.
8	Awaiting pin change event 0.	The task resumes as if the pin change had occurred.
9	Awaiting pin change event 1.	The task resumes as if the pin change had occurred.
10	Awaiting pin change event 2.	The task resumes as if the pin change had occurred.
11	Awaiting pin change event 3.	The task resumes as if the pin change had occurred.
12	Awaiting `OutputCapture()`.	The task resumes as if the `OutputCapture()` had completed.
13	Awaiting interrupt 3.	The task resumes as if the interrupt had occurred.
14	Awaiting interrupt 4.	The task resumes as if the interrupt had occurred.
15	Awaiting interrupt 5.	The task resumes as if the interrupt had occurred.
16	Awaiting interrupt 6.	The task resumes as if the interrupt had occurred.
17	Awaiting interrupt 7.	The task resumes as if the interrupt had occurred.
18	Awaiting pin change event A0.	The task resumes as if the pin change event had occurred.
19	Awaiting pin change event A1.	The task resumes as if the pin change event had occurred.
20	Awaiting pin change event B0.	The task resumes as if the pin change event had occurred.
21	Awaiting pin change event B1.	The task resumes as if the pin change event had occurred.
22	Awaiting pin change event C0.	The task resumes as if the pin change event had occurred.
23	Awaiting pin change event C1.	The task resumes as if the pin change event had occurred.
24	Awaiting pin change event D0.	The task resumes as if the pin change event had occurred.
25	Awaiting pin change event D1.	The task resumes as if the pin change event had occurred.
26	Awaiting pin change event E0.	The task resumes as if the pin change event had occurred.
27	Awaiting pin change event E1.	The task resumes as if the pin change event had occurred.
28	Awaiting pin change event F0.	The task resumes as if the pin change event had occurred.
29	Awaiting pin change event F1.	The task resumes as if the pin change event had occurred.
30	Awaiting pin change event H0.	The task resumes as if the pin change event had occurred.
31	Awaiting pin change event H1.	The task resumes as if the pin change event had occurred.
32	Awaiting pin change event J0.	The task resumes as if the pin change event had occurred.
33	Awaiting pin change event J1.	The task resumes as if the pin change event had occurred.
34	Awaiting pin change event K0.	The task resumes as if the pin change event had occurred.
35	Awaiting pin change event K1.	The task resumes as if the pin change event had occurred.
36	Awaiting pin change event Q0.	The task resumes as if the pin change event had occurred.

37	Awaiting pin change event Q1.	The task resumes as if the pin change event had occurred.
38	Awaiting analog comp. A0.	The task resumes as if the analog event had occurred.
39	Awaiting analog comp. A1.	The task resumes as if the analog event had occurred.
40	Awaiting analog comp. AW.	The task resumes as if the analog event had occurred.
41	Awaiting analog comp. B0.	The task resumes as if the analog event had occurred.
42	Awaiting analog comp. B1.	The task resumes as if the analog event had occurred.
43	Awaiting analog comp. BW.	The task resumes as if the analog event had occurred.
254	Task exiting.	None, exiting tasks can't be resumed.
255	Terminated.	None, halted tasks can't be resumed.

If this routine is invoked using an array other than one that is or was being used for a task stack the result is undefined. See the section on Task Management in the ZBasic Reference Manual for additional information regarding task management.

Compatibility

This routine is not available in BasicX compatibility.

See Also   ExitTask, RunTask, StatusTask, WaitForInterrupt

Right

Type	Function returning String		

Invocation   Right(str, length)

Parameter	Method	Type	Description
str	ByVal	String	The string from which to extract characters.
length	ByVal	int8/16	The number of characters to extract from the string.

Discussion

This function returns a string consisting of the rightmost characters of the string passed as the first parameter. The maximum number of characters in the returned string is the smaller of 1) the number of characters in the passed string and 2) the value of the second parameter. Internally, the length is interpereted as a 16-bit signed value and negative values are treated as zero.

This function produces the same result as `Mid(str, Len(str) – length + 1 , length)` assuming that the passed string is at least `length` characters long.

Example

```
Dim s as String, s2 as String

s = "Hello, world!"
s2 = Right(s, 6)         ' the result will be "world!"
```

See Also   Left, Mid, Trim

Rnd

Type   Function returning Single

Invocation   Rnd()

Discussion

This function will return a pseudo-random value in the range of 0.0 to 1.0. The first time that `Rnd()` is called after the processor starts up the pseudo-random number generator is initialized with a seed value. The sequence of values returned will be repeatable when starting from the same seed.

You can alter the sequence of returned values in two ways. Firstly, you can set the value of `Register.SeedPRNG`. The next call to `Rnd()` will initialize the pseudo-random number generator with that seed value before returning the first random value. The second way to modify the sequence is to call the `Randomize()` subroutine. Doing so will initialize the pseudo-random number generator with the current value of `Register.RTCTick`. This provides a way to introduce some non-repeatability into the sequence of values returned by `Rnd()`. It is especially effective if the time at which `Randomize()` is called is controlled by some external, unpredictable event like a user pressing a key.

Example

```
Dim i as Integer

' print 10 random values
For i = 1 to 10
      Debug.Print CStr(Rnd())
Next
```


Compatibility

BasicX does not support `Register.seedPRNG`. Instead, it has a system global variable named `seedPRNG`. This built-in variable is also supported in ZBasic for compatibility.

See Also   Randomize

RunTask

Type	Subroutine
Invocation	RunTask(taskStack)
	RunTask()

Parameter	Method	Type	Description
taskStack	ByRef	array of Byte	The stack for a task of interest.

Discussion

Calling this routine alters the normal task rotation regimen by immediately attempting to run the specified task or, if no task stack is explicitly given, the `Main()` task. If the specified task cannot run (because it is sleeping, waiting for InputCapture, etc.) the list of tasks is examined in order beginning with the task immediately following the specified task and the first ready-to-run task that is found will be run.

Because this routine interferes with the normal task rotation it must be used carefully to avoid starving out one or more tasks. If this routine is invoked using an array other than one that is or was being used for a task stack the result is undefined.

See the section on Task Management in the ZBasic Reference Manual for additional information regarding task management.

Compatibility

This routine is not available in BasicX compatibility mode.

See Also	ExitTask, ResumeTask, StatusTask

SearchQueue

Type	Function returning UnsignedInteger		

Invocation   SearchQueue(queue, val)
SearchQueue(queue, dataLen, data)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue of interest.
val	ByVal	Byte	The byte value for which to search.
dataLen	ByVal	Integral	The length of the byte sequence for which to search.
data	ByRef	Any type	The byte sequence for which to search.

Discussion

This function searches the data in a queue looking for the specified byte value (first form) or a sequence of bytes (second form). If the queue is empty or does not contain the byte value/byte sequence, zero is returned. Otherwise, the return value indicates the number of bytes in the queue up to and including the sought byte value/byte sequence.

Note that before any queue operations are performed, the queue data structure must be initialized. See the discussion of `OpenQueue()` for more details.

Examples

```
Dim q(1 to 40) as Byte
Dim data(1 to 4) as Byte
Dim dataLen as UnsignedInteger

' search for a byte value (linefeed)
dataLen = SearchQueue(q, &H0a)

' search for a byte sequence (carriage return, linefeed)
data(1) = &H0d
data(2) = &H0a
dataLen = SearchQueue(q, 2, data)
```

Compatibility

This function is not available on ZX models that are based on the ATmega32 processor (e.g. the ZX-24). Moreover, it is not available in BasicX compatibility mode.

See Also   GetQueue, GetQueueStr, OpenQueue

Semaphore

Type	Function returning Boolean
Invocation	Semaphore(var)

Parameter	Method	Type	Description
var	ByRef	Boolean	A variable used as a semaphore.

Discussion

This function will test the provided variable and if it is already True, the function will return False. Otherwise, if the semaphore variable is False, the call will set it to True and return True. This is referred to in computer science as an "atomic test and set" operation.

A semaphore is a signaling and synchronization mechanism used in multi-tasking systems. The idea is that if two or more tasks each want to use a particular resource they first request ownership of a semaphore. The request mechanism ensures that even if multiple requests occur near the same time, one and only one request will be satisfied. Therefore, the task that is granted the semaphore will have exclusive access to the resource until it has completed its objective. Subsequently, other tasks can request the semaphore and, if they receive it, they can perform their objective. Thus you can see that a particular semaphore can control access to some set of resources that you define. Your system may have multiple semaphores, each controlling access to a set of resources. Note, however, that if multiple semaphores are required to complete an operation the possibility of deadlock exists. This problem will occur if one task obtains one semaphore, another task obtains another semaphore and then both tasks wait for the other semaphore to be available.

In order for this mechanism to be effective, the same semaphore variable must be used by each task for gaining access to a particular set of resources. For this reason, the semaphore variable passed to Semaphore() will almost always be a global variable but it may be public or private as suits your application. The semaphore variable must be initially False, otherwise no Semaphore() request on that semaphore can ever succeed. Also, after a task has successfully gotten the semaphore and has finished using the related resources, the semaphore must be set False again so that a future Semaphore() call will succeed.

Example

```
Dim serSem as Boolean

serSem = False

' wait until we get the semaphore
Do While (Not Semaphore(serSem))
    Call Delay(0.5)
Loop

' now we can use the controlled resources
[add code here]

' finished with the resources, release the semaphore
serSem = False
```

SerialGetByte

Type	Function returning Byte
Invocation	SerialGetByte(inAddr, mask, loopCnt, flags, delayCnt) SerialGetByte(inAddr, mask, loopCnt, flags) SerialGetByte(inAddr, mask, loopCnt)

Parameter	Method	Type	Description
inAddr	ByVal	integral	The address of the input register of a port.
bitMask	ByVal	Byte	A bit mask for the input pin of the port.
loopCnt	ByVal	integral	The loop count to effect the baud rate.
flags	ByVal	Byte	Flag bits controlling the reception process.
delayCnt	ByVal	integral	The delay count to effect the start of sampling.

Discussion

This function reads a byte, transmitted in 8-N-1 serial form (8 data bits, no parity, 1 stop bit) via a pin. It is intended to be called from a routine (e.g. an ISR) that has detected the presence of the start bit.

The pin, assumed to be configured as an input, is specified by giving the address of the input register of the port containing the pin (`inAddr`) and a bit mask containing a single 1 bit corresponding to the bit position in the port (`bitMask`). The `loopCnt` parameter controls the timing of the bit window sampling, specifying a number of 4-cycle loops to delay for each iteration. Note that, in addition to the delay loop, there is an 11-cycle loop overhead that must be taken into account. Consequently, the total number of cycles between each sample is 11 + `loopCnt` * 4.

The `flags` parameter allows control of some aspects of the reception process. The table below describes the meaning of the fields. If the `flags` parameter is omitted, zero is assumed.

Flag Parameter Values

Function	Hex Value	Bit Mask
Non-inverted Logic	&H00	xxxx xxx0
Inverted Logic	&H01	xxxx xxx1

The remaining bits are currently undefined but may be used in the future. For compatibility with new functionality that may be added in the future, the unused bits should always be zero.

If the delayCnt parameter is specified, it gives the number of 4-cycle loops to execute before beginning sampling; this allows for adjusting the beginning of the sampling cycle which should optimally begin in the approximate middle of the start bit. The actual delay from entry to the first sample is approximately 16 + delayCnt * 4. For diagnostic purposes, you may configure a pin to be an output and then specify that pin as the read strobe pin using the compiler directive Option SerialReadStrobe. Immediately before taking the first sample (the start bit) the pin's state will be toggled. Observing this signal transition and its position relative to the center of the start bit can provide information to adjust the delayCnt parameter. It should be noted that the toggling operation adds 3 more cycles before the first sample.

Compatibility

This function is not available in BasicX compatibility mode nor on VM-based devices.

See Also	SerialIn, SerialOut

SerialIn

Type   Function returning Byte

Invocation   SerialIn(pin, baudRate)

Parameter	Method	Type	Description
pin	ByVal	Byte	The pin from which to read the data.
baudRate	ByVal	integral	The baud rate for the serial input.

Discussion

This function reads a byte, transmitted in 8-N-1 serial form (8 data bits, no parity, 1 stop bit), via a pin. The initial state of the pin (expected to be configured as an input) is used to infer logic mode (logic 1 means non-inverted, logic zero means inverted). The function waits for a start bit and then reads eight data bits, sampling the input at the approximate midpoint of the bit window given the specified baud rate. If no start bit is ever detected the function will never return.

While waiting for the start bit, interrupts are not disabled but when the start bit is detected interrupts are disabled for the remainder of the character time, typically about 9.5 bit times. Note that the logic level of the stop bit is not verified. Because relatively precise timing is required for reliable start bit detection and synchronization, this function is best used when few interrupts (preferably none) will occur. In some cases, it may be best to disable interrupts before invoking the function (using, for example, an Atomic block). This strategy, however, has its own shortcomings particularly because it is not known beforehand how long it will be before the start bit arrives.

The theoretical maximum baud rate varies by processor frequency and is expressed as (F_CPU / 19) while the theoretical minimum baud rate is (F_CPU / 262159). Note, particularly, that if the RTC is enabled and the character time (i.e. 10 / baudRate) is greater than approximately 1.5 times the RTC interrupt interval, the RTC may lose time. At 14.7MHz with a 1024Hz RTC interrupt, the minimum standard baud rate that avoids missing RTC interrupts would be 9600.

This function is useful primarily on devices that have no hardware UARTs and/or in cases where you do not want to dedicate a timer for the software UART channels, leaving it free for other purposes.

Example

```
Const pin as Byte = A.0
Dim c as Byte

' configure the pin as an input
Call PutPin(pin, zxInputTriState)

' read a character at 38.4K baud
c = SerialIn(pin, 38400)
```

Compatibility

This function is not available in BasicX compatibility mode nor on VM-based devices.

See Also   SerialGetByte, SerialOut

SerialNumber

Type   Subroutine

Invocation   SerialNumber(serNum)

Parameter	Method	Type	Description
serNum	ByRef	array of Byte	The array to which the serial number will be written.

Discussion

A call to this routine will copy six bytes of serial number information to the provided array. At present, only three of the bytes are defined, representing the version number of the system firmware (for VM mode devices) or the ZBasic library code (for native mode devices). The first byte is the major version number, the second is the minor version number and the third byte is the variant number. The remaining bytes are undefined.

Caution

If the array provided is less than 6 bytes long, subsequent memory will be overwritten, possibly with detrimental results.

Compatibility

The serial number of this implementation may be different than that of BasicX.

SerialOut

Type   Subroutine

Invocation   SerialOut(data, pin, baudRate)

Parameter	Method	Type	Description
data	ByVal	Byte or String	The data to be output.
pin	ByVal	Byte	The pin on which to output the data.
baudRate	ByVal	integral	The baud rate for the serial output.

Discussion

This subroutine outputs the data byte, or the characters of the String, at the specified baud rate on the specified pin. The pin must have been previously configured to be an output in either the high state (for non-inverted data) or the low state (for inverted data). The initial state of the pin determines whether the data will be sent in non-inverted or inverted mode.

The characters transmitted in 8-N-1 format, i.e. 8 data bits, no parity, 1 stop bit. The transmission of each byte is performed with interrupts disabled, comprising an interval of approximately (10 / baudRate) seconds, also known as the "character time". Consequently, higher baud rates are preferable to lower baud rates. The theoretical maximum baud rate varies by processor frequency and is expressed as (F_CPU / 25) while the theoretical minimum baud rate is (F_CPU / 262165). Note, particularly, that if the RTC is enabled and the character time is greater than approximately 1.5 times the RTC interrupt interval, the RTC may lose time. At 14.7MHz with a 1024Hz RTC interrupt, the minimum standard baud rate that avoids missing RTC interrupts would be 9600.

The serial output mechanism is the same as that used when `Option Console` is specified with a pin designator (which pin is available via the compile-time constant `Option.ConsolePin`). See the description of `Option Console` in the ZBasic Language Reference Manual for more Information.

This subroutine is useful primarily on devices that have no hardware UARTs and/or in cases where you do not want to dedicate a timer for the software UART channels, leaving it free for other purposes.

Example

```
Const pin as Byte = A.0
Const str as String = "Hello, world!" & Chr(&H0d) & Chr(&H0a)

' configure the pin as an output, send a string
Call PutPin(pin, zxOutputHigh)  ' non-inverted idle state
Call SerialOut(str, pin, 38400) ' send string chars at 38.4K baud
```

Compatibility

This subroutine is not available in BasicX compatibility mode or on VM-based devices.

See Also   SerialGetByte, SerialIn

SetBits

Type   Subroutine

Invocation   SetBits(target, mask, value)

Parameter	Method	Type	Description
target	ByRef	Byte	The byte to be modified.
mask	ByVal	Byte	The mask indicating which bits to modify.
value	ByVal	Byte	The value of the bits to store.

Discussion

This subroutine allows you to set the value of one or more bits in a byte while leaving others unchanged. Effectively, the result is the same as using the statement below.

```
target = (target And Not mask) Or (value And mask)
```

The `mask` parameter governs which bits will get updated. For each bit of the `mask` parameter that is a 1, the corresponding bit of the `target` will be set to the state of the corresponding bit of the `value` parameter. Bits of the `target` that correspond to zero bits of the `mask` parameter will remain unchanged.

The advantage to using the `SetBits()` subroutine instead of the equivalent statement is twofold. Firstly, it is more efficient, resulting in less code and faster execution time. Secondly, and perhaps more importantly, it performs the action as an atomic operation, i.e. one that is guaranteed, once begun, to complete without an intervening task switch. This characteristic makes `SetBits()` useful for modifying I/O ports and other `Byte` values in a multi-tasking environment.

Example

```
' set the middle 4 bits of Port C to the binary value &B0110
Call SetBits(Register.PortC, &H3C, &H18)
```

Compatibility

This routine is not available in BasicX compatibility mode. Also, it is only supported by ZX firmware later than v1.0.0.

See Also   ToggleBits

SetInterval

Type	Subroutine	

Invocation   SetInterval(interval)

Parameter	Method	Type	Description
interval	ByVal	Single or int16	The interval counter period, in RTC ticks (if an integral value is specified) or seconds (if a `Single` value is given).

Discussion

This routine sets the period of the built-in interval counter. On each RTC tick, the interval counter will be decremented. When it gets to zero, it is reloaded with the specified value and it begins to count down again. Furthermore, if a task is awaiting the interval expiration, it is immediately scheduled for execution (unless a higher priority task requires service). If no task is awaiting the interval expiration, the fact that the interval counter expired is recorded. Subsequently, a task may request a wait on the interval and, depending on the nature of the request, the task may be immediately triggered or it may await the next interval expiration.

Internally the interval period is stored as a 16-bit unsigned integer value. This limits the interval period to a maximum of slightly less than 128 seconds. Of course, longer interval periods may be effectively implemented by maintaining a counter and taking action after the expiration of a number interval periods.

Example

```
Call SetInterval(200)     'about 391 milliseconds
Call SetInterval(10.0)    'about 10 seconds
```

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   WaitForInterval

SetJmp

Type   Function returning Integer

Invocation   SetJmp(jmpbuf)

Parameter	Method	Type	Description
jmpbuf	ByRef	array of Byte	A buffer to hold the return context, see description below.

Discussion

This function, in conjunction with `LongJmp()`, provides a way to circumvent the normal call-return structure and return directly to a distant caller. It is the equivalent of a non-local Goto function and can be used, among other things, to handle exceptions in your programs. The parameter specifies a `Byte` array that will be initialized with context information to allow a direct return from deeply nested calls. The array must be a minimum size (either 6 bytes or 24 bytes for VM mode and native mode, respectively) to hold the context information for unwinding the call stack. You can use the built-in constant `System.JumpBufSize` to ensure that it is the proper size.

On the initial call to `SetJmp()` the return value will always be zero. When control is returned via a call to `LongJmp()`, the return value will be the value supplied as the second parameter to the `LongJmp()` call. Generally, you should choose this value to indicate the nature of the exception condition and in most cases it should be non-zero.

The jump buffer needs to be accessible to the `LongJmp()` caller. Often, this is realized by making it a global or module-level variable. If you want it to be a local variable, you'll have to pass the buffer as a parameter down the call chain. See the section on Exception Handling in the ZBasic Reference Manual for more details.

Caution

If the provided array is less than minimum required size, adjacent memory locations will be modified usually with undesirable results. Your application should not directly modify the contents of the array. Doing so may cause unpredictable behavior.

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   LongJmp

SetQueueX10

Type   Subroutine

Invocation SetQueueX10(chan, forTransmit, phase, queue)

Parameter	Method	Type	Description
chan	ByVal	Byte	The X-10 communication channel of interest.
forTransmit	ByVal	Boolean	True for specifying a transmit queue, false for receive.
phase	ByVal	Byte	The phase to be associated with the queue.
queue	ByRef	array of Byte	The queue of to assign.

Discussion

This subroutine assigns a queue for X-10 operation. In single phase mode, the queues are usually assigned with the call to OpenX10(). In three phase mode, the queues for phase 1 are assigned with the call to OpenX10() and the additional queues needed, if any, are assigned using subsequent calls to this subroutine. If the `forTransmit` parameter is True, the queue will be used for the transmit operation of the specified phase (1, 2 or 3). Otherwise, the queue will be used for the receive operation of the specified phase.

Note that in three phase mode, if a transmit queue is not specified for phase 2, any data sent on phase 1 will also be sent on phase 2. The same is true for phase 3. In contrast, if no receive queue is specified for phase 2, any data received on phase 2 in three phase mode will be ignored; the same is true for phase 3.

Compatibility

This function is only available on native mode devices it is not available in BasicX compatibility mode.

See Also   CloseX10**,** DefineX10**,** OpenX10**,** ResetX10

ShiftIn

Type   Function returning Byte

Invocation   ShiftIn(dataPin, clkPin, bitCnt)

Parameter	Method	Type	Description
dataPin	ByVal	Byte	The pin used to input data.
clkPin	ByVal	Byte	The pin used to output a clocking signal.
bitCnt	ByVal	Byte	The number of bits to read in (1 to 8).

Discussion

This function can be used to input data from a synchronous serial device like a shift register. The pin specified for input will be made an input but the pin specified for the clock signal must already be an output and be at the desired initial logic level.

For each of the number of bits specified, then the clock line will be pulsed by changing its logic level twice. The data line will be sampled approximately 2 CPU clock cycles after the leading edge of the clock pulse. With a 14.7MHz CPU clock, this equates to about 135nS after the leading edge.

The returned value consists of the data bits read with the bit first read in the most significant bit position. This is referred to as MSB first. If fewer than 8 bits are read, the low order bits will be zero.

Resource Usage

This subroutine uses the I/O Timer. If the I/O Timer is already in use, the function returns immediately and the return value is zero. No other use of this resource should be attempted while the shifting is in progress. Interrupts are disabled during the shifting process. However, RTC ticks are accumulated during the shifting process so the RTC should not lose time.

Compatibility

For compatibility with I2C/TWI devices the clock rate is approximately 200kHz with `Register.TimerSpeed1` at its default value of 1. If the value of `Register.TimerSpeed1` is changed, the bit rate will be slower.

See Also   ShiftInEx, ShiftOut, ShiftOutEx

ShiftInEx

Type Function returning UnsignedInteger

Invocation ShiftInEx(dataPin, clkPin, bitCnt, flags)
ShiftInEx(dataPin, clkPin, bitCnt, flags, bitTime)

Parameter	Method	Type	Description
dataPin	ByVal	Byte	The pin used to input data.
clkPin	ByVal	Byte	The pin used to output a clocking signal.
bitCnt	ByVal	Byte	The number of bits to read in (1 to 16).
flags	ByVal	Byte	Flag bits controlling the operation.
bitTime	ByVal	int16	The optional duration of each bit in ticks (see description).

Discussion

This function can be used to input data from a synchronous serial device like a shift register. The pin specified for input will be made an input but the pin specified for the clock signal must already be an output and be at the desired initial logic level. The `flags` parameter controls how the shifting process is performed as described in the table below.

Control Flag Definitions

Function	Hex Value	Bit Mask
MSB first	&H00	xx xx xx x0
LSB first	&H01	xx xx xx x1
Sample the input after the active clock edge	&H00	xx xx xx 0x
Sample the input before the active clock edge	&H02	xx xx xx 1x
Fastest possible bit time	&H00	xx xx x0 xx
Use the provided `bitTime` parameter	&H04	xx xx x1 xx
The active clock edge is the leading clock edge	&H00	xx xx 0x xx
The active clock edge is the trailing clock edge	&H08	xx xx 1x xx

The remaining bits are currently undefined but may be employed in the future.

For each of the number of bits specified, either the state of the data pin will be read and saved first or the clock line will be changed to the opposite state first depending on bit 1 of the `flags` parameter. Finally, the clock line will be returned to the original state thus completing one bit cycle.

If the `flags` parameter so specifies, the `bitTime` parameter value will be used to control the bit rate of the shifting process. The units of the bitTime parameter are, by default, 1 CPU cycle (67.8ns at 14.7MHz). However, `Register.TimerSpeed1` may be changed to adjust the controlling clock speed. If the `bitTime` parameter is not provided or if the value given is zero, the shifting will occur at the maximum rate.

Due to processing overhead the minimum bit time in the controlled speed mode is approximately 60 CPU cycles (4µS at 14.7MHz). Attempting faster bit times in the controlled speed mode will produce undefined results. Without speed control, the bit time is approximately 37 CPU cycles (2.5µS at 14.7MHz). Note that the duty cycle of the clock signal will be closer to 50% in the controlled speed mode. Without speed control, the active clock phase can be as little as 20% of the period.

The returned value consists of the data bits read arranged in MSB or LSB order as specified by the `flags` parameter. If MSB order is specified, the first bit read will be in the most significant bit position of the result. If LSB order is specified, the first bit read will be in the least significant bit position. If fewer than 16 bits are read, the remaining bits will be zero.

For reference purposes, the `ShiftIn()` function is roughly equivalent to `ShiftInEx(dpin, cpin, bitCnt, &H04, 74)`. However, the value read will be in the high order 8 bits of the returned value.

Resource Usage

This subroutine uses the I/O Timer if the `flags` parameter has bit 2 on. If the I/O Timer is already in use, the function returns immediately and the return value is zero. No other use of this resource should be attempted while the shifting is in progress. Interrupts are disabled during the shifting process. However, RTC ticks are accumulated during the shifting process so the RTC should not lose time.

Timing

Bit 3 of the `flags` parameter specifies the active edge of the clock pulse, i.e. whether the data line will be sampled relative to the leading edge or the trailing edge of the clock pulse. Bit 1 of the `flags` parameter controls whether the sampling will be done before or after the active edge. When bit 1 of the `flags` parameter is zero, the data line will be sampled approximately 2 CPU clock cycles after the active edge of the clock pulse. When bit 1 of the `flags` parameter is one, the data line will be sampled approximately 5 CPU clock cycles before the active edge of the clock pulse. With a 14.7MHz CPU clock, these intervals are approximately 135nS and 340nS, respectively.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   ShiftIn, ShiftOut, ShiftOutEx

ShiftOut

Type	Subroutine		
Invocation	ShiftOut(dataPin, clkPin, bitCnt, val)		

Parameter	Method	Type	Description
dataPin	ByVal	Byte	The pin used to output data.
clkPin	ByVal	Byte	The pin used to output a clocking signal.
bitCnt	ByVal	Byte	The number of bits to shift out (1 to 8).
val	ByVal	Byte	The value to shift out.

Discussion

This function can be used to output data to a synchronous serial device like a shift register. The pin specified for output will be made an output but the pin specified for the clock signal must already be an output and be at the desired initial logic level.

For each of the number of bits specified, the data pin will be set to the state of the corresponding bit in the `val` parameter beginning with the most significant bit first. Then the clock line will be pulsed by changing its logic level twice.

Data is shifted out MSB first. If a data width of fewer than 8 data bits is specified, the data must be positioned in the most significant bits of the value and the state of the remaining low order bits in the value is of no consequence.

Resource Usage

This subroutine uses the I/O Timer. If the I/O Timer is already in use, the subroutine returns immediately. No other use of this resource should be attempted while the shifting is in progress. Interrupts are disabled during the shifting process. However, RTC ticks are accumulated during the shifting process so the RTC should not lose time.

Compatibility

For compatibility with I2C/TWI devices the clock rate is approximately 200kHz with `Register.TimerSpeed1` at its default value of 1. If the value of `Register.TimerSpeed1` is changed, the bit rate will be slower.

See Also   ShiftIn, ShiftInEx, ShiftOutEx

ShiftOutEx

Type	Subroutine

Invocation   ShiftOutEx(dataPin, clkPin, bitCnt, val, flags)
ShiftOutEx(dataPin, clkPin, bitCnt, val, flags, bitTime)

Parameter	Method	Type	Description
dataPin	ByVal	Byte	The pin used to output data.
clkPin	ByVal	Byte	The pin used to output a clocking signal.
bitCnt	ByVal	Byte	The number of bits to shift out (1 to 16).
val	ByVal	int8/16	The value to shift out.
flags	ByVal	Byte	Flag bits controlling the operation.
bitTime	ByVal	int16	The optional duration of each bit in ticks (see description).

Discussion

This function can be used to output data to a synchronous serial device like a shift register. The pin specified for output will be made an output but the pin specified for the clock signal must already be an output and be at the desired initial logic level. The `flags` parameter controls how the shifting process is performed as described in the table below.

Control Flag Definitions

Function	Hex Value	Bit Mask
MSB first	&H00	xx xx xx x0
LSB first	&H01	xx xx xx x1
Fastest possible bit time	&H00	xx xx x0 xx
Use the provided `bitTime` parameter	&H04	xx xx x1 xx
Normal data pin output	&H00	xx xx 0x xx
Open drain data pin output	&H08	xx xx 1x xx

The remaining bits are currently undefined but may be employed in the future. For compatibility, the undefined bits should always be zero.

For each of the number of bits specified, the data pin will be set to the state of the corresponding bit in the `val` parameter beginning with the either the most significant bit first or the least significant bit first depending on bit 0 of the `flags` parameter. Then the clock line will be pulsed by changing its logic level twice.

Note that if a data width of fewer than 16 data bits is specified, the bits to be shifted out must be properly aligned in the value provided. If MSB order is specified, the data bits must be positioned in the most significant bits of the value provided. If LSB order is specified, the data bits must be positioned in the least significant bits of the value provided.

If the `flags` parameter so specifies, the `bitTime` parameter value will be used to control the bit rate of the shifting process. The units of the `bitTime` parameter are, by default, 1 CPU cycle (67.8ns at 14.7MHz). However, `Register.TimerSpeed1` may be changed to adjust the controlling clock speed. If the `bitTime` parameter is not provided or if the value given is zero, the shifting will occur at the maximum rate.

Due to processing overhead the minimum bit time in the controlled speed mode is approximately 60 CPU cycles (4µS at 14.7MHz). Attempting faster bit times in the controlled speed mode will produce undefined results. Without speed control, the bit time is approximately 32 CPU cycles (2.2µS at 14.7MHz). Note that the duty cycle of the clock signal will be closer to 50% in the controlled speed mode. Without speed control, the active clock phase can be as little as 20% of the period.

Normally, the data pin will be driven high or low according to the data bits being shifted out. For compatibility with certain data bus interfaces, the `flags` parameter bit 3 can be used to specify that the data pin should be put in high impedance input mode when outputting a one bit and actively pulled to ground for a zero bit. In this mode, an external pullup resistor will need to be used to obtain a voltage level corresponding to a logic one.

For reference purposes, the `ShiftOut()` routine is roughly equivalent to `ShiftOutEx(dpin, cpin, bitCnt, Shl(CInt(val), 8), &H04, 74)`.

Resource Usage

This subroutine uses the I/O Timer if the `flags` parameter has bit 2 on. If the I/O Timer is already in use, the subroutine returns immediately. No other use of this resource should be attempted while the shifting is in progress. Interrupts are disabled during the shifting process. However, RTC ticks are accumulated during the shifting process so the RTC should not lose time.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   ShiftIn, ShiftInEx, ShiftOut

Shl

Type   Function returning the same type as the first parameter.

Invocation   Shl(val, shiftCnt)

Parameter	Method	Type	Description
val	ByVal	integral	The value to be shifted.
shiftCnt	ByVal	int8/16	The number of bit positions to shift (0-16).

Discussion

This function returns the value provided as the first parameter but shifted left the number of bit positions specified by the second parameter. If the `shiftCnt` is zero, the value is returned unchanged. If the `shiftCnt` is greater than or equal to the number of bits in the value provided, the return value will be zero. For signed types, the sign of the result will be the same as that of the provided value.

The type of the return value will be the same as the type of the first parameter.

Example

```
Dim i as Integer, j as Integer

i = 23
j = Shl(i, 5)      ' result will be 736
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   Shr

Shr

Type		Function returning the same type as the first parameter.	

Invocation   Shr(val, shiftCnt)

Parameter	Method	Type	Description
val	ByVal	integral	The value to be shifted.
shiftCnt	ByVal	int8/16	The number of bit positions to shift (0-16).

Discussion

This function returns the value provided as the first parameter but shifted right the number of bit positions specified by the second parameter. If the `shiftCnt` is zero, the value is returned unchanged. If the `shiftCnt` is greater than or equal to the number of bits in the value provided, the return value will be zero. For signed types, the sign of the result will be the same as that of the provided value.

The type of the return value will be the same as the type of the first parameter.

Example

```
Dim i as Integer, j as Integer

i = 23
j = Shr(i, 2)      ' result will be 5
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   Shl

Signum

Type   Function returning the same type as the first parameter.

Invocation   Signum(val)

Parameter	Method	Type	Description
val	ByVal	signed	The value to be tested for positive, zero, negative.

Discussion

This function returns +1, 0 or –1 depending on whether the value provided is positive, zero or negative. The type of the return value will be the same as the type of the parameter value.

Example

```
Dim i as Integer, j as Integer

i = -23
j = Signum(i)      ' result will be -1
```

Compatibility

This function is not available in BasicX compatibility mode.

Sin

Type   Function returning Single

Invocation   Sin(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The angle, in radians, of which the sine will be computed.

Discussion

The return value will be the sine of the supplied value, in the range –1.0 to 1.0.

Example

```
Const pi as Single = 3.14159
Dim val as Single

val = Sin(pi / 2.0)      ' result is approximately 1.0
```

See Also   Asin, DegToRad, RadToDeg

SizeOf

Type	Function returning an integral value		

Invocation   SizeOf(var)

Parameter	Method	Type	Description
var	ByRef	any type	The variable whose size, in bytes, is desired.

Discussion

This function returns the number of bytes constituting the supplied variable.

The primary purpose of this function is to allow writing code that is more easily maintained. For example, instead of hard coding the size value to pass to the `OpenQueue()` subroutine, you can use `SizeOf(queue)` instead. When you change the size of the queue there will be no need to update the `OpenQueue()` calls.

When used with arrays, you may give the array name without any index parameters and `SizeOf()` will return the total number of bytes occupied by the array. Alternately, you may specify constant expressions for all of the array dimensions and `SizeOf()` will return the number of bytes occupied by a single element of the array. This function is not particularly useful with sub-byte types (Bit and Nibble).

The SizeOf() function also allows the argument to name one of the fundamental data types (except `String`). In this case it returns the number of bytes comprising the type. For example, `Sizeof(Integer)` returns the value 2.

Example

```
Dim cnt as Integer
Dim val as Single
Dim ia(1 to 20) as Integer

cnt = SizeOf(val)        ' result is 4
cnt = SizeOf(ia)         ' result is 40
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   SizeOfU

SizeOfU

Type	Function returning UnsignedInteger

Invocation	SizeOfU(var)

Parameter	Method	Type	Description
var	ByRef	any type	The variable whose size, in bytes, is desired.

Discussion

Note: `SizeofU()` is deprecated, in new code you should use `SizeOf()` instead which returns an integral value compatible with all integral types.

This function returns the number of bytes constituting the supplied variable.

The primary purpose of this function is to allow writing code that is more easily maintained. For example, instead of hard coding the size value to pass to the `OpenQueue()` subroutine, you can use `SizeOfU(queue)` instead. When you change the size of the queue there will be no need to update the `OpenQueue()` calls.

When used with arrays, you may give the array name without any index parameters and `SizeOfU()` will return the total number of bytes occupied by the array. Alternately, you may specify constant expressions for all of the array dimensions and `SizeOfU()` will return the number of bytes occupied by a single element of the array. This function is not particularly useful with sub-byte types (Bit and Nibble).

The SizeOfU() function also allows the argument to name one of the fundamental data types (except `String`). In this case it returns the number of bytes comprising the type. For example, `SizeofU(Integer)` returns the value 2.

Example

```
Dim cnt as UnsignedInteger
Dim val as Single
Dim ia(1 to 20) as Integer

cnt = SizeOfU(val)        ' result is 4
cnt = SizeOfU(ia)         ' result is 40
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also	SizeOf

Sleep

Type   Subroutine

Invocation Sleep(time)

Parameter	Method	Type	Description
time	ByVal	Single or int16	The amount of time to delay, in seconds (Single) or ticks (int16)

Discussion

This routine suspends the current task for a period of time up to as long as specified. If the RTC is not enabled in your application, the resolution of the delay period is 1mS. If the RTC is enabled, the resolution is the same as an RTC tick period, i.e. $1/F\_RTC\_TICK$ (typically 1.95mS for ZX devices). The actual sleep time experienced by the calling task depends on what other tasks actually do that may run in the interim. It is possible that the task will be suspended indefinitely depending on what another task might do.

Note that if the current task is locked, this call will unlock it.

There is a subtle difference between Delay() and Sleep() when the RTC is enabled and the arguments are non-zero. For Delay() the specified time is the minimum amount of delay that the task will experience assuming that no other task is ready to run and the actual delay could be up to 1 unit longer than the specified delay. For Sleep(), the specified time is the maximum amount of delay that the task will experience assuming that no other task is ready to run and the actual delay could be up to 1 unit less than the specified delay.

It is important to note that internally, the sleep function utilizes an integral tick value. If the supplied parameter is a Single value it is converted to the equivalent integral number of ticks (if the value is a known constant at compile time, otherwise at run time). Consequently, if a Single value is specified that is less than the equivalent of one tick the sleep time will be zero ticks.

Example

```
Do
     Call PutPin(Pin.RedLED, 0)
     Call Sleep(0.5)          ' a half-second delay
     Call PutPin(Pin.RedLED, 1)
     Call Sleep(256)          ' a half-second delay
Loop
```

This loop causes the red LED to turn on and off alternately for a half second each.

Compatibility

The BasicX documentation specifically indicates that Sleep() will unlock a locked task. However, tests indicate that this only happens if the parameter to Sleep() is non-zero. This implementation unlocks a task on any Sleep() call.

See Also   Delay**,** DelayUntilClockTick**,** Pause**,** WaitForInterval, Register.RTCStopWatch

SngClass

Type   Function returning Byte

Invocation   SngClass(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value of which to determine the floating point classification.

Discussion

The IEEE 754 standard floating point format used by ZBasic specifies a set of classifications for floating point values. This function returns a numeric value indicating the class to which the passed `Single` value belongs. The table below enumerates the return values and describes the meaning of each.

Floating Point Value Classes

Class	Value	Description
ClassNormal	1	Normalized - This class represents "normal" floating point values such as 1.537 but does not include 0.0.
ClassZero	2	Zero - This class represents the zero value (positive and negative).
ClassInfinity	3	Infinity - This class represents positive and negative infinity. Dividing a positive value by zero results in positive infinity.
ClassDenormal	4	Denormalized - This class represents an internal form known as denormalized values. Such values should never be generated as a result of a floating point operation. However if you copy some random bytes into a floating point variable the result may be a denormalized value.
ClassNaN	5	NaN - This class represents values that are "Not A Number". Taking the square root of a negative value or the logarithm of zero results in a NaN.

The names in the first column are available as built-in constants. Except for ClassNaN, the return value may include the flag &H80 to indicate a negative value. For example, `SngClass(-1.0)` returns the value &H81 to indicate a negative ClassNormal value. The built-in constant representing the negative flag is ClassNegative. The built-in constant ClassMask may be used to remove the negative flag from the return value, e.g. `SngClass(fval) And ClassMask`.

Examples

```
Dim class as Byte

class = SngClass(1.0)                 ' result is 1
class = SngClass(-1.0)                ' result is &H81
class = SngClass(-1.0) And ClassMask  ' result is 1
class = SngClass(Sqr(-1.0))           ' result is 5
class = SngClass(1.0 / 0.0)           ' result is 3
```

Compatibility

This function is not available in BasicX compatibility mode.

Span

Type	Function returning an integral value
Invocation	Span(array) or
	Span(array, dimension)

Parameter	Method	Type	Description
array	ByRef	any array	The array about which the dimension information is desired.
dimension	ByVal	int16	The dimension of interest. See the description for more details.

Discussion

This function returns the number of elements in a dimension of the specified array. There are two forms. The first requires only the array to be specified. In this case, the number of elements of the first dimension of the array is returned. The second form specifies a dimension number (which must be a constant value), the valid range of which is 1 to the number of dimensions of the array. The array may be located in RAM, Program Memory or Persistent Memory.

Note that the use of this function instead of hard-coding values makes your code easier to maintain because it automatically adapts if the definition of an array changes.

Example

```
Dim ba(1 to 20) as Byte
Dim ma(3 to 5, -6 to 7) as Byte
Dim i as Integer

i = Span(ba)            ' the result is 20
i = Span(ma)            ' the result is 3
i = Span(ma, 1)         ' the result is 3
i = Span(ma, 2)         ' the result is 14
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   LBound, UBound

SPICmd

Type   Subroutine

Invocation   SPICmd(channel, writeCnt, writeData, readCnt, readData)

Parameter	Method	Type	Description
channel	ByVal	Byte	The SPI channel number (1-4).
writeCnt	ByVal	integral	The number of bytes to write (0 – 65535).
writeData	ByRef	any type	The variable containing the data to write to the device.
readCnt	ByVal	integral	The number of bytes to read (0 – 65535).
readData	ByRef	any type	The variable in which to place the data read from the device.

Discussion

This routine allows you to send and/or receive data from a device using the SPI protocol. The specified channel must have been previously opened with a call to `OpenSPI()`. If the channel has not been opened, the results are undefined. If a hardware SPI controller is being used, the target device must be connected to the controller's SPI bus (on a 24-pin ZX device, the holes on the end of the device between pins 1 and 24). Otherwise, the pins most recently set by DefineSPI are used for the SPI clock and data.

If both `writeCnt` and `readCnt` are zero the routine returns immediately without doing anything. You may specify the value 0 for either `writeData` or `readData` if no data is being provided. If the value of `readCnt` exceeds the size of the `readData` variable, the additional bytes will be written to subsequent memory locations, possibly with undesirable results.

The execution of the SPI command occurs in four phases:
> The chip select is asserted by setting the previously specified pin to the active level. The active level (typically logic zero) is specified by bit 6 in the `flags` parameter passed to `OpenSPI()`.
> If the `writeCnt` parameter is non-zero, the data bytes at `writeData` are written sequentially to the SPI interface. The data returned by the SPI device during this phase is discarded.
> If the `readCnt` parameter is non-zero, the existing data beginning at `readData` are written to the SPI device and the returned bytes are stored sequentially in the specified variable. That is, the byte at `readData(1)` is sent to the device and the byte that the device sends back is stored at `readData(1)`. The same occurs for `readData(2)`, etc.
> Finally, the chip select is deasserted by setting the previously specified pin to the inactive level.

Whether you use `writeData` or `readData` or both depends on the particulars of the device you're using. In some cases, you'll need to populate readData and in other cases not. Careful study of the datasheet of the target device will be required to determine how `SPICmd()` can be used to communicate with it.

For an SPI channel that is opened with an non-zero `rxDelay` parameter specified (see `OpenSPI()`), a delay is implemented prior to each SPI cycle for which the data read is placed in the `readData` buffer, i.e., the third phase described above. The delay value specified is interpreted as the number of cycles of the SPI clock frequency (but ignoring the Double Speed configuration bit). Of course, during the delay time the SPI clock signal (SCK) will be idle. This delay is useful when communicating with slave devices that must compute data values to return, for example, a ZX-24n operating in SPI slave mode.

Example

```
Dim odata(1 to 2) as Byte, idata(1 to 10) as Byte
Call OpenSPI(1, 0, 12)
odata(1) = &H06
odata(2) = &H00
Call SPICmd(1, 2, odata, 10, idata)
```

In this example `idata` is not initialized before calling `SPICmd()`. If your SPI device needs specific data written to it during the read phase, `idata` would need to be initialized before the call.

Compatibility

The use of a zero value to indicate that no data buffer is being supplied is not supported in BasicX compatibility mode. Also, in BasicX compatibility mode, both `writeCnt` and `readCnt` are Byte values and, thus, limited to a maximum of 255.

See Also   CloseSPI, OpenSPI

SPIGetByte

Type	Function returning Byte
Invocation	SPIGetByte(channel, writeData) or SPIGetByte(channel)

Parameter	Method	Type	Description
channel	ByVal	Byte	The SPI channel number (1-4).
writeData	ByVal	Byte	The data value to send to the slave for each byte received.

Discussion

This function sends a byte of data to an SPI slave and returns the value sent back by the SPI slave. For the second form, the data byte sent while receiving is zero. If the channel number is invalid or the channel is not properly prepared, the return value is zero. Note that this is a low level function that must be used in concert with other low level routines to effect an SPI bus transaction.

Example

```
Dim b as Byte
Call OpenSPI(1, 0, 12)
Call SPIStart(1, &H03)
b = SPIGetByte(1, &Hff)
```

Compatibility

This routine is not available in BasicX compatibility mode nor it is available for VM devices.

See Also   CloseSPI, OpenSPI, SPIPutByte, SPIGetData, SPIPutData, SPIStart, SPIStop

SPIPutByte

Type   Function returning Byte

Invocation   SPIPutByte(channel, data)

Parameter	Method	Type	Description
channel	ByVal	Byte	The SPI channel number (1-4).
data	ByVal	Byte	The variable containing the data to write to the slave.

Discussion

This function sends a byte of data to an SPI slave and returns the value sent back by the SPI slave. If the channel number is invalid or the channel is not properly prepared, the return value is zero. Note that this is a low level function that must be used in concert with other low level routines to effect an SPI bus transaction.

Example

```
Dim b as Byte
Call OpenSPI(1, 0, 12)
Call SPIStart(1, &H03)
b = SPIPutByte(1, &H23)
```

Compatibility

This routine is not available in BasicX compatibility mode nor it is available for VM devices.

See Also   CloseSPI, OpenSPI, SPIGetByte, SPIGetData, SPIPutData, SPIStart, SPIStop

SPIGetData

Type	Function returning UnsignedInteger		

Invocation   SPIGetData(channel, readCnt, readData, writeData)
                    SPIGetData(channel, readCnt, readData)

Parameter	Method	Type	Description
channel	ByVal	Byte	The SPI channel number (1-4).
readCnt	ByVal	integral	The number of bytes to read (0 – 65535).
readData	ByRef	any type	The variable in which to place the data read from the slave device.
writeData	ByVal	Byte	The data value to send to the slave for each byte received.

Discussion

This function sends a data byte to an SPI slave the specified number of times and stores the byte returned by the slave in response in successive bytes of the variable provided (typically a Byte array). For the second form, the data byte sent while receiving is zero. The value returned is equal to the number of bytes placed in the variable. If the channel number is invalid or the channel is not properly prepared, the return value is zero. Note that this is a low level function that must be used in concert with other low level routines to effect an SPI bus transaction.

Example

```
Dim idata(1 to 10) as Byte
Call OpenSPI(1, 0, 12)
Call SPIStart(1, &H03)
Call SPIGetData(1, 10, idata)
```

Compatibility

This routine is not available in BasicX compatibility mode nor it is available for VM devices.

See Also   CloseSPI, OpenSPI, SPIGetByte, SPIPutByte, SPIPutData, SPIStart, SPIStop

SPIPutData

Type	Function returning UnsignedInteger		

Invocation   SPIPutData(channel, writeCnt, writeData)

Parameter	Method	Type	Description
channel	ByVal	Byte	The SPI channel number (1-4).
writeCnt	ByVal	integral	The number of bytes to write (0 – 65535).
writeData	ByRef	any type	The variable containing the data to write to the slave device.

Discussion

This function sends the specified number of bytes to an SPI slave from the variable provided (typically a Byte array) discarding the data returned by the slave in response to each byte sent. The value returned is equal to the number of bytes sent to the slave. If the channel number is invalid or the channel is not properly prepared, the return value is zero. Note that this is a low level function that must be used in concert with other low level routines to effect an SPI bus transaction.

Example

```
Dim odata(1 to 2) as Byte
Call OpenSPI(1, 0, 12)
Call SPIStart(1, &H03)
odata(1) = &H06
odata(2) = &H00
Call SPICmd(1, 2, odata)
```

Compatibility

This routine is not available in BasicX compatibility mode nor it is available for VM devices.

See Also   CloseSPI, OpenSPI, SPIGetByte, SPIGetData, SPIPutByte, SPIStart, SPIStop

SPIStart

Type   Subroutine

Invocation   SPIStart(channel, flags)

Parameter	Method	Type	Description
channel	ByVal	Byte	The SPI channel number (1-4).
flags	ByVal	Byte	Flag bits indicating actions to take (see discussion).

Discussion

If the specified channel is valid and is open, this subroutine modifies the state of the SPI interface according to the bits that are on in the *flags* parameter as described below. If multiple bits are on in the *flags* parameter they are processed in order from least significant to most significant.

Flag Parameter Action Bits

Hex Value	Binary Value	Description
&H01	xx xx xx x1	Initialize the SPI controller associated with the channel (ignored for a software channel).
&H02	xx xx xx 1x	Deassert the chip select pin associated with the channel.
&H04	xx xx xx1 xx	Assert the chip select pin associated with the channel.

For compatiblility with future enhancements the unused bits in the *flags* parameter should always be zero. Note that this is a low level routine that must be used in concert with other low level routines to effect an SPI bus transaction.

If this routine is called with bits 1 and 2 both on (i.e. &H06) when the chip select is already asserted, the result will be that the chip select is deasserted for about 150 CPU cycles (~10uS at 14.7MHz) and then reasserted. This idiom is useful for terminating one SPI cycle and immediately beginning another.

Example

```
Call OpenSPI(1, 0, 12)
Call SPIStart(1, &H05)
```

Compatibility

This routine is not available in BasicX compatibility mode nor it is available for VM devices.

See Also   CloseSPI, OpenSPI, SPIGetByte, SPIPutByte, SPIGetData, SPIPutData, SPIStop

SPIStop

Type   Subroutine

Invocation   SPIStop(channel, flags)

Parameter	Method	Type	Description
channel	ByVal	Byte	The SPI channel number (1-4).
flags	ByVal	Byte	The number of bytes to write (0 – 65535).

Discussion

If the specified channel is valid and is open, this subroutine modifies the state of the SPI interface according to the bits that are on in the *flags* parameter as described below. If multiple bits are on in the *flags* parameter they are processed in order from most significant to least significant.

Flag Parameter Action Bits

Hex Value	Binary Value	Description
&H01	xx xx xx x1	Deinitialize the SPI controller associated with the channel (ignored for a software channel).
&H02	xx xx xx 1x	Deassert the chip select pin associated with the channel.

For compatiblility with future enhancements the unused bits in the *flags* parameter should always be zero. Note that this is a low level routine that must be used in concert with other low level routines to effect an SPI bus transaction.

Example

```
Call OpenSPI(1, 0, 12)
Call SPIStop(1, &H03)
```

In this example `idata` is not initialized before calling `SPICmd()`. If your SPI device needs specific data written to it during the read phase, `idata` would need to be initialized before the call.

Compatibility

This routine is not available in BasicX compatibility mode nor it is available for VM devices.

See Also   CloseSPI, OpenSPI, SPIGetByte, SPIPutByte, SPIGetData, SPIPutData, SPIStart

Sqr

Type   Function returning Single

Invocation   Sqr(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The value of which the square root will be computed.

Discussion

The return value will be the square root of the supplied value. Note that the `Sqr()` function will return NaN if the argument is negative.

Example

```
Dim val as Single

val = Sqr(2.0)    ' result is approximately 1.414
```

StackCheck

Type   Subroutine

Invocation   StackCheck(enable)

Parameter	Method	Type	Description
enable	ByVal	Boolean	The enable/disable state desired.

Discussion

This subroutine enables or disables stack checking. See the section on Run Time Stack Checking in the ZBasic Reference Manual for more information.

Example

```
Call StackCheck(true)
```

Compatibility

This routine is not available in BasicX compatibility mode nor is it available for native mode devices.

StatusCom

Type   Function returning Byte

Invocation   StatusCom(chan)

Parameter	Method	Type	Description
chan	ByVal	Byte	The serial channel of interest.

Discussion

This function returns a set of flag bits that indicate the status of the specified serial channel. The meaning of each of the flag bits is shown in the table below.

Serial Channel Status Bit Values

Value	Meaning
&H01	The channel number is valid but may or may not be open.
&H02	The channel is open.
&H04	The channel has data yet to be transmitted.
&H08	The channel is a software UART channel.
&H10	The channel's receive flow control pin is in the inactive state.
&H20	The channel's transmit flow control pin is in the inactive state.

The remaining bits are currently undefined but may convey additional information in the future. It is strongly advised that you apply an AND mask to the returned value before comparing it to a fixed value. Doing so will prevent the future addition of bits from affecting your existing code.

Note that the &H04 bit will be set if there are any unsent characters in the transmit queue associated with the channel as well as if a character is currently being transmitted.

Example

```
' wait until all characters are transmitted
Do While CBool(StatusCom(chan) And &H04)
Loop
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   CloseCom, ComChannels, ControlCom, DefineCom, OpenCom

StatusQueue

Type   Function returning Boolean

Invocation   StatusQueue(queue)

Parameter	Method	Type	Description
queue	ByRef	array of Byte	The queue of interest.

Discussion

This function returns `True` if there data bytes in the queue, otherwise `False`.

Note that before any queue operations are performed, the queue data structure must be initialized. See the discussion of `OpenQueue()` for more details.

Compatibility

BasicX allows any type for the first parameter. The ZBasic implementation requires that it be an array of `Byte`.

See Also   GetQueueCount, OpenQueue

StatusTask

Type   Function returning Byte

Invocation   StatusTask(taskStack)
                  StatusTask()

Parameter	Method	Type	Description
taskStack	ByRef	array of Byte	The stack for a task of interest.

Discussion

This function returns a value indicating the status of the task associated with the given task stack. If no task stack is explicitly given, the task stack for the `Main()` routine is assumed. The return values and their respective meanings are shown in the table below.

Task Status Values

Constant	Value	Meaning
TaskReady	0	The task is running or ready to run.
TaskSleeping	1	The task is sleeping.
TaskWaitInputCapture	2	The task is waiting for InputCapture() to complete.
TaskWaitInt0	3	The task is awaiting Interrupt 0.
TaskWaitInt1	4	The task is awaiting Interrupt 1.
TaskWaitInt2	5	The task is awaiting Interrupt 2.
TaskWaitInterval	6	The task is waiting for the interval counter to expire.
TaskWaitAnalogCompare	7	The task is waiting for an analog comparator event.
TaskWaitPinChange0	8	The task is waiting for a pin change event 0.
TaskWaitPinChange1	9	The task is waiting for a pin change event 1.
TaskWaitPinChange2	10	The task is waiting for a pin change event 2.
TaskWaitPinChange3	11	The task is waiting for a pin change event 3.
TaskWaitOutputCapture	12	The task is waiting for OutputCapture() to complete.
TaskWaitInt3	13	The task is awaiting Interrupt 3.
TaskWaitInt4	14	The task is awaiting Interrupt 4.
TaskWaitInt5	15	The task is awaiting Interrupt 5.
TaskWaitInt6	16	The task is awaiting Interrupt 6.
TaskWaitInt7	17	The task is awaiting Interrupt 7.
TaskWaitPinChangeA0	18	The task is waiting for a pin change event, port A.
TaskWaitPinChangeA1	19	The task is waiting for a pin change event, port A.
TaskWaitPinChangeB0	20	The task is waiting for a pin change event, port B.
TaskWaitPinChangeB1	21	The task is waiting for a pin change event, port B.
TaskWaitPinChangeC0	22	The task is waiting for a pin change event, port C.
TaskWaitPinChangeC1	23	The task is waiting for a pin change event, port C.
TaskWaitPinChangeD0	24	The task is waiting for a pin change event, port D.
TaskWaitPinChangeD1	25	The task is waiting for a pin change event, port D.
TaskWaitPinChangeE0	26	The task is waiting for a pin change event, port E.
TaskWaitPinChangeE1	27	The task is waiting for a pin change event, port E.
TaskWaitPinChangeF0	28	The task is waiting for a pin change event, port F.
TaskWaitPinChangeF1	29	The task is waiting for a pin change event, port F.
TaskWaitPinChangeH0	30	The task is waiting for a pin change event, port H.
TaskWaitPinChangeH1	31	The task is waiting for a pin change event, port H.
TaskWaitPinChangeJ0	32	The task is waiting for a pin change event, port J.
TaskWaitPinChangeJ1	33	The task is waiting for a pin change event, port J.
TaskWaitPinChangeK0	34	The task is waiting for a pin change event, port K.
TaskWaitPinChangeK1	35	The task is waiting for a pin change event, port K.
TaskWaitPinChangeQ0	36	The task is waiting for a pin change event, port Q.

TaskWaitPinChangeQ1	37	The task is waiting for a pin change event, port Q.
TaskWaitAnalogCompA0	38	The task is waiting for an analog comparator A event.
TaskWaitAnalogCompA1	39	The task is waiting for an analog comparator A event.
TaskWaitAnalogCompAW	40	The task is waiting for an analog comparator A window event.
TaskWaitAnalogCompB0	41	The task is waiting for an analog comparator B event.
TaskWaitAnalogCompB1	42	The task is waiting for an analog comparator B event.
TaskWaitAnalogCompBW	43	The task is waiting for an analog comparator B window event.
TaskHalting	254	The task is in the process of terminating.
TaskHalted	255	The task has terminated.

The shaded entries in the table above are specific to ZBasic devices based on ATTiny and ATmega processors. The values 18-43 are specific to ZBasic devices based on ATxmega processors. See the Resource Usage sub-section Pin Change Interrupts for information about the mapping of ports to pin change interrupt events.

If this function is invoked using an array other than one that is or was being used for a task stack the result is undefined. See the section on Task Management in the ZBasic Reference Manual for additional information regarding task management.

See Also   ExitTask, ResumeTask, RunTask, TaskIsValid, WaitForInterrupt

StatusX10

Type   Function returning Byte

Invocation   Status X10(chan)

Parameter	Method	Type	Description
chan	ByVal	Byte	The X-10 communication channel of interest.

Discussion

This function returns a set of flag bits that indicate the status of the specified X-10 channel. The bits and their meanings are shown in the table below. The return value may comprise zero or more of the status bits.

X-10 Channel Status Bit Values

Value	Meaning	Cleared by ResetX10
&H01	The channel number is valid but may or may not be open.	No
&H02	The channel is open.	No
&H04	The channel has data yet to be transmitted.	No
&H10	An end-of-command condition was detected during reception (phase 3).	Yes
&H20	An end-of-command condition was detected during reception (phase 2).	Yes
&H40	An end-of-command condition was detected during reception (phase 1).	Yes
&H80	A collision was detected during transmission.	Yes

The remaining bits are currently undefined but may convey additional information in the future. Some of the status bits represent state information for the channel that can be cleared by calling ResetX10(); see the third column of the table above.

Compatibility

This function is not available on ZX models that are based on the ATmega32 processor (e.g. the ZX-24). Moreover, it is not available in BasicX compatibility mode.

See Also   CloseX10, DefineX10, OpenX10, ResetX10

StrAddress

Type	Function returning UnsignedInteger		

Invocation   StrAddress(str)

Parameter	Method	Type	Description
str	ByVal	String	The string variable whose string address is desired.

Discussion

This function returns the memory address of the first character of a string stored in a string variable. Note that for dynamically allocated strings, the string address will be zero if the string is empty. Note also that the returned address may refer to RAM, Program Memory or Persistent memory. The function `StrType()` can be used to determine which address space contains the string's characters. For statically allocated strings, the string address will always be non-zero even if the string is empty.

See the section on Strings in the ZBasic Reference Manual for more details about dynamically vs. statically allocated strings.

Example

```
Dim str as String
Dim addr as UnsignedInteger
Dim b as Byte

str = "Hello, world!"
addr = StrAddress(str)
b = RamPeek(addr)          ' result will be 72, the letter H
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   StrType

StrCompare

Type	Function returning Integer		

Invocation	StrCompare(str1, str2)		
	StrCompare(str1, str2, ignoreCase)		

Parameter	Method	Type	Description
str1	ByVal	String	The first string to compare.
str2	ByVal	String	The second string to compare.
ignoreCase	ByVal	Boolean	A flag controlling whether alphabetic case is significant.

Discussion

This function returns a value indicating the "sort order" of the two strings. If the returned value is negative, the first string precedes the second in sort order, i.e. the first string would appear before the second in a list sorted alphabetically. If the returned value is zero, the strings have the same sort order and if it is greater than zero, the second string has a higher sort order. If the optional `ignoreCase` parameter is given, the comparison is done either observing or ignoring differences in alphabetic case depending on the value of the parameter. For the purposes of this parameter only the characters A-Z and a-z (&H41 to &H5a and &H61 to &H7a) are considered to be alphabetic. If the `ignoreCase` parameter is omitted, the comparison is performed observing case differences.

Example

```
Dim str1 as String
Dim str2 as String

If (StrCompare(str1, str2, true) = 0) Then
     Debug.Print "The strings match"
End If
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also StrFind

StrFind

Type	Function returning Byte

Invocation	StrFind(inStr, findStr)
	StrFind(inStr, findStr, startIdx)
	StrFind(inStr, findStr, startIdx, ignoreCase)

Parameter	Method	Type	Description
inStr	ByVal	String	The string to be searched.
findStr	ByVal	String	The string being sought.
startIdx	ByVal	integral	The index of inStr at which to begin the search.
ignoreCase	ByVal	Boolean	A flag controlling whether alphabetic case is significant.

Discussion

This function attempts to find the first occurrence of the `findStr` string within the `inStr` string. If it is found, the return value gives the 1-based index where the sought string was found within the searched string. If the sought string is not found, zero is returned. If the optional `startIdx` parameter is not given, the search begins at the first character of the searched string, equivalent to specifying 1 for `startIdx`. If the optional `ignoreCase` parameter is not given, the search is performed observing alphabetic case differences, otherwise alphabetic case differences are significant or not depending on the value specified for `ignoreCase`. For the purposes of this parameter only the characters A-Z and a-z (&H41 to &H5a and &H61 to &H7a) are considered to be alphabetic.

Searching for a zero length string will always be successful and the return value will be the specified or implied starting index. Searching for a non-zero length string within a zero length string will always fail, returning 0.

Examples

```
Dim idx as Byte

idx = StrFind("haystack", "needle")              ' returns 0
idx = StrFind("haystack with needle", "needle") ' returns 15
idx = StrFind("foo bar foo", "foo", 2)          ' returns 9
idx = StrFind("foo bar foo", "", 2)             ' returns 2
idx = StrFind("foo bar FOO", "FOO")             ' returns 9
idx = StrFind("foo bar FOO", "FOO", 1, true)    ' returns 1
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also	MemFind, ProgMemFind, StrCompare

StrReplace

Type   Function returning String

Invocation   StrReplace(str, findStr, replStr)
StrReplace(str, findStr, replStr, startIdx)
StrReplace(str, findStr, replStr, startIdx, replCount)
StrReplace(str, findStr, replStr, startIdx, replCount, ignoreCase)

Parameter	Method	Type	Description
str	ByVal	String	The subject string in which to perform replacement.
findStr	ByVal	String	The sought string.
replStr	ByVal	String	The replacement string.
startIdx	ByVal	integral	The index of 'str' at which to begin the replacement.
replCount	ByVal	integral	The number of replacements to perform.
ignoreCase	ByVal	Boolean	A flag controlling whether alphabetic case is significant.

Discussion

This routine produces a new string by replacing occurrences of the sought string with the replacement string in the subject string. If the optional startIdx parameter is not given, the search begins at the first character of the subject string, equivalent to specifying 1 for startIdx. If the optional replCount parameter is not given, all occurrences of the sought string will be replaced. If the optional ignoreCase parameter is not given, the search is performed observing alphabetic case differences, otherwise alphabetic case differences are significant or not depending on the value specified for ignoreCase. For the purposes of this parameter, only the characters A-Z and a-z (&H41 to &H5a and &H61 to &H7a) are considered to be alphabetic.

If the subject string contains no occurrences of the sought string, or if the sought string is zero length, or if the replacement count is zero, the returned string will be identical to the subject string. The replacement count and the start index are treated internally as signed 16-bit values. If the value of the start index is less than 1, a starting index of 1 is assumed.

Compatibility

This function is not available in BasicX compatibility mode.

StrType

Type   Function returning Byte

Invocation   StrType(str)

Parameter	Method	Type	Description
str	ByVal	String	The string variable whose string type is desired.

Discussion

This function returns a value indicating the nature of a string variable. The values returned have the meaning shown in the table below.

Type	Meaning
`&H00`	The string is a standard statically allocated string or a bounded string. The value returned by `StrAddress()` is a RAM address and can be read using `RamPeek()` or `MemCopy()`.
`&He0`	The string is dynamically allocated. The value returned by `StrAddress()` is a RAM address (which may be zero) and can be read using `RamPeek()` or `MemCopy()`.
`&He2`	The string is in Program Memory. The value returned by `StrAddress()` is a Program Memory address and can be read using `GetProgMem()`.
`&He3`	The string is in Persistent Memory. The value returned by `StrAddress()` is a Persistent Memory address and can be read using `GetPersistent()`.
`&He4`	The string is in RAM. The value returned by `StrAddress()` is a RAM address (which may be zero) and can be read using `RamPeek()` or `MemCopy()`.
`&He5`	The string is in RAM and is limited to 1 or 2 characters. The value returned by `StrAddress()` is a RAM address and can be read using `RamPeek()` or `MemCopy()`.
`&He6`	The string is in RAM. The value returned by `StrAddress()` is a RAM address and can be read using `RamPeek()` or `MemCopy()`. This special string type is used for native-mode code to pass a bounded string or fixed-length string to a subroutine/function ByVal.
`&Hff`	The string is a statically allocated fixed-length string. The value returned by `StrAddress()` is a RAM address and the data can be read using `RamPeek()` or `MemCopy()`.

See the section on strings in the ZBasic Reference Manual for more details about dynamically vs. statically allocated strings.

Compatibility

This function is not available in BasicX compatibility mode.

See Also   StrAddress

System.Alloc

Type	Function returning UnsignedInteger

Invocation	System.Alloc(numBytes)

Parameter	Method	Type	Description
numBytes	ByVal	integral	The size of the requested allocation.

Discussion

This function allocates a block of memory from the heap of the specified size and returns the address of the first byte of the block. If a block of the specified size cannot be allocated, zero is returned. The block can be returned to the heap using the subroutine `System.Free()`.

This function and the block of memory it returns must be used with great care. If your program fails to deallocate the block using `System.Free()` when it is no longer needed, the heap may eventually be exhausted. Since space for strings is also allocated from the heap, exhaustion may cause string operations to fail. Moreover, if your program writes to memory outside of the bounds of the block, the heap data structures may be corrupted. This may cause future heap allocation requests to fail.

For native mode devices (e.g. the ZX-24n) a heap allocation may fail if the heap size is set too small compared to the needs of your application. To aid in determining a sufficient heap size the System Library function System.HeapHeadRoom() may be used to discover the amount of space in the heap that has not yet been used at the time of the call.

Example

```
Dim addr as UnsignedInteger
addr = System.Alloc(50)
[other code here that uses the allocated block]
Call System.Free(addr)
addr = 0
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also	System.Free

System.DeviceID

Type   Subroutine

Invocation   System.DeviceID(buffer)

Parameter	Method	Type	Description
buffer	ByRef	array of Byte	The array to which the identification characters will be written.

Discussion

A call to this routine will copy up to 10 bytes to the buffer provided. The data copied to the buffer comprise characters of a string that identify the ZBasic device on which the progam is executing. The last byte of the identification is followed by a zero byte that serves to mark the end of the identification characters. The example below illustrates how the data can be used to create a string.

Although this subroutine is primarily intended for manufacturing test purposes, it may be useful for other purposes as well.

Caution

If the array provided is less than 10 bytes long, subsequent memory may be overwritten, possibly with detrimental results.

Example

```
Dim buf(1 to 10) as Byte
Dim idStr as String
Dim idx as Byte

Call System.DeviceID(buf)
idStr = MakeString(buf.DataAddress, SizeOf(buf))
idx = StrFind(idStr, Chr(0))
If (idx <> 0) Then
    idStr = Left(idStr, idx - 1)
End If
Debug.Print idStr  ' Displays "ZX24" on a ZX-24
```

Compatibility

This routine is not available in BasicX compatibility mode and it is supported only on ZX devices.

System.Free

Type	Subroutine

Invocation   System.Free(addr)

Parameter	Method	Type	Description
addr	ByVal	UnsignedInteger	The address of the block to free.

Discussion

This subroutine returns a block of allocated memory to the heap so that it may be later re-used. The `addr` parameter must be the value returned by an earlier call to `System.Alloc()` that has not yet been freed. Invoking this subroutine with `addr` equal to 0 is a special case that is benign.

This function and its companion, `System.Alloc()`, must be used with great care. If `System.Free()` is called with a non-zero value that is not one returned by `System.Alloc()` or a value that has already been freed, the heap management data structures will almost certainly be corrupted and future allocations will likely fail. It is a good practice to set an address to zero after it has been freed as illustrated in the example below.

Example

```
Dim addr as UnsignedInteger
addr = System.Alloc(50)
[other code here that uses the allocated block]
Call System.Free(addr)
addr = 0
```

Compatibility

This routine is not available in BasicX compatibility mode.

See Also   System.Alloc

System.HeapHeadRoom

Type   Function returning UnsignedInteger

Invocation   System.HeapHeadRoom()

Discussion

This function determines the amount of space in the string heap that has never been used irrespective of the current end-of-heap position. The primary use for it is to determine the amount of heap space used by an application in order to balance the requirements of the heap and the various task stacks.

Compatibility

For VM mode devices, this function will always return `&HFFFF` unless you have specified a heap limit value, directly or indirectly. See the `Option MainTaskStackSize`, `Option HeapSize`, and `Option HeapLimit` directives for more information.

See Also   System.TaskHeadRoom

System.HeapSize

Type   Function returning UnsignedInteger

Invocation   System.HeapSize()

Discussion

This function determines the amount of space reserved for the string heap. This value may be of use in special circumstances such as allocating extra buffers or dynamic task stacks.

See Also   System.HeapHeadRoom

System.TaskHeadRoom

Type   Function returning UnsignedInteger

Invocation   System.TaskHeadRoom(taskStack)
System.TaskHeadRoom()

Parameter	Method	Type	Description
taskStack	ByRef	array of Byte	The stack for a task of interest.

Discussion

This function determines the amount of space in task stack of the specified task that has never been used, irrespective of the current position of the task stack pointer. The primary use for it is to determine the amount of task stack space used by a task in order to balance the requirements of the heap and the various task stacks. If the supplied parameter does not refer to a valid task stack (i.e. a stack for a task that is in the task list), the return value will be `&Hffff`.

For the second form, with no task stack specified, the stack of the calling task is examined. In either case, if zero is returned it is nearly certain that the task stack has overflowed, possibly overwriting adjacent data.

Compatibility

For VM mode devices, calling this function for the `Main()` task will always return `&HFFFF` unless you have specified, directly or indirectly, a stack limit for `Main()`. See the `Option MainTaskStackSize`, `Option HeapLimit`, and `Option HeapSize` directives for more information.

See Also   System.HeapHeadRoom

Tan

Type   Function returning Single

Invocation   Tan(arg)

Parameter	Method	Type	Description
arg	ByVal	Single	The angle, in radians, of which the tangent will be computed.

Discussion

The return value will be the tangent of the supplied value. Note that the `Tan()` function may return positive or negative infinity values.

Example

```
Const pi as Single = 3.14159
Dim val as Single

val = Tan(pi / 4.0)      ' result is approximately 1.0
```

See Also   Atn, Atn2, DegToRad, RadToDeg

TaskIsLocked

Type   Function returning Boolean

Invocation   TaskIsLocked()

Discussion

This function will return `True` if the calling task is locked, `False` otherwise.

See Also   LockTask, UnlockTask

TaskIsValid

Type   Function returning Boolean

Invocation   TaskIsValid(taskStack)

Parameter	Method	Type	Description
taskStack	ByRef	array of Byte	The stack for a task of interest.

Discussion

This function will return `True` if the specified task stack is currently in the task list, `False` otherwise. This function can be used with allocated task stacks to determine when it is safe to deallocate the task stack memory.

See Also   StatusTask

Timer

Type   Function returning Single

Invocation   Timer()

Discussion

This function returns the current RTC time represented as the number of seconds since midnight with a best-case resolution of 1/F_RTC_TICK. Note that `Register.RTCTick` gives you the equivalent information albeit in the form of a 32-bit value representing the number of RTC ticks (increments of 1/F_RTC_TICK) since midnight. Depending on your needs, one or the other may be more efficient to use.

Compatibility

This subroutine is not available if the RTC is not enabled in your application.

To*<enum>*

Type	Function returning an Enum member

Invocation	To*<enum>*(val)

Parameter	Method	Type	Description
val	ByVal	integral	The value to convert to an Enum member.

Discussion

This page describes a set of functions that convert the given value to a member of a specific enumeration. For each enumeration that you define in your program the compiler automatically provides a conversion function whose name is the name of the enumeration with the prefix *To*.

To use this conversion function, replace the *<enum>* portion of the function name as shown above with the actual enumeration name for which value-to-member conversion is desired. See below for an example of how this is done.

See the section on enumerations for more information.

Compatibility

This function is provided for backward compatibility. It is recommended to use `CType()` for new applications.

Example

```
Enum Color
      Red
      Green
      Blue
End Enum

Dim c as Color

c = ToColor(1)   ' c will have the value Green
```

See Also	CType

ToggleBits

Type   Subroutine

Invocation   ToggleBits(target, mask)

Parameter	Method	Type	Description
target	ByRef	Byte	The byte to be modified.
mask	ByVal	Byte	The mask indicating which bits to modify.

Discussion

This subroutine allows you to change the state of one or more bits in a byte while leaving others unchanged. Effectively, the result is the same as using the statement below.

```
target = target Xor mask
```

The `mask` parameter governs which bits will get changed. For each bit of the `mask` parameter that is a 1, the corresponding bit of the `target` will be set to the opposite of its current state. Bits of the `target` that correspond to zero bits of the `mask` parameter will remain unchanged.

The advantage to using the `ToggleBits()` subroutine instead of the equivalent statement is twofold. Firstly, it is more efficient, resulting in less code and faster execution time. Secondly, and perhaps more importantly, it performs the action as an atomic operation, i.e. one that is guaranteed, once begun, to complete without an intervening task switch. This characteristic makes Toggle`Bits()` useful for modifying I/O ports and other `Byte` values in a multi-tasking environment.

Example

```
' change the state of the two least significant bits of Port C
Call ToggleBits(Register.PortC, &H03)
```

Compatibility

This routine is not available in BasicX compatibility mode. Also, it is only supported by ZX VM firmware later than v1.0.0.

See Also   SetBits

Trim

Type   Function returning String

Invocation   Trim(str)

Parameter	Method	Type	Description
str	ByVal	String	The string from which blanks will be stripped.

Discussion

This function returns a new string containing the same characters as the passed string except that leading and trailing spaces will be removed. If the string consists solely of spaces, the resulting string will be zero length.

Example

```
Dim s as String, s1 as String
s = "  Hello, world!  "
s2 = Trim(s)            ' the result will be "Hello, world!"
```

See Also   Left, Mid, Right

UBound

Type	Function returning an integral value

Invocation	UBound(array) or
	UBound(array, dimension)

Parameter	Method	Type	Description
array	ByRef	any array	The array about which the bound information is desired.
dimension	ByVal	int16	The dimension of interest. See the discussion for more details.

Discussion

This function returns the upper bound of a dimension of the specified array. There are two forms. The first requires only the array to be specified. In this case, the upper bound of the first dimension of the array is returned. The second form specifies a dimension number (which must be a constant value), the valid range of which is 1 to the number of dimensions of the array. The array may be located in RAM, Program Memory or Persistent Memory.

In contrast to LBound(), a parameter that is an array cannot be passed to UBound() since the return value of UBound() is computed at compile-time and many different sized arrays may be passed as a parameter.

Note that the use of this function instead of hard-coding values makes your code easier to maintain because it automatically adapts if the definition of an array changes.

Example

```
Dim ba(1 to 20) as Byte
Dim ma(3 to 5, -6 to 7) as Byte
Dim i as Integer

i = UBound(ba)          ' the result is 20
i = UBound(ma)          ' the result is 5
i = UBound(ma, 1)       ' the result is 5
i = UBound(ma, 2)       ' the result is 7
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   LBound, Span

UCase

Type   Function returning String

Invocation   UCase(str)

Parameter	Method	Type	Description
str	ByVal	String	The string to be changed to upper case.

Discussion

This function returns a new string containing the same characters as the passed string except that all lower case characters will be replaced with upper case characters.

Example

```
Dim s as String, s1 as String
s = "Hello, world!"
s2 = UCase(s)            ' the result will be "HELLO, WORLD!"
```

See Also   LCase

UnlockTask

Type   Subroutine

Invocation   UnlockTask()

Discussion

This routine causes the running task to become unlocked so that other tasks can run. Calling `UnlockTask()` when a task is not actually locked has no effect.

See Also   LockTask

UpdateRTC

Type   Subroutine

Invocation   UpdateRTC(fastTicks)

Parameter	Method	Type	Description
fastTicks	ByVal	int16	The number of fast ticks to add to the RTC.

Discussion

This subroutine can be used to update the RTC with the number of fast ticks missed during a long operation performed with interrupts disabled. In order to determine the number of fast ticks that are missed, your code must periodically check the interrupt flag of the RTC timer and, if it is set, increment a local counter value and then reset the interrupt flag.

Example

```
' This example is for ZX devices that use Timer0 for the RTC timer.
Atomic
     Dim missedTicks as UnsignedInteger
     Const TickFlag as Byte = &H02
     missedTicks = 0
     Do
          ' place code here that performs one iteration of a
          ' long process and eventually exits the loop

          ' check the RTC flag, reset it
          If (CBool(Register.TIFR0 And TickFlag)) Then
               missedTicks = missedTicks + 1
               Register.TIFR0 = TickFlag
          End If
     Loop
     Call UpdateRTC(missedTicks)
     Call Yield()
End Atomic
```

Compatibility

This subroutine is not available if the RTC is not enabled in your application.

See Also   Yield

ValueI

Type   Subroutine

Invocation   ValueI(str, val, flag)

Parameter	Method	Type	Description
str	ByVal	String	The string from which to extract an `Integer` value.
val	ByRef	int16	The variable to receive the value.
flag	ByRef	Boolean	The variable to receive a success indicator.

Discussion

This routine converts a character representation of an integral number, contained in the `str` parameter, to an `Integer` value returned in the `val` parameter. If the string is in an acceptable format, the `flag` parameter is set to `True`. Otherwise, the `flag` parameter is set to `False` and the `val` parameter will be 0.

The string may contain any number of leading and/or trailing spaces. The value itself may consist of an optional plus or minus sign, an optional radix indicator, and one or more digits. The supported radix indicators are &H for hexadecimal, &O for octal and &B or &X for binary (all case insensitive). If no radix indicator is present, decimal is assumed.

If the provided string has the proper format but represents a value that is too large or too small to be represented as an `Integer`, the result will be invalid but no such indication will be given.

Examples of integral values accepted by `ValueI()` are:

```
103
+123
&H55
-&B01101
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   ValueL, ValueS

ValueL

Type   Subroutine

Invocation   ValueL(str, val, flag)

Parameter	Method	Type	Description
Str	ByVal	String	The string from which to extract an Long value.
Val	ByRef	int32	The variable to receive the value.
Flag	ByRef	Boolean	The variable to receive a success indicator.

Discussion

This routine converts a character representation of an integral number, contained in the str parameter, to a Long value returned in the val parameter. If the string is in an acceptable format, the flag parameter is set to True. Otherwise, the flag parameter is set to False and the val parameter will be 0.

The string may contain any number of leading and/or trailing spaces. The value itself may consist of an optional plus or minus sign, an optional radix indicator, and one or more digits. The supported radix indicators are &H for hexadecimal, &O for octal and &B or &X for binary (all case insensitive). If no radix indicator is present, decimal is assumed.

If the provided string has the proper format but represents a value that is too large or too small to be represented as a Long, the result will be invalid but no such indication will be given.

Examples of integral values accepted by ValueL() are:

```
103
+123
&H55
-&B01101
```

Compatibility

This function is not available in BasicX compatibility mode.

See Also   Value, ValueS

ValueS

Type   Subroutine

Invocation   ValueS(str, val, flag)

Parameter	Method	Type	Description
str	ByVal	String	The string from which to extract a floating point value.
val	ByRef	Single	The variable to receive the value.
flag	ByRef	Boolean	The variable to receive a success indicator.

Discussion

This routine converts a character representation of a floating pointer number, contained in the `str` parameter, to a `Single` value returned in the `val` parameter. If the string is in an acceptable format, the `flag` parameter is set to `True`. Otherwise, the `flag` parameter is set to `False` and the `val` parameter will be 0.0.

The string may contain any number of leading and/or trailing spaces. The value itself may consist solely of decimal digits or may have a leading plus or minus sign. The value may include a decimal point, with or without preceding digits. However, there must be a digit either preceding the decimal point or following it, or both. Optionally, there may be a multiplier value consisting of the letter E (upper or lower case), optionally followed by a plus or minus sign, followed by one or more digits. Note that the range of acceptable input is wider than that for real values in ZBasic statements.

If the provided string has the proper format but represents a value that is too large or too small to be represented as a `Single`, the result will be invalid but no such indication will be given.

Examples of floating point numbers accepted by `ValueS()` are:

```
.30103
3.14159
-200.
1e05
+6.02E+23
123
```

See Also   Value, ValueL

VarPtr

Type   Function returning UnsignedInteger

Invocation   VarPtr(var)

Parameter	Method	Type	Description
var	ByRef	any variable	The variable of which the address is desired.

Discussion

This function returns the `UnsignedInteger` representation of the RAM address of the specified variable. Note that for arrays, you may also specify subscript expressions for all of the array dimensions to yield the address of an individual array element. Without the subscript expressions, the resulting value will be the address of the first element of the array.

This function is useful for deriving the address to pass to the several functions that require a RAM address, e.g. `BitCopy()`, `RamPeek()`, `RamPoke()`, etc.

This function is identical to `MemAddressU()` and is provided for BasicX compatibility.

See Also   MemAddress, MemAddressU

WaitForInterrupt

Type	Subroutine
Invocation	WaitForInterrupt(mode)
	WaitForInterrupt(mode, intNum)

Parameter	Method	Type	Description
mode	ByVal	integral	A value specifying what action will trigger the interrupt. See the discussion below.
intNum	ByVal	Byte	A designator for the interrupt to await (see discussion below).

Discussion

This routine allows a task to suspend itself and wait for an interrupt. The particular interrupt awaited is controlled by the `intNum` designator in combination with the `mode` value.   There are three general sources of interrupts that can be awaited: external interrupts, pin change interrupts and analog comparator interrupts.

External Interrupts 0-7 (ATtiny and ATmega targets)

A task may await an external interrupt by specifying the value 0 through 7 (corresponding to external interrupt INT0 to INT7, respectively) for the `intNum` parameter. Not all target devices support the full range of external interrupts. See the Resource Usage sub-section External Interrupts for information on the available external interrupts for each target device along with the corresponding `intNum` value and the interrupt input pin for each.   The allowable values for the `mode` parameter and their respective meanings are given in the table below.

Hardware Interrupt Mode Values

Value	Built-in Constant	Interrupt Trigger
&H10	zxPinLow	A low level on the interrupt pin.
&H14	zxPinChange	Any logic level change on the interrupt pin.
&H18	zxPinFallingEdge	A high to low transition on the interrupt pin.
&H1C	zxPinRisingEdge	A low to high transition on the interrupt pin.

All other values are reserved for future use. For compatibility with BasicX, there are similarly named built-in constants that begin with the prefix bx instead of zx except that there is no equivalent for `zxPinChange`. Additionally, for all of the targets in the table below, Interrupt 2 is not capable of the first two trigger modes; it can only be triggered on a rising edge or a falling edge.

Target Devices with Limited Functionality on INT2

mega16	mega16A	mega32
mega32A	mega161	mega162
mega323	mega8515	mega8535

The built-in constants `WaitInt0` through `WaitInt7` may be used to specify the `intNum` parameter. If no `intNum` parameter is given, Interrupt 1 is assumed (for compatibility with BasicX). This is equivalent to using `WaitForInterrupt(mode, 1)`.

Pin Change Interrupts (ATtiny and ATmega targets)

For most ATtiny and ATmega target devices, a task may await a state change on one or more pins of one or more I/O ports. This mode is selected by specifying a special value for the `intNum` parameter that indicates the port(s) and the `mode` parameter contains a bit mask indicating the bits of interest. For

example, if the `mode` value is `&H21`, a pin change interrupt will be generated if either bit 0 or bit 5 of the specified port changes state. Clearly, a `mode` value of zero is useless since no pin change interrupt can ever occur in that case. See the Resource Usage sub-section Pin Change Interrupts for information on the available pin change interrupts for each target device.

Pin Change Interrupts (ATxmega targets)

For ATxmega target devices, a task may await a state change on one or more pins of an I/O port. Each port has two separate channels of pin-change detection, e.g. `WaitPinChangeA0` and `WaitPinChangeA1` both sensitive to pin changes on Port A. The port and channel are specified via the `intNum` parameter and the `mode` parameter contains a bit mask indicating the bits of interest. For example, if the `mode` value is `&H21`, a pin change interrupt will be generated if either bit 0 or bit 5 of the specified port changes state. Clearly, a `mode` value of zero is useless since no pin change interrupt can ever occur in that case. See the Resource Usage sub-section Pin Change Interrupts for information on the available pin change interrupts for each target device.

By default, each pin that is enabled for a pin change interrupt will trigger the interrupt on either edge. You may configure the sensitivity for each individual pin to trigger an interrupt on either edge, rising edge only, falling edge only or a low level. The setting for each pin change sensitivity is made in a "pin control" register specific to that pin. For example, the pin control register for bit 3 of port C is named PORTC_PIN3CTRL. Consult the applicable ATxmega datasheet for more information on these registers.

Analog Comparator Interrupt (ATtiny and ATmega targets)

A task may await an analog comparator interrupt by specifying the value `waitAnalogComp` (&H10) for intNum. The corresponding built-in constant is waitAnalogComp. In this case, the mode parameter specifies the comparator output transition that will cause the interrupt to occur.

Analog Comparator Interrupt Mode Values

Value	Built-in Constant	Interrupt Trigger
`&H00`	`zxAnalogCompChange`	Comparator output rising edge or falling edge.
`&H02`	`zxAnalogCompFalling`	Comparator output falling edge.
`&H03`	`zxAnalogCompRising`	Comparator output rising edge.

With all of the `mode` values in the table above, the analog comparator's positive input is AIN0 and the comparator's negative input is either AIN1 or, (on some target devices) if the ACME bit is set in a CPU register (see below), the analog input specified by the multiplexor select bits in Register.ADMUX. Another option for the positive comparator input is to select the internal "band gap" voltage. This voltage level (approximately 1.23 volts) is selected by adding the value &H40 to the mode values in the table above. The built-in constant `zxAnalogReference` has this value.

See the Resource Usage sub-section Analog Comparator Interrupts for information on the location of the AIN0 and AIN1 pins for each target device and which register contains the ACME bit (where available). See the section in the Atmel microcontroller documentation describing the analog comparator for further details.

Analog Comparator Interrupt (ATxmega targets)

On xmega devices, the analog comparator has two channels and a task may await an analog comparator interrupt on one of the analog comparator channels by giving the appropriate value for `intNum`. See the Resource Usage sub-section Analog Comparator Interrupts for information on the selector values and the number of analog comparators supported for individual xmega devices.

In order to use any of the analog comparator interrupts, you must first configure positive and negative inputs to the corresponding comparator(s) using the processor registers ACA_AC0MUXCTRL, ACA_AC1MUXCTRL, ACB_AC0MUXCTRL and/or ACB_AC1MUXCTRL. Also, the high speed/low power control bit and the hysteresis control bits in the ACA_AC0CTRL, ACA_AC1CTRL, ACB_AC0CTRL and

ACB_AC1CTRL registers may be configured as desired before invoking WaitForInterrupt(). Consult the applicable ATxmega datasheet for more information on these registers.

The window mode of the analog comparator utilizes both channels of the comparator, with channel 0 representing the high limit of the window and channel 1 representing the low limit of the window. The mode value to use when setting up an analog comparator interrupt differs depending on whether single channel or window mode is being used, see the tables below.

Analog Comparator Interrupt Mode Values – Single Channel Mode

Value	Built-in Constant	Interrupt Trigger
&H00	zxAnalogCompChange	Comparator output rising edge or falling edge.
&H02	zxAnalogCompFalling	Comparator output falling edge.
&H03	zxAnalogCompRising	Comparator output rising edge.

Analog Comparator Interrupt Mode Values – Window Mode

Value	Interrupt Trigger
&H00	Input signal above the window.
&H01	Input signal inside the window.
&H02	Input signal below the window.
&H03	Input signal outside the window.

Operation

For all forms, when the trigger condition occurs an interrupt will be generated and the task awaiting the interrupt will rise to the highest priority. This will cause an immediate task switch meaning that the next instruction that executes will be the one following the WaitForInterrupt() invocation. Note that if another task performs an action that causes interrupts to be disabled, response to the interrupt will be delayed until interrupts are re-enabled. The fact that the current task is locked does not prevent the interrupt task from executing next.

If two or more interrupts occur simultaneously, the task awaiting the highest priority interrupt is activated first. For VM mode devices, the priorities of the various interrupts are given in the table below.

VM Mode Devices
Interrupt Priority (highest to lowest)

Interrupt 0
Interrupt 1
Interrupt 2
Analog Comparator Interrupt
Interrupt 3
Interrupt 4
Interrupt 5
Interrupt 6
Interrupt 7
Pin Change Interrupt, Port A
Pin Change Interrupt, Port B
Pin Change Interrupt, Port C
Pin Change Interrupt, Port D
Pin Change Interrupt, Port E
Pin Change Interrupt, Port J
Pin Change Interrupt, Port K

For native mode devices, the interrupt priority corresponds to the order of the entries in the processor's interrupt vector table: the lower the vector number the higher the priority. Consult the corresponding ATtiny, ATmega or ATxmega processor datasheet for more information on this topic.

Note that a task awaiting an interrupt will exhibit some latency between the occurence of the interrupt and when the waiting task begins execution. The latency depends on a number of factors including the specific instruction being executed at the time of the interrupt and the number and frequency of system interrupts that need to be handled. Instructions that may take a long time to execute such as OutputCapture(), ShiftIn(), ShiftOut(), X10Cmd(), etc. will introduce more latency than simple instructions like assigning a value to a variable.

Examples

```
Call WaitForInterrupt(zxPinChange)
Call WaitForInterrupt(zxPinRisingEdge, WaitInt2)
Call WaitForInterrupt(&H40, WaitPinChangeA) ' await a change on Port A, bit 6
```

Resource Usage

Only one task can be awaiting each interrupt at any particular time. If a task is already awaiting the specified interrupt, another call to `WaitForInterrupt()` for that same interrupt will return immediately.

Also, on the ZX-24 the interrupt pins are common with I/O pins as shown in the table below. This means that you should set the corresponding pin to be an input (either tri-state or pull-up) when you want to use `WaitForInterrupt()`. Note, however, that if the pin is an output and a task is awaiting an interrupt, a transition on the corresponding output can generate the interrupt for the waiting task. This may be of use in special situations as a "software interrupt".

Interrupt and I/O Pin Sharing for
ZX-24, ZX-24a, ZX-24p, ZX-24n, ZX-24r, ZX-24s, ZX-24t

Interrupt	Port/Bit	Pin
0	Port C, Bit 6	6
1	Port C, Bit 1	11
2	Port A, Bit 2	18

Compatibility

The second parameter is not supported in BasicX compatibility mode. The built-in constant `zxPinChange` is not available in BasicX. It is not known if the capability is supported or not.

WaitForInterval

Type   Subroutine

Invocation   WaitForInterval(flags)

Parameter	Method	Type	Description
flags	ByVal	Byte	A set of flag bits that control the operation. See the discussion below.

Discussion

This routine allows a task to suspend itself and wait for an interval timer to expire. The length of the interval is set by the routine `SetInterval`. Note that there is only one interval timer that is shared by all tasks. This means that at most one task may be awaiting the expiration of an interval at any one time. If another task is already awaiting an interval, calls to `WaitForInterval` will return immediately.

The bit values for the `flags` parameter are described in the table below.

<div align="center">

Interval Timer Flag Values

Value	Description
&H01	Wait until the next interval expires.
&H02	Reset the interval counter to its original value.

</div>

The remaining bits are currently undefined but may be employed in the future.

After a call to `SetInterval` the interval counter is decremented on every RTC tick. When it reaches zero, if a task is awaiting the expiration of the interval, that task will be scheduled to run immediately. If no task is awaiting the expiration of the interval, the fact that the interval expired is recorded and the interval counter is reset to the original value.

If the `flags` value is zero when a task calls `WaitForInterval` and an interval expiration has previously been recorded (with no waiting task), the call will return immediately. Otherwise, the task will be suspended until the interval expiration. If the `flags` value is &H01, the task will be suspended until the next expiration of the interval. If the flags value is &H03, interval counter will be reloaded and then the task will be suspended until the interval expires. The last mode of operation is similar to a task calling `Sleep`. The difference is that when the interval expires, the task is immediately reactivated. With a `Sleep` call, the task will execute again when its sequential turn comes up.

A task awaiting the expiration of an interval has lower priority than one awaiting an interrupt. Note that a task awaiting the expiration of an interval will exhibit some latency between the expiration of the interval and when the waiting task begins execution. The latency depends on a number of factors including the specific instruction being executed at the time and the number and frequency of system interrupts that need to be handled. Instructions that may take a long time to execute such as OutputCapture, ShiftIn, ShiftOut, X10Cmd, etc. will introduce more latency than simple instructions like assigning a value to a variable.

Example

```
Call SetInterval(1.0)
Do
    Call WaitForInterval(0)
    <other code here>
Loop
```

Resource Usage

The interval counter is driven off of the real time clock. If interrupts are disabled for long periods of time, the timing won't be accurate. I/O routines that disable interrupts typically track RTC ticks and then update the RTC when the I/O process has completed. At this same time, the interval counter will be updated as well accounting for, at most, one missed expiration.

There is a single, system-wide interval timer. Only one task can be awaiting an interval at a time. If a task is already waiting, another call to `WaitForInterval()` will return immediately.

Compatibility

If the RTC is not included in your application this routine will not be available. This routine is also not available in BasicX compatibility mode.

See Also   SetInterval

WatchDog

Type   Subroutine

Invocation   WatchDog()

Discussion

This routine resets the watchdog timer, preventing it from resetting the system. A watchdog timer is useful to ensure that your program continues to operate normally.

To implement a watchdog timer you first call `OpenWatchDog()` to prepare the watchdog timer for use. Thereafter, if your program doesn't call `WatchDog()` often enough, the watchdog will eventually time out and cause a system reset.

See Also   CloseWatchDog**,** OpenWatchDog

X10Cmd

Type   Subroutine

Invocation   X10Cmd(outPin, syncPin, house, devCmd, count)
                  X10Cmd(outPin, syncPin, house, devCmd, count, flags)

Parameter	Method	Type	Description
outPin	ByVal	Byte	The pin on which the X10 signal will be generated.
syncPin	ByVal	Byte	The pin on which the 60Hz sync signal will be received.
house	ByVal	Byte	The house code.
devCmd	ByVal	Byte	The device code or command code.
count	ByVal	Byte	The number of times to repeat the transmission.
flags	ByVal	Byte	Flag bits to control the operation of the command.

Discussion

This routine produces an X-10 compatible signal on the pin specified by `outPin`. The signal is synchronized to the zero-crossing signal on the pin specified by `syncPin`. The generated signal will include the specified house code and command/device code and will be repeated the specified number of times without any spacing between the code sequences. The X-10 specification indicates that most commands should be repeated twice and that successive commands should be separated by at least 3 power line cycles (~50 milliseconds). The exception is for bright and dim commands that can be repeated any number of times.
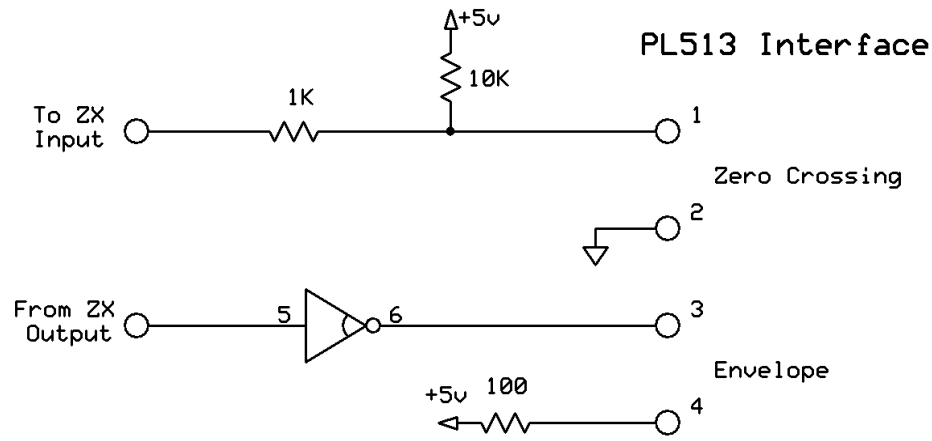
If the `flags` parameter is not present, the transmission is implemented as a single 1millisecond pulse near the edge of the zero crossing signal. If the `flags` parameter is present it has the effects shown in the table below depending on the value of the parameter.

Function	Hex Value	Bit Mask
Three-phase output	&H01	xx xx xx x1
50Hz timing	&H02	xx xx xx 1x

If the three-phase output flag bit is asserted, three 1 millisecond pulses will be output during each half-cycle. The 50Hz timing flag is used to control the phase timing of the three-phase output. If the flag bit is not asserted, 60Hz timing is utilized. This flag bit is only used when generating three-phase output.

External Circuitry

In order to control X-10 devices, you will need a power line interface device such as the PL513 or the TW523, both of which are available from a variety of sources. The technical documentation for both interface devices is available on the Internet. A simplified interface between the ZX and the PL513 is shown below. Note that this circuit will not work for the TW523. The suggested OEM circuit in the X10 Technical Note, or something similar, should be used.

PL513 Interface

```
          +5v
           ▲
          ▓ 10K
          ▓
  1K       │
To ZX  ───\/\/\───●──────────── ○ 1
Input
                              Zero Crossing
                        ┌──── ○ 2
                       ▽

From ZX   5  ▷o 6
Output  ───────▷o───────────── ○ 3

                              Envelope
       +5v  100
       ◁──\/\/\──────────── ○ 4
```

Simple PL513 Interface

Example

The code below sends the commands to turn on device A1.

```
Const HouseCodeA as Byte = &H06
Const DeviceCode1 as Byte = &H0c
Const DeviceOn as Byte = &H05

Call X10Cmd(20, 19, HouseCodeA, DeviceCode1, 2)
Call Delay(0.50)
Call X10Cmd(20, 19, HouseCodeA, DeviceOn, 2)
```

Compatibility

The BasicX documentation indicates that the transmission process is done in the background. On this implementation X10Cmd() will not return until the transmission is complete. In BasicX compatibility mode the flags parameter is not supported.

Yield

Type   Subroutine

Invocation   Yield()

Discussion

This routine is can be called whenever it is desirable to allow another task to run that is ready to run. One particular situation in which it is useful is at the end of a long process during which UpdateRTC() has been called one or more times. Normally, when an RTC interrupt occurs a task switch is performed immediately if the current task's time slice has expired or if a task is awaiting the expiration of an interval and the interval period has elapsed. However, if interrupts are disabled this automatic task switch cannot be performed. A call to UpdateRTC() will prepare the system for an eventual task switch which is then triggered by a call to Yield().

Example

See the example at UpdateRTC.

See Also   UpdateRTC

ZXCmdMode

Type   Subroutine

Invocation   ZXCmdMode()
ZXCmdMode(highSpeed)

Parameter	Method	Type	Description
highSpeed	ByVal	Boolean	A flag controlling the communication speed in command mode.

Discussion

This routine causes the ZX (or generic target device, if using a ZBasic-compatible bootloader) to stop executing your application and enter "command mode". When in command mode, the device will respond to download commands and other special commands. If the `highSpeed` parameter is specified and it is `True`, command mode is invoked and the baud rate of Com1 is changed to 115.2K baud (the standard download baud rate). If the `highSpeed` parameter is `False` or omitted, command mode is invoked and the baud rate is left unchanged (however, see the Compatability section, below).

Note: for compatibility with the IDE and the zload utility, this subroutine must be invoked with a `False` or omitted parameter. The `highSpeed` parameter is provided in case a custom downloader might require it.

You can use this routine in your application to facilitate downloading triggered by some particular event, e.g. receipt of a certain character or sequence of characters, the occurrence of an external signal, etc. You can use the downloader DLL source code (installed as part of ZBasic) to construct a special purpose downloader for your application. Alternately, if your application detects receipt of an "ATN character" and then invokes `ZXCmdMode()`, you can use the ZLoad command line utility or the ZBasic IDE to perform downloading without needing to have DTR connected to the device.

Example

```
Call ZXCmdMode()
```

Compatibility

With VM versions prior to v3.0.4 and with native mode bootloaders prior to v1.4, invoking this subroutine with a `False` or omitted parameter results in the baud rate switching to 19.2K, the standard debug baud rate. In most cases, this change will be insignificant because in order for the download to succeed the serial channel would be set to 19.2K baud and the Com1 baud rate would be 19.2K baud as well. Since the new behavior does not change the Com1 baud rate, existing applications should continue to work as they did before. The new behavior was implemented for compatibility with fixed-speed communication links such as BlueTooth and XBee. Since the baud rate doesn't get changed, you can configure your system for any desirable baud rate and perform downloading over the fixed-speed link.

The VM version can be determined using the `SerialNumber()` subroutine. The bootloader version number can be determined using `Register.BootVersion`.