# Basic Express
# Biblioteca del sistema

Versión 2.0





Traducción española: Alicia Bernal, Revisión: Pablo Pompa
www.superrobotica.com

2.00.H

# Biblioteca del sistema

El sistema operativo BasicX proporciona una biblioteca de llamadas del sistema clasificadas en las categorías siguientes:

## Funciones matemáticas (Math functions)

| | |
|---|---|
| Abs | Valor absoluto |
| ACos | Arco coseno |
| ASin | Arco seno |
| Atn | Arco tangente |
| Cos | Coseno |
| Exp | Eleva *e* a la potencia especificada |
| Exp10 | (Eleva *10* a la potencia especificada) |
| Fix | Trunca un valor de punto flotante |
| Log | Logaritmo natural |
| Log10 | Logaritmo en base 10 |
| Pow | Eleva un operando a una determinada potencia |
| Randomize | Fija el punto inicial (*seed)* para el generador de números aleatorios |
| Rnd | Genera un número aleatorio |
| Sin | Seno |
| Sqr | Raíz cuadrada |
| Tan | Tangente |

## Funciones de cadenas (String functions)

| | |
|---|---|
| Asc | Devuelve el código ASCII de un carácter |
| Chr | Convierte un valor numérico en un carácter |
| LCase | Convierte una cadena a minúsculas |
| Len | Devuelve la longitud de una cadena |
| Mid | Copia una subcadena |
| Trim | Recorta los espacios en blanco de delante y detrás de una cadena |
| UCase | Convierte una cadena a mayúsculas |

## Funciones relacionadas con la memoria (Memory-related functions)

| | |
|---|---|
| BlockMove | Copia un bloque de datos desde una ubicación de la memoria RAM a otra. |
| FlipBits | Genera una imagen espejo de un patrón de bits (Sólo BX-24, BX-35) |
| GetBit | Lee un solo bit de una variable (Sólo BX-24, BX-35) |
| GetEEPROM | Lee los datos desde la memoria EEPROM |
| GetXIO | Lee los datos desde las entradas/salidas adicionales (Sólo BX-01)) |
| GetXRAM | Lee los datos desde la memoria XRAM (Sólo BX-01) |
| MemAddress | Devuelve la dirección de una variable o matriz |
| MemAddressU | Devuelve la dirección de una variable o matriz |
| PersistentPeek | Lee un byte desde la memoria EEPROM |
| PersistentPoke | Escribe un byte en la memoria EEPROM |
| PutBit | Escribe un sólo bit en una variable (Sólo BX-24, BX-35) |
| PutEEPROM | Escribe datos en la memoria EEPROM |
| PutXIO | Escribe datos en las entradas/salidas ampliadas (Sólo BX-01) |
| PutXRAM | Escribe datos en la memoria XRAM (Sólo BX-01) |
| RAMPeek | Lee un byte desde la memoria RAM |
| RAMPoke | Escribe un byte en la memoria RAM |
| SerialNumber | Devuelve el número de versión de un chip BasicX |

## Colas

| | |
|---|---|
| GetQueue | Lee datos de una cola |
| OpenQueue | Define una matriz como cola |
| PeekQueue | Ve los datos de la cola sin eliminar ningún dato de ella |
| PutQueue | Escribe datos en una cola |
| PutQueueStr | Escribe una cadena en una cola |
| StatusQueue | Determina si una cola tiene datos que se pueden leer |

## Tareas

| | |
|---|---|
| CallTask | Inicia una tarea |
| CPUSleep | Pone el procesador en distintos modos de baja potencia |
| Delay | Detiene momentáneamente la tarea y permite que se ejecuten otras tareas |
| DelayUntilClockTick | Detiene momentáneamente la tarea hasta que el reloj de tiempo real vuelva a marcar |
| FirstTime | Determina si el programa ha sido ejecutado alguna vez desde su descarga |
| LockTask | Bloquea la tarea y evita que se ejecuten las otras tareas |
| OpenWatchdog | Inicia el temporizador watchdog |
| ResetProcessor | Resetea y reinicia el procesador |
| Semaphore | Coordina la compartición de los datos entre las tareas |
| Sleep | Detiene momentáneamente la tarea y permite que se ejecuten otras tareas |
| TaskIsLocked | Determina si la tarea está bloqueado |
| UnlockTask | Desbloquea una tarea |
| WaitForInterrupt | Permita una tarea responder a una interrupción de hardware |
| Watchdog | Resetea el temporizador watchdog |

## Conversiones de tipos

| | |
|---|---|
| CBool | Convierte Byte a Boolean (Booleano) (Sólo BX-24, BX-35) |
| CByte | Convierte a Byte (Byte) |
| CInt | Convierte a Integer (Entero) |
| CLng | Convierte a Long (Largo) |
| CSng | Convierte a floating point (single) (punto flotante –sencillo) |
| CStr | Convierte a string (cadena) |
| CuInt | Convierte a UnsignedInteger (entero sin signo) |
| CuLng | Convierte a UnsignedLong (largo sin signo) |
| FixB | Trunca un valor de punto flotante, y los convierte a Byte (Byte) |
| FixI | Trunca un valor de punto flotante, y los convierte a Integer (Entero) |
| FixL | Trunca un valor de punto flotante, y los convierte a Long (Largo) |
| FixUI | Trunca un valor de punto flotante, y los convierte a UnsignedInteger (Entero sin signo) |
| FixUL | Trunca un valor de punto flotante, y los convierte a UnsignedLong (Largo sin signo) |
| ValueS | Convierte una cadena a un tipo flotante (single) –sencillo. |

## Reloj de tiempo real

| | |
|---|---|
| GetDate | Devuelve la fecha |
| GetDayOfWeek | Devuelve el día de la semana |
| GetTime | Devuelve la hora del día |
| GetTimestamp | Devuelve la fecha y hora del día |
| PutDate | Fija la fecha |
| PutTime | Fija la hora del día |
| PutTimestamp | Fija la fecha, día de la semana y hora del día |
| Timer | Devuelve los segundos en puntos flotantes desde la media noche |

## Pines de entradas/salidas (I/O)

| | |
|---|---|
| ADCToCom1 | Trasmite datos desde el puerto ADC al puerto serie (Sólo BX-24, BX-35) |
| Com1ToDAC | Trasmite datos desde el puerto serie al puerto DAC (Sólo BX-24, BX-35) |
| CountTransitions | Cuenta las transiciones lógicas en un pin de entrada (Sólo BX-24, BX-35) |
| DACPin | Genera un voltaje pseudo-analógico en el pin de salida |
| FreqOut | Genera ondas sinusoidales duales en el pin de salida (Sólo BX-24, BX-35) |
| GetADC | Devuelve un voltaje analógico (Sólo BX-24, BX-35) |
| GetPin | Devuelve el nivel lógico de un pin de entrada |
| InputCapture | Registra el tren de pulsos en el pin de entrada de captura |
| OutputCapture | Envía un tren de pulsos al pin de salida de captura |
| PlaySound | Reproduce el sonido de los datos de muestra de la memoria EEPROM (Sólo BX-24, BX-35) |
| PulseIn | Asegura la amplitud de pulsos del pin de entrada |
| PulseOut | Envía un pulso a un pin de salida |
| PutDAC | Genera voltaje pseudo-analógico en el pin de salida |
| PutPin | Configura un pin con 1 de los 4 estados de salida o entrada |
| RCTime | Mide el retardo de tiempo hasta que tiene lugar una transición de pin |
| ShiftIn | Cambia los bits desde un pin I/O a una variable de byte (Sólo BX-24, BX-35) |
| ShiftOut | Cambia los bits de una variable de byte variable a un pin I/O (Sólo BX-24, BX-35) |

## Comunicaciones

| | |
|---|---|
| Debug.Print | Envía una cadena al puerto serie Com1 |
| DefineCom3 | Define los parámetros de las entradas/salidas serie de un pin arbitrario (Sólo BX-24, BX-35) |
| Get1Wire | Recibe un bit de datos utilizando el protocolo Dallas 1-Wire (Sólo BX-24, BX-35) |
| OpenCom | Abre un puerto serie RS-232 |
| OpenSPI | Abre las comunicaciones SPI |
| Put1Wire | Trasmite un bit de datos utilizando el protocolo Dallas 1-Wire (Sólo BX-24, BX-35) |
| SPICmd | Comunicaciones SPI |
| X10Cmd | Transmite datos X-10 (Sólo BX-24, BX-35) |

## Network (Sólo BX-01)

| | | |
|---|---|---|
| GetNetwork | Lee los datos desde una ubicación RAM remota | (Sólo BX-01) |
| GetNetworkP | Lee los datos desde una ubicación EEPROM remota | (Sólo BX-01) |
| OpenNetwork | Abre una red | (Sólo BX-01) |
| PutNetwork | Envía los datos a una ubicación RAM remota | (Sólo BX-01) |

| | | |
|---|---|---|
| PutNetworkP | Envía los datos a una ubicación EEPROM remota | (Sólo BX-01) |
| PutNetworkPacket | Envía un paquete especial a través de la red | (Sólo BX-01) |
| PutNetworkQueue | Envía datos a una cola remota | (Sólo BX-01) |

Las siguientes llamadas del sistema BX-24 requieren la versión 2.1 del chip BX-24 (o superior):

ADCToCom1
CBool
Com1ToDAC
FlipBits
GetBit
PutBit
ShiftIn
ShiftOut

Puede conocer la versión del chip BasicX puede utilizando el procedimiento SerialNumber en todos los sistemas BasicX. En los sistemas BX-24, la versión 2.1 puede identificarse de manera visual a través de un punto amarillo en el chip SPI EEPROM.

# Función Abs

## Sintaxis

*F* = Abs(*Operand*) (Operando)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand (Operando)* | Cualquier tipo numérico | Entrada | Operando |
| *F* | Mismo que el operando | Salida | Valor devuelto por función |

## Descripción

Devuelve el valor absoluto del operando.

## Ejemplo

```
Dim X As Single
Dim I As Integer

X = Abs(-5.3) ' X es 5.3

I = Abs(-47)  ' I es 47
```

## Errores conocidos

La función Abs puede generar mensajes de error de tipos de correspondencias erróneas en expresiones de los siguientes tipos:

Long
UnsignedLong
UnsignedInteger

# Función ACos

## Sintaxis

*F* = ACos(*Operand*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Calcula el arco coseno. El valor devuelto por función está expresadoen unidades de radianes.

## Ejemplo

```
Dim F As Single

F = ACos(0.707107) ' F es Pi/4 radianes (45 grados)
```

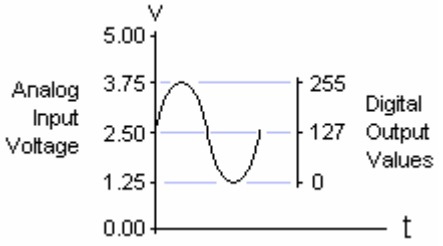
# Procedimiento ADCToCom1

**Sintaxis** **Sólo BX-24, BX-35**

Call ADCToCom1(*PinNumber, DataRate*)

**Argumentos**

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *PinNumber* | Byte | Entrada | Número de pin de ADC. El rango es de 13 a 20. |
| *DataRate* | Integer | Entrada | Tasa de datos. Las unidades son muestras por segundo, a 1 byte por muestra. El rango es de 113 a 11.000. |




## Descripción

ADCToCom1 lee el ADC (Convertidor de señal analógica a digital) y envía los datos desde el puerto serie Com1 con la tasa de datos en muestras por segundo de *DataRate*. La tasa de baudios es constante a 115.200 baudios (No es necesario el procedimiento OpenCom dado que este procedimiento utiliza el puerto serie Com1). Para detener la transmisión, deberá llamar al procedimiento utilizando 0 como *PinNumber*.

**Salida digital:** transmisión de bytes en el rango de 0 a 255 y escalado aproximadamente como para que la entrada de 1.25 V genere una salida de 0, y una entrada de 3.75 V genere una salida de 255.

**Entrada analógica:** Una señal AC centrada a 2.5 V con un rango máximo de 2.5 +/- 1.25 V, como se muestra en la imagen.

## Advertencia

No se deberían realizar otras lecturas de ADC mientras ADCToCom1 esté activa. Así mismo, este procedimiento utiliza el temporizador Timer1, lo que significa que podría entrar en conflicto con cualquier elemento que dependa de dicho temporizador Timer1, como por ejemplo Input Capture y Output Capture.

## Ejemplo

```
' Lee el pin 16 de ADC, envía la señal a Com1 a 5000 muestras/seg.
Call ADCToCom1(16, 5000)

' Detener la transmisión después de 1 segundo.
Call Delay(1.0)
Call ADCToCom1(0, 5000)
```

# Función Asc

## Sintaxis

*F* = Asc(*Source*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Source* | String | Entrada | Fuente de la cadena. |
| *F* | Byte | Salida | Código ASCII del primer carácter de *Source*. |

## Descripción

Devuelve el código ASCII del primer carácter de una cadena.

## Ejemplo

```
Dim Tx As String * 3, Code As Byte
Tx = "ABC"
Code = Asc(Tx) ' El código es 65 (ASCII "A")
```

# Función ASin

## Sintaxis

*F* = ASin(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Calcula el arco seno. El valor devuelto por la función está expresado en unidades de radianes.

## Ejemplo

```
Dim F As Single

F = ASin(1.0) ' F es Pi/2 radianes (90 grados)
```

# Función Atn

## Sintaxis

*F* = Atn(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Calcula el arco tangente. El valor devuelto por función está expresado en unidades de radianes.

## Ejemplo

```
Dim Y As Single

F = Atn(1.0) ' F es Pi/4 radianes (45 grados).
```

# Función BlockMove

## Sintaxis

Call BlockMove(*NumberOfBytes, SourceAddress, DestinationAddress*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *NumberOfBytes* | UnsignedInteger | Entrada | Número de bytes que copiar. El rango legal es de 1 a 65.535. |
| *SourceAddress* | UnsignedInteger | Entrada | Dirección de la fuente. |
| *DestinationAddress* | UnsignedInteger | Entrada | Dirección de destino. |

## Descripción

Copia un bloque de memoria desde la fuente hasta el destino en la memoria RAM. La función BlockMove puede copiar un bloque grande de memoria de forma arbitraria en una sola operación; además el bloque puede aplicarse a múltiples variables en la memoria.

## Ejemplo

```
Sub Main()

    Dim UI As New UnsignedInteger
    Dim B(1 To 2) As Byte

    UI = &h9ABC&

    ' Copiar la variable de entero sin signo de 16 bits a la
    ' matriz de dos bytes.
    Call BlockMove( 2, MemAddress(UI), MemAddress(B) )

    ' En este punto, B(1) es BCh y B(2) es 9Ah (tenga en cuenta que
    ' B(2) es el byte más importante).

End Sub
```

# Procedimiento **CallTask**

## Sintaxis

CallTask *"TaskName", TaskStack*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *TaskName* | Nombre de tarea | Entrada | Nombre del procedimiento a utilizar como tarea. El nombre debe estar entre comillas. |
| *TaskStack* | Matriz de Bytes | Entrada/Salida | Memoria de la pila que utilizará la tarea – debe ser una matriz de bytes de nivel de módulo. |

## Descripción

CallTask inicia un procedimiento como una tarea paralela en ejecución. El procedimiento *TaskName* es como cualquier otro procedimiento excepto en que no puede tener parámetros formales.

La tarea debe tener memoria asignada para su uso como área de trabajo. Esta es la función de *TaskStack*. Esta matriz se utiliza como área de trabajo de la memoria para procesar las expresiones, las funciones matemáticas, y llamar a los subprogramas, etc.

Podrán ejecutarse múltiples tareas al mismo tiempo, hasta el límite disponible de memoria de la pila. Cada tarea se ejecuta de manera secuencial. Se pueden ejecutar múltiples copias de la misma tarea al mismo tiempo utilizando diferentes pilas.

Las tareas pueden iniciar otras tareas, y una tarea puede llamar a otros subprogramas. Excepto el programa principal, siempre que se sale de una tarea, bien a través de un comando End Sub o a través de un comando Exit Sub, la tarea finaliza y deja de ejecutarse. La pila utilizada por la tarea es liberada para la utilice otra tarea.

El programa principal es una excepción. Nunca finaliza siempre que el procesador siga en funcionamiento.

## Advertencia

Si una tarea no tiene espacio de pila suficiente, todo el sistema del chip BasicX se volverá inestable. Por lo tanto, es preferible pecar de mucho espacio en la pila que de poco espacio.

Tenga en cuenta que la matriz *TaskStack* requiere 15 bytes de sobrecarga para su estructura interna de tareas. Esto significa que la matriz necesita al menos 15 bytes de largo, más el espacio suficiente para la pila real de las tareas.

## Ejemplo

Consulte el fichero de ejemplo CallTaskExample.bas en el subdirectorio BX01_Docs\Examples\Doc_Examples.

# Función CBool

## Sintaxis

**Sólo BX-24, BX-35**

*F* = CBool(*Operand*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Byte | Entrada | Operando |
| *F* | Boolean | Salida | Valor devuelto por función |

## Descripción

Convierte un tipo Byte en un tipo Boolean.

Si el operando es cero, la función devuelve un valor False. Si el operando es distinto de cero, la función devuelve un valor True.

## Ejemplo

```
Dim B As Boolean

B = CBool(255) ' B es True.

B = CBool(127) ' B es True.

B = CBool(0)   ' B es False.
```

# Función CByte

## Sintaxis

*F* = CByte(*Operand*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Cualquier numérico | Entrada | Operando |
| *F* | Byte | Salida | Valor devuelto por función |

## Descripción

Convierte cualquier tipo numérico en un tipo Byte.

## Ejemplo

```
Dim X As Single
Dim B As Byte

X = 2.4
B = CByte(X) ' B es 2
```

# Función Chr

## Sintaxis

*F* = Chr(*Code*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Code* | Byte | Entrada | Código ASCII del carácter. |
| *F* | String | Salida | Carácter en cadena. |

## Descripción

Convierte un código ASCII en un carácter de una cadena. Se la cadena de destino tiene más de un carácter, la cadena se queda justificada y se rellena el espacio en blanco.

## Ejemplo

```
Dim Tx as String * 1
Tx = Chr(65) ' Tx es "A" (ASCII 65).
```

# Función CInt

## Sintaxis

*F* = CInt(*Operand*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Cualquier numérico | Entrada | Operando |
| *F* | Integer | Salida | Valor devuelto por función |

## Descripción

Convierte cualquier tipo numérico en un tipo Integer.

## Ejemplo

```
Dim X As Single
Dim Y As Integer

X = 1.5
Y = CInt(X) ' Y es 2
```

# Función CLng

## Sintaxis

*F* = CLng(*Operand*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Cualquier numérico | Entrada | Operando |
| *F* | Long | Salida | Valor devuelto por función |

## Descripción

Convierte cualquier tipo numérico en un tipo Long.

## Ejemplo

```
Dim X As Single
Dim L as Long

X = 1.5
L = CLng(X) ' L = 2
```

# Procedimiento **Com1ToDAC**

## Sintaxis

**Sólo BX-24, BX-35**

Call Com1ToDAC(*PinNumber*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *PinNumber* | Byte | Entrada | Número de pin entrada/salida (I/O) |

## Descripción

Este procedimiento envía los datos desde el puerto serie Com1 al convertidor DAC en el pin de salida *PinNumber*. La fuente de los datos debería ser un sistema remoto BasicX que ejecute el procedimiento ADCToCom1.

A medida que los bytes llegan al puerto serie, el procesador modificará el valor del convertidor DAC con el que comunicarse. La tasa de baudios de Com1 se fija automáticamente en 115.200 baudios (no es necesario llamar OpenCom). Se actualizará el convertidor DAC con una tasa constante de actualización de 10.000 por segundo, mientras que el rango de voltaje de salida es de 0 V a 5 V (en los sistemas de 5 V).

Hay dos formas de terminar un procedimiento Com1ToDAC. En primer lugar, puede llamar al procedimiento con un *PinNumber* de 0. En segundo lugar, el sistema remoto puede terminar su procedimiento ADCToCom1.

## Ejemplo

```
' Enviar los datos desde el puerto Com1 a un puerto DAC en el pin 17.
Call Com1ToDAC(17)

' Desactivar la transmisión después de 1,5 segundos.
Call Delay(1.5)
Call Com1ToDAC(0)
```

# Función Cos

## Sintaxis

*F* = Cos(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Calcula el coseno. El operando está expresado en unidades de radianes.

## Ejemplo

```
Dim F As Single

' Cos(Pi/4)
F = Cos(0.785398) ' F es 0.707 107
```

# Función CountTransitions (versión flotante)

### Sintaxis

**Sólo BX-24, BX-35**

*F* = CountTransitions(*PinNumber, TimeInterval*)

### Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *PinNumber* | Byte | Entrada | Número de pin. |
| *TimeInterval* | Single | Entrada | Intervalo de tiempo de recuento. Las unidades están expresadas en segundos. El rango es aprox. desde 2.441 µs a 4800.0 s. La resolución es aproximadamente 2.441 µs. |
| *F* | Long | Salida | Número de transiciones. |

### Descripción

Esta función cuenta el número de transiciones lógicas que tienen lugar desde un intervalo de tiempo especificado. Tanto los flancos ascendentes como los descendentes se registran como transiciones. La tasa de máxima de muestras es 409.600 muestras/seg.

El recuento comienza inmediatamente al llamar a la función. Si no se producen transiciones a lo largo del intervalo de tiempo especificado, la función devuelve el valor 0.

### Advertencia

Este procedimiento detiene todas las capacidades de multitarea durante la duración de la llamada. El reloj de tiempo real (RTC), la conmutación de tareas y el tráfico de la red quedan suspendidos durante este tiempo. Si *TimeInterval* es comparable en tamaño o mayor que el periodo de marcación del reloj de tiempo real (RTC) -aproximadamente 1,95 milisegundos-, el reloj de tiempo real perderá tiempo.

# Función CountTransitions (versión de entero)

**Sintaxis** **Sólo BX-24, BX-35**

*F* = CountTransitions(*PinNumber, TimeInterval*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *PinNumber* | Byte | Entrada | Número de pin.. |
| *TimeInterval* | Long | Entrada | Intervalo de tiempo en el que realizar el recuento. Las unidades son (1 / 409 600) segundos (aprox. 2.441 µs). |
| *F* | Long | Salida | Número de transiciones. |

## Descripción

Esta función recuenta el número de transiciones lógicas que tienen lugar durante el espacio de tiempo especificado. Tanto los flancos ascendentes como los descendentes se registran como transiciones. La tasa de máxima de muestras 409 600 muestras/seg.

El recuento comienza inmediatamente al llamar a la función. Si no se producen transiciones a lo largo del intervalo de tiempo especificado, la función devuelve el valor 0.

## Advertencia

Este procedimiento detiene todas las capacidades de multitarea durante la duración de la llamada. El reloj de tiempo real (RTC), la conmutación de tareas y el tráfico de la red quedan suspendidos durante este tiempo. Si *TimeInterval* es comparable en tamaño o mayor que el periodo de marcación del reloj de tiempo real (RTC) -aproximadamente 1,95 milisegundos-, el reloj de tiempo real perderá tiempo.

# Procedimiento **CPUSleep**

## Sintaxis

Call CPUSleep( )

## Argumentos

Ninguno.

## Descripción

Este procedimiento hace que el procesador ejecute una instrucción especial SLEEP escrita en lenguaje de máquina. Esta instrucción puede poner al procesador en distintos modos de baja potencia dependiendo de la configuración de los registros internos. Consulte la documentación de Atmel sobre el uso de la instrucción SLEEP.

# Función CStr

## Sintaxis

*F* = CStr(*Operand*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Booleano o numérico | Entrada | Operando |
| *F* | String (Cadena) | Salida | Valor devuelto por función |

## Descripción

Convierte los tipos Boolean y numérico a tipos String.

## Ejemplo

```
Dim Tx As String, B As Boolean

Tx = "V = " & CStr(-193) & " m/s"  ' Tx es "V = -193 m/seg."

B = True
Tx = "State = " & CStr(B)          ' Tx es "State = True"
```

# Función CSng

## Sintaxis

*F* = CSng(*Operand*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Cualquier numérico | Entrada | Operando |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Convierte cualquier tipo numérico en tipo Single.

## Ejemplo

```
Dim Y As Single
Dim I As Integer

I = 3
Y = CSng(I) ' Y es 3.0
```

# Función CuInt

## Sintaxis

*F* = CuInt(*Operand*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Cualquier numérico discreto | Entrada | Operando |
| *F* | UnsignedInteger | Salida | Valor devuelto por función |

## Descripción

Convierte cualquier tipo numérico discreto (no flotante) al tipo UnsignedInteger.

## Ejemplo

```
Dim L As Long
Dim U As New UnsignedInteger

L = 65535
U = CuInt(L) ' U es 65 535
```

# Función CuLng

## Sintaxis

*F* = CuLng(*Operand*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Cualquier numérico discreto | Entrada | Operando |
| *F* | UnsignedLong | Salida | Valor devuelto por función |

## Descripción

Convierte cualquier tipo numérico discreto (no flotante) al tipo UnsignedLong.

## Ejemplo

```
Dim U As New UnsignedLong
Dim B As Byte

B = 255
U = CuLng(B) ' Conversión de tipo -- U es ahora 255
```

# Procedimiento **DACPin**

## Sintaxis

Call DACPin(*Pin, DACvalue, DACcounter*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Pin* | Byte | Entrada | Número de pin. |
| *DACvalue* | Byte | Entrada | Voltaje de salida, el rango es de 0 a 255 unidades. Conversiones de unidades:<br>0 = 0 voltios<br>255 = 5 voltios (en sistemas de 5 V)<br>**Ejemplo:** Conversión de voltios a DACvalue<br>1.6 voltios = 1.6 V * 256 / 5.0V = DACvalue = 82<br>**Ejemplo:** Conversión de DACvalue a voltios<br>167 recuentos = 167 * 5.0 V / 256 = 3.26 voltios |
| *DACcounter* | Byte | Entrada/Salida | DACcounter es un valor que debe obtenerse cada vez que se llama a la rutina de modo que DAC siga sincronizado. Si tiene varios convertidores DAC en ejecución de manera simultánea, entonces deberá tener un DACcounter diferente para cada pin. |

## Descripción

DACPin genera un voltaje pseudoanalógico de 8 bits en un pin de entrada/salida (I/O). En los sistemas de 5 voltios, el rango de voltaje es desde 0.0 V a 5.0 V, con una resolución de aproximadamente 19.6 mV.

Se cronometra un conjunto rápido de pulsos para producir el voltaje deseado. Se necesitará externamente un circuito sencillo de filtro de paso bajo para filtrar la salida. PutDAC produce esta "carga" de pulsos durante un corto periodo de tiempo, después coloca el pin en un estado de alta impedancia antes de devolver el valor.

Se confía en el uso del circuito del filtro externo para mantener el voltaje de llamada a llamada. El procedimiento DACPin debería ser llamado periódicamente para refrescar el pin y mantener el voltaje dentro de valores tolerados. La tasa óptima de refresco depende de las características de circuito con el que está conectado el pin. Podría considerar llamar al procedimiento DACPin en una tarea independiente si necesita refrescar el pin de manera continua.

Consulte el apartado de **PutDAC** para ver el equivalente de punto flotante de DACPin.

## Advertencia

DACPin convierte el pin seleccionado en un pin de salida independiente de cualquier otra configuración. Así mismo, si el pin de salida no se refresca de manera periódica, no se podrá mantener el voltaje de la salida analógica.

### Ejemplo

```
Dim DACcounter As Byte

' Fijar pin 17 en 3.26 voltios = (167 * 5.0V / 256)
Call DACPin(17, 167, DACcounter)
```

# Método Debug.Print

## Sintaxis

Debug.Print *Operand*$_1$ [ ; *Operand*$_2$ ; ... *Operand*$_N$ ] [ ; ]

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand*$_N$ | String | Entrada | Operando |

## Descripción

El método Debug.Print envía uno o más cadenas desde el puerto serie Com1 a 19.200 baudios. Los múltiples parámetros de las cadenas deben estar separados por puntos y comas (;).

Se adjuntará automáticamente un par de retorno de carro/salto de línea a menos que la línea tenga un punto y coma (;) opcional al final de la línea. Un método Debug.Print vacío produce sólo un retorno de carro/salto de línea.

Debug.Print configura automáticamente el puerto Com1 para la salida. OpenCom no es necesario.

## Ejemplo

```
Debug.Print "Velocity = "; CStr(193);

Debug.Print "m/s"

Debug.Print ' Produce sólo <CR><LF>

'La salida es "Velocity = 193 m/s<CR><LF><CR><LF>
```

# Procedimiento **DefineCom3**

## Sintaxis

**Sólo BX-24, BX-35**

Call DefineCom3 (*InputPin, OutputPin, ParameterMask*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *IntputPin* | Byte | Entrada | Número de pin de entrada. |
| *OutputPin* | Byte | Entrada | Número de pin de salida. |
| *ParameterMask* | Byte | Entrada | Parámetros de comunicación (ver valores en la siguiente tabla). |

Valores permitidos para el patrón interno de bits en *ParameterMask:*

| Parámetro | Valor | Patrón de bits (x = no es relevante) |
|---|---|---|
| | | |
| Lógica invertida | &H80 | 1 0 x x x x x x |
| Lógica no invertida | &H00 | 0 0 x x x x x x |
| | | |
| Paridad par | &H30 | x x 1 1 x x x x (sólo 7 bit) |
| Paridad impar | &H20 | x x 1 0 x x x x (Sólo 7 bit) |
| Sin paridad | &H00 | x x 0 0 x x x x |
| | | |
| 7 bits de datos | &H07 | x x x x 0 1 1 1 |
| 8 bits de datos | &H08 | x x x x 1 0 0 0 |

## Descripción

DefineCom3 define los parámetros del puerto serie Com3. Este procedimiento se utiliza en conjunto con OpenCom para definir el puerto, que puede conectarse con cualquier par de pines de entrada/salida (I/O). Com3 siempre utiliza 1 bit de parada.

Si desea abrir un puerto con un solo pin (como Sólo-entrada-o Sólo-salida), puede utilizar el pin 0 como uno de los número de pin. El pin 0 se trata como un pin ficticio.

## Advertencia

La paridad no es compatible con datos de 8 bits.

### Ejemplo

```
' Definir puerto 3 para utilizar el pin 16 como entrada el 17 como
salida. También utilizar
' lógica invertida, paridad par, 7 bits de datos. 1 bit de parada
implícito.
Call DefineCom3(16, 17, bx1011_0111)

' Definir la tasa de baudios y buffers para el puerto 3.
Call OpenCom(3, 19200, InputBuffer, OutputBuffer)
```

# Procedimiento **Delay**

## Sintaxis

Call Delay(*Interval*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Interval* | Single | Entrada | Periodo de retardo. El rango es de 0.0 a 127.0 seg. La resolución es aprox. 1,95 ms. |

## Descripción

Suspende la tarea actual durante el intervalo de tiempo especificado como mínimo. Al final del retardo, la tarea estará de nuevo lista. El tiempo real que tarde la tarea en volver a ejecutarse dependerá de lo ocupado que esté el sistema con otras tareas.

Un retardo de 0.0 es una forma útil para que se ejecuten otras tareas, a la vez que permite una reanudación inmediata en caso de que no haya otras tareas que ejecutar.

El tiempo real de retardo está garantizado durante al menos el tiempo de retardo especificado, siempre que *Interval* cumpla los requisitos. Un retardo negativo se trata como un equivalente a un retardo de 0.0.

Si la tarea está bloqueada, Delay desbloqueará la tarea (ver procedimiento LockTask).

## Ejemplo

```
' Fijar el pin 16 como high
Call PutPin(16, bxOutputHigh)

' Detener momentáneamente esta tarea un mínimo de 1/2 seg, después
activarla.
Call Delay(0.5)

'Fija el pin 16 como low
Call PutPin(16, bxOutputLow)
```

# Procedimiento **DelayUntilClockTick**

## Sintaxis

Call DelayUntilClockTick

## Argumentos

Ninguno.

## Descripción

Suspende la tarea actual hasta la próxima pulsación del reloj de tiempo real (RTC). Durante la suspensión está permitida la ejecución de otras tareas. El tiempo real que tarde la tarea en volver a ejecutarse dependerá de lo ocupado que esté el sistema con otras tareas.

Si la tarea está bloqueada, este procedimiento desbloqueará la tarea (ver procedimiento LockTask).

## Ejemplo

```
' Conmutar el pin 17 a la tasa de pulsaciones de RTC.
Do
    Call DelayUntilClockTick
    Call PutPin(17, 1)
    Call DelayUntilClockTick
    Call PutPin(17, 0)
Loop
```

# Función Exp

## Sintaxis

*F* = Exp(*Operand*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Eleva *e* a la potencia especificada por el operando. La constante e (base de logaritmo natural) es aproximadamente 2.718 282.

## Ejemplo

```
Dim F As Single

F = Exp(1.0) ' F es igual a "e"
```

# Función Exp10

## Sintaxis

*F* = Exp10(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Eleva 10 a la potencia especificada por el operando.

## Ejemplo

```
Dim Y As Single

Y = Exp10(3.0) ' Y es 1000.0
```

# Función FirstTime

## Sintaxis

*F* = FirstTime( )

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *F* | Boolean | Salida | Registra si se ha llamado alguna vez la función desde la descarga del programa. |

## Descripción

Devuelve un valor Boolean que indica si esta es la primera vez que se ha llamado esta función desde la descarga del programa.

La función FirstTime es útil si desea que un programa se comporte de manera diferente la primera vez que se ejecute. Por ejemplo, puede fijar las variables persistentes con los valores iniciales para que se apliquen sólo en la primera ejecución del programa. Cuando se reinicie el procesador, puede evitar que se reinicien dichas variables, o bien puede fijarlas con otros valores.

Esto es lo que ocurre en un segundo plano – siempre que un programa se descarga, se fija una variable especial en la memoria no volátil EEPROM con un valor distinto de cero. Cuando llama a la función FirstTime, esta consulta a la variable. Si esta es distinta de cero, se eliminará la variable y la función devuelve un valor *true*. De lo contrario, la función devuelve un valor *false*. Todas las llamadas posteriores a FirstTime devolverán un valor *false*, incluso después de reiniciar el sistema.

## Ejemplo

```
Dim Setpoint As New PersistentSingle

Sub Initialize()

    If (FirstTime) Then

        ' Esta es la primera vez que se ha ejecutado el programa.
        ' Inicializar el valor inicial por defecto.
        Setpoint = 72.0
    End If

End Sub
```

# Función Fix

## Sintaxis

*F* = Fix(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Trunca un valor de punto flotante sin cambiar el tipo de los datos. La truncación es hacia 0.0.

## Ejemplo

```
Dim Y1 As Single
Dim Y2 As Single

Y1 = Fix(1.1)  ' Y1 es 1.0

Y2 = Fix(-4.9) ' Y2 es -4.0
```

# Función FixB

## Sintaxis

*F* = FixB(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Byte | Salida | Valor devuelto por función |

## Descripción

Trunca el operando de punto flotante y convierte el resultado en tipo Byte. La truncación es hacia 0.

## Ejemplo

```
Dim B1 As Byte
Dim B2 As Byte

B1 = FixB(6.4)  ' B1 es 6

B2 = FixB(-9.8) ' B2 es -9
```

# Función FixI

## Sintaxis

*F* = FixI(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Integer | Salida | Valor devuelto por función |

## Descripción

Trunca el operando de punto flotante y convierte el resultado en el tipo Integer. La truncación es hacia 0.

## Ejemplo

```
Dim I As Integer

I = FixI(-1.5) ' I es -1
```

# Función FixL

## Sintaxis

*F* = FixL(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Long | Salida | Valor devuelto por función |

## Descripción

Trunca el operando de punto flotante y convierte el resultado en un tipo Long. La truncación es hacia 0.

## Ejemplo

```
Dim L As Long

L = FixL(12.9) ' L es 12
```

# Función FixUI

## Sintaxis

*F* = FixUI(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | UnsignedInteger | Salida | Valor devuelto por función |

## Descripción

Trunca el operando del punto flotante y convierte el resultado en el tipo UnsignedInteger. La truncación es hacia 0.

## Ejemplo

```
Dim I As New UnsignedInteger

I = FixUI(-1.5) ' I es -1
```

# Función FixUL

## Sintaxis

*F* = FixUL(*Operand*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | UnsignedLong | Salida | Valor devuelto por función |

## Descripción

Trunca el operando de punto flotante y convierte el resultado en el tipo UnsignedLong. La truncación es hacia 0.

## Ejemplo

```
Dim L As New UnsignedLong

L = FixUL(5.9) ' L es 5
```

# Función FlipBits

### Sintaxis

**Sólo BX-24, BX-35**

*F* = FlipBits(*Operand*)

### Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Byte | Entrada | Operando |
| *F* | Byte | Salida | Valor devuelto por función |

### Descripción

FlipBits genera una imagen espejo de patrón de bit del operando. LSbit se convierte en MSbit y viceversa.

### Ejemplo

```
Dim A As Byte, B As Byte

A = bx11110100

B = FlipBits(A) ' B es bx00101111.
```

# Procedimiento **FreqOut** (versión flotante)

## Sintaxis

**Sólo BX-24, BX-35**

Call FreqOut(*Pin, Freq1, Freq2, Duration*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Pin* | Byte | Entrada | Número de pin de salida |
| *Freq1* | Integer | Entrada | Frecuencia 1. Las unidades están en Hz. |
| *Freq2* | Integer | Entrada | Frecuencia 2. Las unidades están en Hz. |
| *Duration* | Single | Entrada | Duración de la señal. Las unidades están en segundos. El rango es aprox. desde1.0 ms hasta 2.56 s. |

## Descripción

Genera una señal analógica que consiste de dos ondas sinusoidales superimpuestas. Se genera la señal durante la duración especificada, expresada en unidades de segundos.

## Advertencia

Este procedimiento detiene todas las capacidades multitarea durante la duración de la llamada. El reloj de tiempo real (RTC), la conmutación de tareas y el tráfico de la red quedan suspendidos durante este tiempo. Si *TimeInterval* es comparable en tamaño o mayor que el periodo de marcación del reloj de tiempo real (RTC) -aproximadamente 1,95 milisegundos-, el reloj de tiempo real perderá tiempo.

# Procedimiento **FreqOut** (versión de entero)

## Sintaxis

**Sólo BX-24, BX-35**

Call FreqOut(*Pin, Freq1, Freq2, Duration*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Pin* | Byte | Entrada | Número de pin de salida. |
| *Freq1* | Integer | Entrada | Frecuencia 1. Las unidades están en Hz. |
| *Freq2* | Integer | Entrada | Frecuencia 2. Las unidades están en Hz. |
| *Duration* | Integer | Entrada | Duración de la señal. Las unidades son ms. EL rango es desde 1 ms a 2560 ms. |

## Descripción

Genera una señal analógica que consiste en dos ondas sinusoidales. Se genera la señal durante la duración especificada, expresada en milisegundos.

## Advertencia

Este procedimiento detiene todas las capacidades multitarea durante la duración de la llamada. El reloj de tiempo real (RTC), la conmutación de tareas y el tráfico de la red quedan suspendidos durante este tiempo. Si *TimeInterval* es comparable en tamaño o mayor que el periodo de marcación del reloj de tiempo real (RTC) -aproximadamente 1,95 milisegundos-, el reloj de tiempo real perderá tiempo.

# Función Get1Wire

## Sintaxis

**Sólo BX-24, BX-35**

*F* = Get1Wire(*PinNumber*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *PinNumber* | Byte | Entrada | Número de pin. |
| *F* | Byte | Salida | Valor de bit. El rango es de 0 a 1. |

## Descripción

Recibe un solo bit utilizando el protocolo Dallas 1-Wire. El bit entra en el número de pin especificado.

# Procedimiento **GetADC** (versión flotante)

## Sintaxis

**Sólo BX-24, BX-35**

Call GetADC(*PinNumber, NondimVolt)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *PinNumber* | Byte | Entrada | Número de pin. |
| *NondimVolt* | Integer | Salida | Voltaje no dimensional. El rango es de 0.0 a 1.0. La resolución es aprox. 0.0978 %. |

## Descripción

GetADC devuelve un voltaje analógico de 10 bits. El valor devuelto es no dimensional, con un rango desde 0.0 a 1.0. Para sistemas de 5 voltios, el rango corresponde de 0.0 V a 5.0 V, con una resolución de aproximadamente 4.89 mV ( 5 / 1023 ).

Los números de los pines del convertidor ADC dependen del sistema:

Pines ADC de BX-24: De 13 a 20
Pines ADC de BX-35: De 33 a 40

Tenga en cuenta que GetADC configura automáticamente el pin para la entrada analógica. No necesitará una llamada independiente para configurar el pin en modo de entrada (Input).

## Ejemplo

```
Dim NondimVolt As Single
Const PinNumber As Byte = 13

Call GetADC(PinNumber, NondimVolt)
```

# Función GetADC (versión de entero)

### Sintaxis

**Sólo BX-24, BX-35**

*Voltage* = GetADC(*PinNumber)*

### Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *PinNumber* | Byte | Entrada | Número de pin. |
| *F* | Integer | Salida | Voltaje. El rango es de 0 a 1023. Para sistemas de 5 voltios, las unidades están en 5/1023 voltios (aprox. 4.89 mV). |

### Descripción

GetADC devuelve un voltaje analógico de 10 bits. Los números de pin de ADC dependen del sistema:

Pines del ADC de BX-24: De 13 a 20
Pines del ADC de BX-35: De 33 a 40

Tenga en cuenta que GetADC configura automáticamente el pin para la entrada analógica. No necesitará una llamada independiente para configurar el pin en modo de entrada (Input).

### Ejemplo

```
Dim Voltage As Integer
Const PinNumber As Byte = 13

Voltage = GetADC(PinNumber)
```

# Función GetBit

## Sintaxis

**Sólo BX-24, BX-35**

*F* = GetBit(*Operand, BitNumber*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Cualquier variable o matriz | Entrada | Operando |
| *BitNumber* | Byte | Entrada | Número de bit (la numeración empieza en 0). El rango es de 0 a 255. |
| *F* | Byte | Salida | Valor devuelto por función |

## Descripción

GetBit devuelve el valor del bit especificado. La numeración de los bits comienza en 0. Si el operando es una matriz, GetBit puede utilizarse para leer cualquiera de los 256 bits primeros de una matriz.

## Ejemplo

```
' Este ejemplo ilustra la función GetBit para un solo byte.

    Dim A As Byte, B As Byte, C As Byte

    A = bx00100000

    B = GetBit(A, 5) ' B es 1.
    C = GetBit(A, 6) ' C es 0.

' Este ejemplo ilustra la función GetBit para matrices Long de 32 bits.

    Dim L(1 To 2) As Long

    L(1) = 0
    L(2) = 1

    B = GetBit(L, 31) ' B es 0.
        C = GetBit(L, 32) ' C es 1 (El primer bit en el segundo elemento
    de   la matriz).
```

# Procedimiento **GetDate**

## Sintaxis

Call GetDate(*Year, Month, Day*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Year* | Integer | Salida | Año. El rango es de 1999 a 2177. |
| *Month* | Byte | Salida | Mes. |
| *Day* | Byte | Salida | Día del mes. |

## Descripción

GetDate devuelve la fecha.

# Función GetDayOfWeek

## Sintaxis

*F* = GetDayOfWeek( )

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *F* | Byte | Salida | Día de la semana. El rango es de 1 a 7 (bxSunday, bxMonday .. bxSaturday). |

## Descripción

Devuelve el día de la semana. El rango es de bxSunday a bxSaturday (de domingo a sábado)

## Advertencia

El día de la semana no estará definido hasta que la fecha del calendario esté definida. Consulte los procedimientos PutDate o PutTimestamp para definir la fecha del calendario.

# Procedimiento **GetEEPROM**

## Sintaxis

Call GetEEPROM(*Address, Value, Length*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Address* | Long | Entrada | Ubicación de inicio de la fuente en la memoria EEPROM |
| *Value* | Cualquier tipo | Entrada/Salida | Ubicación de inicio de la fuente en la memoria RAM. |
| *Length* | UnsignedInteger | Entrada | Número de bytes a transferir desde la memoria EEPROM a la RAM |

## Descripción

GetEEPROM transfiere los datos desde la memoria EEPROM a la RAM. La memoria EEPROM es donde se almacena el programa BasicX. Dado que un programa en concreto no puede utilizar toda la memoria disponible, este procedimiento le permite utilizar el resto del espacio libre para el almacenamiento de datos no volátiles.

GetEEPROM puede transferir de manera arbitraria un bloque grande de memoria en una sola operación, y el bloque puede expandirse en múltiples variables en la memoria RAM.

## Ejemplo

```
' Cada una de las cadenas requiere 22 bytes de almacenamiento.
' (20 caracteres más dos 2 bytes de sobrecarga).
Dim Name As String * 20
Dim Address As String * 20
Dim Phone As String * 20

Sub Main()

    ' Leer los datos desde la memoria EEPROM y transferir variables de
RAM.
    Call GetEEPROM(1000, Name, 22)
    Call GetEEPROM(1022, Address, 22)
    Call GetEEPROM(1044, Phone, 22)

End Sub
```

# Procedimiento **GetNetwork**

## Sintaxis (Sólo BX-01)

Call GetNetwork(*NodeAddress, MemoryAddress, Value, Result*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *NodeAddress* | UnsignedInteger | Entrada | Dirección de nodo del sistema remoto. |
| *MemoryAddress* | UnsignedInteger | Entrada | Dirección RAM de los datos que se van a copiar. Consulte los ficheros de información de los ficheros de mapas MPX si desea más información acerca de la ubicación de las variables. |
| *Value* | Cualquier tipo escalar | Salida | Destino de la copia |
| *Result* | Byte | Salida | Resultado de la operación de red. Consultar los siguientes valores permitidos. |

Valores permitidos para Resultado (*Resul)t:*

| | | |
|---|---|---|
| bxNetOk | = 0 | Sin errores (No errors) |
| bxNetNoResponse | = 1 | No se han recibido respuesta del sistema remoto (No response from remote system) |
| bxNetBusy | = 255 | Comando de red en progreso (Network command in progress) |

## Descripción

GetNetwork copia una variable escalar desde el sistema BasicX remoto a través de la red.

La tarea que ejecuta el procedimiento GetNetwork se suspenderá hasta que el sistema remoto haya recibido la transferencia de los datos, o bien se haya intentado el procedimiento un número determinado de veces. A continuación, la tarea se reinicia y se devuelve el valor del resultado.

## Errores conocidos

Ver procedimiento PutNetwork.

# Procedimiento **GetNetworkP**

## Sintaxis (Sólo BX-01)

Call GetNetworkP(*NodeAddress, MemoryAddress, Value, Result*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *NodeAddress* | UnsignedInteger | Entrada | Dirección de nodo del sistema remoto. |
| *MemoryAddress* | UnsignedInteger | Entrada | Dirección (persistente) de EEPROM de los datos que se van a copiar. Consulte los ficheros de información de los ficheros de mapas MPX si desea más información acerca de la ubicación de las variables. |
| *Value* | Cualquier tipo escalar | Salida | Destino de la copia. |
| *Result* | Byte | Salida | Resultado de la operación de red. Consultar los siguientes valores permitidos. |

Valores permitidos para Resultado (*Resul)t:*

| | | |
|---|---|---|
| bxNetOk | = 0 | Sin errores (No errors) |
| bxNetNoResponse | = 1 | No se han recibido respuesta del sistema remoto (No response from remote system) |
| bxNetBusy | = 255 | Comando de red en progreso (Network command in progress) |

## Descripción

GetNetwork copia una variable escalar desde el sistema BasicX remoto a través de la red.

La tarea que ejecuta el procedimiento GetNetwork se suspenderá hasta que el sistema remoto haya recibido la transferencia de los datos, o bien se haya intentado el procedimiento un número determinado de veces. A continuación, la tarea se reinicia y se devuelve el valor del resultado.

## Errores conocidos

Ver procedimiento PutNetwork.

# Función GetPin

## Sintaxis

*F* = GetPin(*Pin*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Pin* | Byte | Entrada | Número de pin |
| *F* | Byte | Salida | Nivel lógico (0 o 1). |

## Descripción

GetPin lee el estado de un pin de entrada/salida. GetPin se usa habitualmente junto con el procedimiento PutPin, que es el encargado de configurar el pin.

## Advertencia

Si llama a este procedimiento GetPin si haber configurado previamente el pin como entrada (Input), los resultados son totalmente inesperados. La dirección del pin puede configurarse utilizando PutPin, o bien puede utilizar los cuadros de diálogo del chip del compilador para configurar cada uno de los pines.

## Ejemplo

```
Dim PinLogicLevel As Byte

' Definir pin 16 como Entrada (Input).
Call PutPin(16, bxInputPullup)

' Leer el valor del pin 16.
PinLogicLevel = GetPin(16)
```

## Errores conocidos

En el sistema BX-01, si un pin está configurado como Input-pullup, GetPin cambiará de manera errónea el pin al modo Input-tristate si la función devuelve los siguientes valores:
**Software workaround** – justo después de GetPin, añadir una llamada a PutPin para restaurar el estado del pin como Input-pullup.
**Hardware workaround** – añadir una resistencia pullup externa al pin.

# Procedimiento **GetQueue**

## Sintaxis

Call GetQueue(*Queue, Variable, Count*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Queue* | Matriz de Bytes | Entrada/Salida | Cola desde la que se obtienen los datos. |
| *Variable* | Cualquier tipo | Salida | Destino de los datos extraídos. |
| *Count* | Integer | Entrada | Número de bytes a extraer. |

## Descripción

GetQueue obtiene los datos de una cola y coloca dichos datos en una o más variables de la memoria RAM. GetQueue puede cruzar los límites entre las variables para poder recuperar múltiples datos en una sola operación. Las variables no tienen que ser del mismo tipo al entrar y al salir (ver ejemplo)

Si no hay nada en la cola, GetQueue suspenderá la tarea actual hasta que se coloque en la cola la cantidad correcta de datos.

Las colas son un método útil para pasar datos entre tarea y tarea o bien para almacenar datos para un procesamiento futuro.

## Advertencia

Si no hay nada en la cola, y ninguna tarea coloca nada en la cola, este comando no devolverá ningún valor y la tarea se detendrá de manera indefinida.

## Ejemplo

```
Dim Oven(1 To 50) As Byte
Dim Pi As Single
Dim Fridge(1 To 4) As Byte

Sub Main()

    Call OpenQueue(Oven, 50)
    Pi = 3.14159

    ' Poner algunas Pi en el horno.
    Call PutQueue(Oven, Pi, 4)

    ' Poner cuatro piezas de Pi en la nevera.
    Call GetQueue(Oven, Fridge, 4)

End Sub
```

# Procedimiento **GetTime**

## Sintaxis

Call GetTime(*Hour, Minute, Second*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Hour* | Byte | Salida | Horas. El rango es de 0 a 23. |
| *Minute* | Byte | Salida | Minutos después de la hora. |
| *Second* | Single | Salida | Segundos. La resolución es aprox. 1.95 ms. |

## Descripción

Devuelve la hora del día en un formato de 24 horas. Los segundos se obtienen en un valor de puntos flotantes, con una resolución de 1 / 512 segundos (1.95 ms aprox.). La resolución es independiente de la hora del día.

# Procedimiento **GetTimestamp**

## Sintaxis

Call GetTimestamp(*Year, Month, Day, Hour, Minute, Second*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Year* | Integer | Salida | Año. El rango es de 1999 a 2177. |
| *Month* | Byte | Salida | Mes. |
| *Day* | Byte | Salida | Día. |
| *Hour* | Byte | Salida | Horas. El rango es de 0 a 23. |
| *Minute* | Byte | Salida | Minutos. |
| *Second* | Single | Salida | Segundos. |

## Descripción

Devuelve la fecha y la hora del día. La hora está en formato de 24 horas.

# Función **GetXIO**

## Sintaxis (Sólo BX-01)

*F* = GetXIO(*Address*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Address* | UnsignedInteger | Entrada | Dirección de entrada/salida (I/O), el rango es de 607 a 65.535 |
| *F* | Byte | Salida | Valor del puerto de entrada/salida (I/O) |

## Descripción

GetXIO recibe los datos de un puerto adicional de entrada/salida ( eXtended I/O port). BasicX admite hasta 65.536 de estos puertos I/O.

Al utilizar los mismos pines que la memoria RAM para dirigirse a los recursos de la red: la línea RD, WR y la línea de solicitud de entrada/salida (IO Request line), el sistema BasicX se dirige a los 65.536 puertos.

## Advertencia

Para el argumento *Address*, no utilice valores inferiores a 607 (&H25F).

Este comando habilita los pines RAM/XIO. Si tiene otras funciones o datos en estos pines, estos serán sustituidos.

## Ejemplo

```
Dim Value As Byte
Dim Address As New UnsignedInteger

Address = &H700

' Obtener datos del puerto.
Value = GetXIO(Address)
```

# Procedimiento **GetXRAM**

## Sintaxis (Sólo BX-01)

Call GetXRAM(*Address, Buffer, Count*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Address* | UnsignedInteger | Entrada | Dirección de inicio de la memoria RAM ampliada. El rango legal es desde 608 a 65.535. |
| *Buffer* | Cualquier tipo | Entrada/Salida | Variable o matriz de la memoria RAM en la que se copian los datos |
| *Count* | UnsignedInteger | Entrada | Número de bytes a transferir. El rango legal es desde 1 a 64.928. |

## Descripción

GetXRAM copia datos desde la memoria RAM adicional en las variables de la memoria RAM local. El tamaño de ambas memorias RAM ya sea local o adicional, es de 64 KB.

GetXRAM puede transferir de manera arbitraria un bloque grande de memoria en una sola operación, y el bloque puede expandirse en múltiples variables en la memoria RAM.

## Advertencia

Si la operación de copia desborda la memoria RAM, el sistema podría fallar.

La memoria RAM interna del chip BasicX chip ocupa direcciones dentro del rango 0 a 607 (&H25F). **Cualquier transferencia a la memoria RAM utilizada por el sistema operativo BasicX podría hacer fallar su sistema.** Por favor consulte la información sobre RAM de BasicX para ampliar la información sobre este tema.

## Ejemplo

```
Sub Main()

    Dim LocalData(1 To 20) As Single

    ' Escribir la matriz en XRAM, empezando en la ubicación
    ' 4096 (&H1000). Usar cuatro bytes por elemento
    ' para los tipos de puntos flotantes.
    Call PutXRAM( &H1000, LocalData, 20*4 )

    ' Recuperar la matriz de XRAM. La sintaxis es similar.
    Call GetXRAM( &H1000, LocalData, 20*4 )

End Sub
```

# Procedimiento **InputCapture**

## Sintaxis

Call InputCapture(*CaptureArray, NumberOfPulses, EdgeTrigger*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *CaptureArray* | Matriz de UnsignedInteger | Salida | Matriz de amplitud de pulsos. Las unidades son (1 / 7.372.800) segundos. El rango de amplitud de pulsos es de 1 a 65.535 (aprox. desde 136 ns a 8.89 ms). |
| *NumberOfPulses* | Integer | Entrada | Número de pulsos a capturar. |
| *EdgeTrigger* | Byte | Entrada | Mecanismo de disparo -- 0 significa que un flanco descendente inicia la captura, 1 significa que un flanco ascendente inicia la captura. |

## Descripción

InputCapture captura un tren de pulsos desde el pin de captura de entrada (ver definiciones de pin). Al utilizar un hardware especial dentro del chip BasicX, el procedimiento mide la amplitud de los pulsos con tolerancias muy exactas – los valores están en unidades de 1 / 7.372.800 segundos (aprox. 135.6 nanosegundos).

InputCapture suspende la llamada de la tarea hasta que se llene la matriz *CaptureArray*. El procedimiento no paraliza la máquina para que espere una entrada – es decir, otras tareas pueden ejecutarse mientras la tarea InputCapture está esperando.

En los sistemas BX-01 y BX-35, el pin de captura de entrada está siempre en estado tristate-input (alta impedancia). En los sistemas BX-24, el pin de captura de entrada se comparte con el pin 12 de entrada/salida (I/O), lo que implica que el pin 12 debería estar configurado bien como Input-tristate o Input-pullup antes de llamar InputCapture (ver procedimiento PutPin).

**Nota** – una vez que se ha capturado, el mismo tren de pulsos de *CaptureArray* puede emitirse a través del procedimiento OutputCapture.

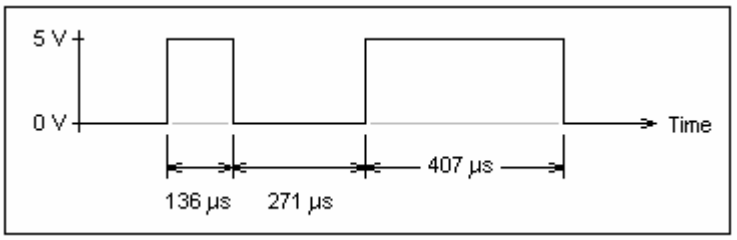
## Advertencia

InputCapture no inicia el registro hasta que se detecte una activación del flanco especificado (ascendente o ascendente). Si el flanco no tiene lugar, el procedimiento nunca devolverá el resultado.

Las pausas devuelven un valor de 65 535 (&HFFFF). Es decir, si empieza una captura, y si se produce una pausa durante uno o más pulsos, los pulsos detenidos devuelven un valor de 65 535.

InputCapture se hace cargo del temporizador Timer1. Si cualquier otra tarea o dispositivo está utilizando el temporizador Timer1, entonces se producirá un conflicto en el sistema. El puerto serie Com2 es un ejemplo de un dispositivo que utilice el temporizador Timer1.

## Ejemplo




En este ejemplo, se supone que el tren de pulsos se recibe a través del pin de captura de entrada:

```
Sub Main()

    Dim PulseTrain(1 To 3) As New UnsignedInteger

    ' Tomar 3 muestras, en la que la primera muestra se inicia con un
    flanco ascendente.
    Call InputCapture(PulseTrain, 3, 1)

    ' Después de captura, la matriz contiene aproximadamente estos
valores:
    '
    '     PulseTrain(1) = 1000 => 136 us
    '     PulseTrain(2) = 2000 => 271 us
    '     PulseTrain(3) = 3000 => 407 us

End Sub
```

Deberá tener en cuenta los anchos de los trenes de pulsos de nivel alto y bajo de la matriz *PulseTrain*.

Número de pines:

Pin de InputCapture de BX-01: 31 (PDIP)

Pin de InputCapture BX-24: 12 (compartido con el pin I/O)

Pin de InputCapture de BX-35: 20 (PDIP)

# Función LCase

## Sintaxis

*F* = LCase(*StringVar*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *StringVar* | String | Entrada | Cadena de entrada |
| *F* | String | Salida | Cadena de salida |

## Descripción

Convierte una cadena en minúsculas.

## Ejemplo

```
Dim Tx1 As String
Dim Tx2 As String

Tx1 = "ABC"
Tx2 = LCase(Tx1)  ' Tx2 es "abc"
```

# Función Len

## Sintaxis

*F* = Len(*StringVar*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *StringVar* | String | Entrada | Variable de cadena |
| *F* | Integer | Salida | Longitud de una cadena |

## Descripción

Encuentra la longitud de una cadena.

## Ejemplo

```
Dim Length As Integer
Dim Tx1 As String
Dim Tx2 As String * 10

Tx1 = "ABC"

   Length = Len(Tx1) ' La longitud de Tx1 es 3.
   Tx2 = "ABC" ' Tx2 está justificado a la izquierda, si espacios en
   blanco.
   Length = Len(Tx2) ' La longitud de Tx2 es (constante) 10.

   Tx1 = ""
   Length = Len(Tx1) ' Ahora la longitud de Tx1 es cero.
```

# Procedimiento **LockTask**

## Sintaxis

Call LockTask( )

## Argumentos

Ninguno.

## Descripción

Locktask impide que se ejecuten otras tareas (con algunas excepciones – ver abajo). BasicX sólo ejecutará la tarea actual. No se ejecutará ninguna otra tarea hasta que no se realice una llamada al procedimiento UnlockTask, Delay, Sleep o a cualquier otro procedimiento que conmute la tarea actual, ya sean llamadas de colas o llamadas del sistema, o si una interrupción de hardware activa alguna tarea.

Está permitido llamar al procedimiento LockTask si una tarea está ya bloqueada – realizar múltiples llamadas a LockTask tiene el mismo efecto que efectuar una sola llamada si una tarea está bloqueada. Por ejemplo, por regla general no se necesitan 2 llamadas a UnlockTask para anular las 2 llamadas a LockTask.

## Advertencia

Si cuando se ejecuta el comando LockTask se están ejecutando otras tareas críticas de tiempo, por lo general estas otras tareas no se ejecutarán. Por lo tanto, debe prestar atención a la hora de cooperar con otras tareas, siempre que sea necesario.

Por regla general, todas las tareas tienen la misma prioridad, aunque si no hay otra tarea bloqueada o esperando una interrupción de hardware (ver WaitForInterrupt), el evento de interrupción tendrá prioridad. La tarea bloqueada se desbloquea temporalmente y el programador de tareas reinicará la conmutación normal de tareas. Inmediatamente después de retomar la ejecución de la tarea previamente bloqueada, se volverá a bloquear de nuevo.

# Función Log

## Sintaxis

*F* = Log(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Calcula el logaritmo natural (base *e*). El valor de *e* es aproximadamente 2.718 282.

## Ejemplo

```
Dim F As Single

F = Log(20.08554) ' F es 3.0 (por ejemplo, 20.085 54 es aproximadamente
e^3)
```

# Función Log10

## Sintaxis

*F* = Log10(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Calcula la base 10 del logaritmo.

## Ejemplo

```
Dim F As Single

F = Log10(100.0) ' F es 2.0.
```

# Función MemAddress

## Sintaxis

*F* = MemAddress(*Variable*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Variable* | Cualquier tipo | Entrada | Variable o matriz |
| *F* | Integer | Salida | Dirección de argumento |

## Descripción

MemAddress devuelve la dirección de la memoria RAM del argumento. Cuando se utiliza en conjunto con RAMPeek o RAMPoke, estas funciones le permiten modificar los datos directamente, al mismo tiempo que le permite ignorar las restricciones impuestas normalmente por el lenguaje de programación.

Si la variable es una matriz, cadena o una variable de múltiples bytes, la función MemAddress devolverá la dirección del primer byte o del byte menos importante.

## Advertencia

MemAddress no debería utilizarse para direcciones superiores a 32 767, que es valor máximo legal para el tipo de función de devolución. Consulte MemAddressU i necesita trabajar con direcciones superiores.

## Ejemplo

Consulte MemAddressU para ver un ejemplo.

# Función MemAddressU

## Sintaxis

*F* = MemAddressU(*Variable*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Variable* | Cualquier tipo | Entrada | Variable o matriz |
| *F* | UnsignedInteger | Salida | Dirección de argumento |

## Descripción

MemAddressU devuelve la dirección de la memoria RAM del argumento. Cuando se utiliza en conjunto con RAMPeek o RAMPoke, estas funciones le permiten modificar los datos directamente, mientras le permite ignorar las restricciones impuestas normalmente por el lenguaje de programación.

Si la variable es una matriz, cadena o una variable de múltiples bytes, la función MemAddressU devolverá la dirección del primer byte o del byte menos importante.

## Ejemplo

```
Sub Main()

    Dim B(1 to 5) As Byte, I As Integer
    Dim Value As Byte

    ' Llenar la matriz de bytes con números pares.
    For I = 1 to 5
        B(I) = 2 * CByte(I)
    Next

    ' Leer elemento 3 de la matriz, que en realidad
    ' emite 2 bytes después del principio de la matriz
    ' en la memoria.
    Value = RAMPeek( MemAddressU(B)+2 )

    ' En este punto, el valor es 6.

End Sub
```

# Función Mid

## Sintaxis

*F* = Mid(*StringVar, Start, Length*)

Mid(*StringVar, Start, Length*) = *F*

## Argumentos (Valor devuelto por función)

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *StringVar* | String | Entrada | Cadena de origen |
| *Start* | Integer | Entrada | Inicio de la subcadena en *StringVar* |
| *Length* | Integer | Entrada | Longitud de la subcadena en *StringVar* |
| *F* | String | Salida | Cadena de destino |

## Argumentos (parte izquierda del comando)

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *StringVar* | String | Salida | Cadena de destino |
| *Start* | Integer | Entrada | Inicio de la subcadena en *StringVar* |
| *Length* | Integer | Entrada | Longitud de la subcadena en *StringVar* |

## Descripción

Mid copia una subcadena de una cadena a otra. Mid se trata de una función exclusiva que puede utilizarse a ambos lados del enunciado del comando (ver ejemplo).

## Advertencia

Si las cadenas de origen y de destino no tienen la misma longitud, la cadena de destino se truncará o se rellenarán los espacios en blanco, según sea necesario.

### Ejemplo

```
Sub Main()

    Dim Istr As String
    Dim Ostr As String

    Istr = "Time heals all wounds"
    Ostr = Istr

    Mid(Ostr, 6, 6) = Mid(Istr, 16, 6)
    Mid(Ostr, 12, 5) = Mid(Istr, 11, 5)
    Mid(Ostr, 17, 5) = Mid(Istr, 6, 5)
    Mid(Ostr, 19, 1) = "e"

    ' En este punto, Ostr = "Time wounds all heels"

End Sub
```

# Procedimiento **OpenCom**

## Sintaxis

Call OpenCom(*PortNumber, BaudRate, InputQueue, OutputQueue*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *PortNumber* | Byte | Entrada | Número de puerto serie. El rango es desde 1 a 3. |
| *BaudRate* | Long | Entrada | Tasa de baudios. Consultar los valores permitidos (abajo). |
| *InputQueue* | Matriz de bytes | Entrada/Salida | Buffer de datos para datos entrantes. |
| *OutputQueue* | Matriz de bytes | Entrada/Salida | Buffer de datos para datos salientes. |

Valores permitidos para *BaudRate:*

Para el puerto 1 (Com1) – el rango es desde 2400 a 460 800.
Para el puerto 2 (Com2) -- el rango es desde 300 a 19 200 (Sólo BX-01)
Para el puerto 3 (Com3) -- el rango es desde 300 a 19 200 (Sólo BX-24, BX-35)

## Descripción

OpenCom se utiliza para configurar e inicializar un puerto serie BasicX. El procedimiento añade dos colas al puerto – uno para la entrada y uno para salida. Debe llamar primero a OpenQueue para ambas colas antes de llamar a OpenCom. Todos los puertos utilizan 1 bit de inicio y 1 bit de parada. Los puertos 1 y 2 no usan paridad y 8 bits de datos. El puerto 3 tiene más flexibilidad en lo que respecta a la paridad, bits de datos y señales invertidas (ver DefineCom3).

Una vez que se ha abierto el puerto, los bytes colocados en la cola de salida se envían a través del puerto, y cualquier byte que llegue se colocará en la cola de entrada. Las dos colas se utilizan para la intercomunicación de datos (buffering), mientras que las entradas/salidas controladas por interrupciones tienen lugar en un segundo plano. Número de pines:

BX-01: Com1 usa los pines 10 y 11. Com2 usa los pines 1 y 12; también el temporizador Timer1.
BX-24: Com1 usa los pines 1 y 2. Com3 usa cualquier pin I/O (excepto 1 ó 2);
también el temporizador Timer2.
BX-35: Com1 usa los pines 14 y 15. Com3 usa cualquier pin I/O (excepto 14 ó 15);
también el temporizador Timer2.

## Advertencia

OpenQueue debe llamarse para las colas de entrada y salida antes de llamar a OpenCom. Si una cola de entrada se llena de bytes antes de que el programa pueda retirarlos, los bytes se perderán.

En el sistema BX-01, Com1 es también el puerto de red por lo que no puede utilizarse al mismo tiempo como puerto serie. **Si utiliza Com1 como un puerto serie en la placa de desarrollo BX-01, deberá desactivar la red** configurando el pin 14 como Input-high (Ver procedimiento PutPin).

En el sistema BX-24 y BX-35, deberá llamar DefineCom3 antes de llamar el procedimiento OpenCom para el puerto 3 (ver DefineCom3).

### Ejemplo

Consultar el fichero de ejemplo OpenComEx.

# Procedimiento **OpenNetwork**

### Sintaxis (Sólo BX-01)

Call OpenNetwork(*BoardAddress, GroupAddress*)

### Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *BoardAddress* | UnsignedInteger | Entrada | Dirección de nodo. El rango es desde 0 a 65 279 (&H0000 a &HFEFF). |
| *GroupAddress* | Byte | Entrada | Dirección de grupo. El rango es de 0 a 254 (&H00 a &HFE). |

### Descripción

Define la red y la dirección de grupo del chip local de BasicX Chip y habilita el acceso a través de los chips BasicX remotos.

Si selecciona la red a través del sistema de descarga de BasicX, la red se iniciará automáticamente. En este caso OpenNetwork no es necesario.

Los datos recibidos a través de la red deben estar dirigidos de manera que vayan dirigidos al chip BasicX correcto o al grupo de chips BasicX correcto.

Algunas direcciones de nodo tienen un significado especial:

`&HFFFF` – Trasmite este mensaje a **todos** los chips BasicX
`&HFFxx` – Trasmite este mensaje a todos los chips BasicX miembros del grupo &Hxx

### Advertencia

Cada chip BasicX de la red debe tener una dirección única. Abrir un chip BasicX conectado en red a la misma dirección de que otro chip BasicX podría causar problemas.

### Ejemplo

```
' Esta llamada le permite recibir todos los paquetes dirigidos a la
' dirección BoardAddress 1234h. También recibiremos los mensajes de
grupo ' dirigidos a GroupAddress 32h
Call OpenNetwork (&H1234, &H32)
```

# Procedimiento **OpenQueue**

## Sintaxis

Call OpenQueue(*Queue, Size*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Queue* | Matriz de datos | Entrada/Salida | Matriz utilizada para crear la cola. |
| *Size* | Integer | Entrada | Tamaño (en bytes) de la cola (*Queue*). El mínimo es 10 bytes. |

## Descripción

Crea una cola a partir de una matriz de bytes.

Las colas son estructuras de datos que tienen propiedades especiales. Las colas actúan como elementos de almacenamiento de datos que pueden llenarse y vaciarse a través de tareas. Se utiliza un código especial dentro del chip BasicX para transferir automáticamente los datos de la cola de tarea a tarea. Internamente, una cola es implementada como si se tratase de un buffer circular, y los punteros para la cola están dentro de la cola propiamente dicha. Al abrir la cola se inicializan los punteros. La sobrecarga del puntero interno requiere 9 bytes, por lo que si define una matriz de colas de 20 bytes (por ejemplo), solamente tendrá 11 bytes libres para los datos.

## Advertencia

Las colas deben ser lo suficientemente grandes para poder aceptar los elementos de datos colocados en ellas, además de los 9 bytes necesarios para la sobrecarga interna. La cola más pequeña permitida es 10 bytes.

## Ejemplo

```
Dim ICom2(1 to 30) As Byte
Dim OCom2(1 to 30) As Byte

Sub Main()

    Dim Ch As Byte

    ' Abrir colas de entrada y salida.
    Call OpenQueue(ICom2, 30)
    Call OpenQueue(OCom2, 30)

    ' Abrir puerto serie y añadir ambas colas al puerto.
    Call OpenCom(2, 19200, ICom2, OCom2)

    Call PutQueueStr(OCom2, "Hello World!")

End Sub
```

# Procedimiento **OpenSPI**

## Sintaxis

Call OpenSPI(*Channel, SetupByte, PinNumber*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Channel* | Byte | Entrada | Número de canal. El rango es de 1 a 4. |
| *SetupByte* | Byte | Entrada | Un byte utilizado para inicializar el puerto SPI antes de acceder al dispositivo utilizando el comando SPICmd. Ver a continuación el formato. |
| *PinNumber* | Byte | Entrada | El número de pin del pin utilizado para seleccionar el dispositivo SPI en el chip. El selector de chip se trata de una señal activa de nivel lógico bajo. |

Formato de *SetupByte:*

```
Nombre          Acción

SPI_LSB         LSB se transmite primero
SPI_CPOL        SCK es alto (high) cuando está inactivo
SPI_CPHA        Consultar tabla


SPI_    SPI_
CPHA    CPOL    Resultado

 0       0      Flanco ascendente en medio de la celda de bits
 0       1      Flanco descendente en medio de la celda de bits
 1       0      Flanco descendente en medio de la celda de bits
 1       1      Flanco ascendente en medio de la celda de bits

SPI_SCK04       SCK = CLK / 4
SPI_SCK16       SCK = CLK / 16
SPI_SCK64       SCK = CLK / 64
SPI_SCK128      SCK = CLK / 128
```

## Descripción

BasicX tiene un bus de interfaz periférica serie (Serial Peripheral Interface - SPI) integrado en el hardware del chip. Utilizando este bus, los periféricos de otros fabricantes como Motorola y National Semiconductor pueden ser utilizados para funciones especiales imposibles de realizarse directamente a través del chip BasicX.

El comando OpenSPI ofrece al programador la capacidad de tener 4 dispositivos SPI conectados al chip BasicX.

El bus SPI puede configurarse de modos diferentes: polaridad, fase del reloj y velocidad. El comando OpenSPI le permite configurar cada canal de manera independiente a los otros dispositivos.

## Ejemplo

```
Dim SetupByte As Byte

SetupByte = SPI_CPHA or SPI_SCK64
Call OpenSPI( 3, SetupByte, 16 )
```

# Procedimiento **OpenWatchdog**

## Sintaxis

Call OpenWatchdog(*TimeoutValue*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *TimeoutValue* | Byte | Entrada | El valor TimeoutValue N es ($16 * 2^N$), el retardo de tiempo expresado en ms, donde N tiene un rango de 0 a 7 (ver abajo). |

Valores permitidos para *TimeoutValue:*

| | |
|---|---|
| 0 | = 16 milisegundos |
| 1 | = 32 ms |
| 2 | = 64 ms |
| 3 | = 128 ms |
| 4 | = 256 ms |
| 5 | = 512 ms (aproximadamente 1/2 segundos) |
| 6 | = 1024 ms (aproximadamente 1 segundo) |
| 7 | = 2048 ms (aproximadamente 2 segundos) |

## Descripción

OpenWatchdog inicia el temporizador watchdog, que reiniciará el procesador a menos que el temporizador se refresque periódicamente.
¿Qué es el temporizador watchdog? En algunos casos una aplicación es tan crítica que desea mantenerla en funcionamiento en prácticamente cualquier condición. Si un programa se bloquea o deja de funcionar por algún motivo – por ejemplo, es posible que se ejecute una ruta de acceso imprevista, o que un pico eléctrico haga que los datos sean ilegibles. Ene ste caso puede reiniciarse la opción de seguridad del temporizador *watchdog* para arrancar el procesador. El temporizador cuenta hacia a atrás hasta llegar al valor predefinido. Si el temporizador no se refresca antes de que termine *TimeoutValue*, el procesador se reseteará.

El procedimiento OpenWatchdog inicia el temporizador watchdog. A continuación, se supone que el programa activa de manera periódica al llamar al procedimiento Watchdog. Esta llamada se inserta normalmente en una sección de código crítico que se ejecuta periódicamente. Si el programa no funciona correctamente y no ejecuta el código, el temporizador (en condiciones normales) nunca se refrescará. El watchdog entonces reseteará y reiniciará el procesador después de que finalice el periodo de inactividad, y el programa empezará desde el principio.

## Advertencia

Por razones de seguridad, el sistema operativo BasicX no incluye previsiones para desactivar un temporizador watchdog una vez activado. Así mismo, tenga en cuenta que el temporizador watchdog podría interferir en la descarga de nuevos programas – si el temporizador watchdog está activo, es posible que sea necesario reiniciar el sistema siempre que descargue un nuevo programa.

## Ejemplo

Consultar el fichero de ejemplo WatchDogEx.bas.

# Procedimiento **OutputCapture**

## Sintaxis

Call OutputCapture(*CaptureArray, NumberOfPulses, StartingEdge*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *CaptureArray* | Matriz de UnsignedInteger | Entrada | Matriz de ancho de pulsos. Las unidades están en 1 / 7 372 800 segundos (aprox. 135.6 ns). |
| *NumberOfPulses* | Integer | Entrada | Número de pulsos generados. |
| *StartingEdge* | Byte | Entrada | Tipo de flanco de pulsos de inicio. El flanco descendente es 0, el flanco ascendente es 1. |

## Descripción

OutputCapture genera un tren de pulsos del pin de captura de salida (ver definiciones de los pines). Al utilizar un hardware especial dentro del chip BasicX, el procedimiento genera unos anchos de pulsos con tolerancias muy precisas – los valores están en unidades de 1 / 7 372 800 segundos (aprox. 135.6 nanosegundos).

OutputCapture suspende la tarea llamada hasta que *CaptureArray* esté terminada. El procedimiento OutputCapture es un método útil para reproducir un tren de pulsos detectado por el procedimiento InputCapture.

## Advertencia

El procedimiento OutputCapture se encarga del temporizador Timer1. Si cualquier otra tarea o dispositivo está utilizando el temporizador Timer1, entonces se producirá un conflicto. El puerto serie Com2 es un ejemplo de dispositivo que utiliza el Timer1.

## Ejemplo

```
Sub Main()

    Dim PulseTrain(1 To 3) As New UnsignedInteger

    PulseTrain(1) = 1000  ' (1000 / 7 372 800) = 136 microsegundos
    PulseTrain(2) = 2000  ' (2000 / 7 372 800) = 271 microsegundos
    PulseTrain(3) = 3000  ' (3000 / 7 372 800) = 407 microsegundos

    ' Genera los 3 pulsos, empezando con un flanco ascendente.
    Call OutputCapture(PulseTrain, 3, 1)

End Sub
```


Este ejemplo produce el siguiente tren de pulsos en el pin de captura de salida:

Pin OutputCapture de BX-01: 29 (PDIP)

Pin OutputCapture de BX-24: 27

Pin OutputCapture de BX-35: 18 (PDIP)

# Procedimiento **PeekQueue**

## Sintaxis

Call PeekQueue(*Queue, Variable, Count*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Queue* | Matriz de bytes | Entrada | Cola de la que se copian los datos. |
| *Variable* | Cualquier tipo | Salida | Destino de los datos copiados. |
| *Count* | Integer | Entrada | Número de bytes copiados. |

## Descripción

PeekQueue copia los datos desde una cola hasta las variables de la memoria RAM, sin quitar realmente los datos de la cola. PeekQueue puede cruzar los límites entre variables para poder copiar múltiples datos en una sola operación. Las variables no tienen que ser del mismo tipo al entrar y al salir.

Si no hay nada en la cola, PeekQueue suspenderá la tarea actual hasta que se coloque la cantidad de datos correcta en la cola.

## Advertencia

Si no hay nada en la cola, y no ninguna tarea coloca ningún dato en la cola, el procedimiento no devolverá ningún valor y la tarea se detendrá de manera indefinida.

## Ejemplo

Ver el fichero de ejemplo PeekQueueEx.bas.

# Función PersistentPeek

## Sintaxis

*F* = PersistentPeek(*Address*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Address* | UnsignedInteger | Entrada | Dirección del origen de los datos, el rango es de 32 a 511 |
| *F* | Byte | Salida | Destino de los datos copiados. |

## Descripción

PersistentPeek lee un byte de los datos ubicados en la memoria persistente.

Hay 480 bytes de memoria persistente, ubicados en las direcciones desde 32 a 511.

## Ejemplo

```
Dim Data As Byte

' Leer los datos de EEPROM en la dirección 1234.
Data = PersistentPeek(1234)
```

# Procedimiento **PersistentPoke**

## Sintaxis

Call PersistentPoke(*Value, Address*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Value* | Byte | Entrada | Dirección de destino, el rango es de 32 a 511. |
| *Address* | UnsignedInteger | Entrada | Origen de los datos copiados. |

## Descripción

PersistentPoke escribe un byte de datos en una ubicación de la memoria persistente.

Hay 480 bytes de memoria persistente, ubicados en las direcciones de 32 a 511.

## Advertencia

Escribir en direcciones fuera del rango legal podría ocasionar un fallo en el sistema.

Nota – la memoria persistente se implementa en la memoria EEPROM, que tiene limitado en número de veces que puede escribirse antes de que se vuelva inservible. Los límites de escritura normales son entre las 100.000 a 1.000.000 veces. Asegúrese de que su programa no está bloqueado en un bucle rápido escribiendo en la memoria persistente o se destruirá rápidamente.

## Ejemplo

```
Dim Data As Byte

' Escribir el valor 65 en la dirección de 1234 de EEPROM.
Data = 65
Call PersistentPoke(Data, 1234)
```

# Procedimiento **PlaySound**

## Sintaxis

**Sólo BX-24, BX-35**

Call PlaySound(*Pin, StartAddress, Length, SampleRate, RepeatCount*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Pin* | Byte | Entrada | Número de pin de salida. |
| *StartAddress* | UnsignedInteger | Entrada | Dirección de inicio de los datos en EEPROM. |
| *Length* | UnsignedInteger | Entrada | Longitud de los datos. Las unidades son bytes. |
| *SampleRate* | UnsignedInteger | Entrada | Tasa de muestras. Las unidades son en Hz. |
| *RepeatCount* | UnsignedInteger | Entrada | Número de veces que se repetirá el sonido. |

## Descripción

PlaySound genera un sonido de los datos sampleados en la memoria EEPROM.

## Advertencia

Este procedimiento detiene todas las capacidades multitarea durante la duración de la llamada. El reloj de tiempo real (RTC), la conmutación de las tareas y el tráfico de la red se suspenden durante este tiempo. Si la combinación de *Length*, *SampleRate* y *RepeatCount* es tal que la duración del sonido supera los 1.95 ms, el reloj de tiempo real (RTC) perderá tiempo.

# Función Pow

## Sintaxis

*F* = Pow(*Operand, Exponent*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *Exponent* | Single | Entrada | Exponente |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Eleva el operando a la potencia especificada por el exponente.

## Ejemplo

```
Dim F As Single

F = Pow(10.0, 3.0) ' F = 10^3 = 1000.0
```

## Errores conocidos

La devolución de la función es incorrecta si el operando es negativo y el exponente tiene un valor entero.

# Procedimiento **PulseIn** (versión flotante)

## Sintaxis

Call PulseIn(*Pin, State, PulseWidth*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Pin* | Byte | Entrada | Número de pin |
| *State* | Byte | Entrada | Especifica un pulso alto (1) o bajo (0) |
| *PulseWidth* | Single | Salida | Intervalo de tiempo. Las unidades están expresadas en segundos. EL rango válido es aprox. desde 1.085 µs a 71.1 ms. La pausa devuelve un valor 0.0. |

## Descripción

Mide el ancho de un pulso en el pin I/O especificado.

PulseIn espera la transición al estado definido, a continuación mide la duración del pulso hasta que se cambie de estado o bien se detenga. PulseIn se detiene aproximadamente en 71 milisegundos y devuelve un valor 0.0 para *PulseWidth*.

La resolución de *PulseWidth* es aproximadamente 1.085 µs.

## Advertencia

PulseIn dedica el procesador para buscar los pulsos. El reloj de tiempo real (RTC), la conmutación de las tareas y el tráfico de la red se suspenden durante este tiempo. Los pulsos de entrada superiores a 1.95 ms, podrían hacer que el reloj de tiempo real (RTC) pierda tiempo.

## Ejemplo

```
Dim PulseWidth As Single

' Esperar un pulso alto en el pin 16.
Call PulseIn(16, 1, PulseWidth)
```

# Función PulseIn (Versión de entero)

## Sintaxis

*PulseWidth* = PulseIn(*Pin, State*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Pin* | Byte | Entrada | Número de pin |
| *State* | Byte | Entrada | Especifica un pulso alto (1) o bajo (0) |
| *PulseWidth* | Integer | Salida | Intervalo de tiempo, en unidades de 8 / 7 372 800 segundos (aprox. 1.085 µs). El rango válido es de 1 a 32 767 unidades. El intervalo devuelve un valor cero o un valor negativo. |

## Descripción

Mide el ancho de un pulso en el pin I/O especificado.

PulseIn espera la transición al estado definido, a continuación mide la duración del pulso hasta que se cambie de estado o bien se detenga. PulseIn se detiene aproximadamente en 35.5 ms y devuelve un valor cero o negativo para *PulseWidth*.

## Advertencia

PulseIn dedica el procesador para buscar los pulsos. El reloj de tiempo real (RTC), la conmutación de las tareas y el tráfico de la red se suspenden durante este tiempo. Los pulsos de entrada superiores a un recuento de 1800, podrían hacer que el reloj de tiempo real (RTC) pierda tiempo.

## Ejemplo

```
Dim PulseWidth As Integer

' Espera un pulso alto en el pin 16.
PulseWidth = PulseIn(16, 1)
```

# Procedimiento **PulseOut** (versión flotante)

## Sintaxis

Call PulseOut(*Pin, PulseWidth, State*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Pin* | Byte | Entrada | Número de pin |
| *PulseWidth* | Single | Entrada | Intervalo de tiempo. Las unidades están expresadas en segundos, el rango es de aprox. 1.085 µs a 71.1 ms. |
| *State* | Byte | Entrada | Especifica un pulso alto (1) o bajo (0) |

## Descripción

PulseOut envía un pulso de nivel lógico alto o bajo desde cualquier pin de entrada/salida disponible. El procedimiento espera hasta que se envíe el pulso antes de volver.

La resolución de PulseOut es 8 / 7 372 800 segundos (aprox. 1.085 µs).

Nota -- PulseOut puede utilizarse por separado como medio de generar un retardo – es decir, sin afectar los pines físicos de entrada/salida. Esto se hace utilizando el pin 0 como el parámetro de los pines. El pin 0 se trata como si fuera un pin ficticio.

## Advertencia

Este procedimiento detiene todas las capacidades de multitarea durante la duración de la llamada. El reloj de tiempo real (RTC), la conmutación de tareas y el tráfico de la red quedan suspendidos durante este tiempo. Los pulsos de salida superiores a 1.95 milisegundos se traducirán en una pérdida de tiempo del reloj de tiempo real (RTC).

Así mismo, el comportamiento de PulseOut es indefinido si *PulseWidth* viola las limitaciones del rango.

## Ejemplo

```
' Envía un pulso alto al pin 17. El ancho de pulsos es 1.5 ms.
Call PulseOut(17, 1.5E-3, 1)
```

# Procedimiento **PulseOut** (versión de entero)

## Sintaxis

Call PulseOut(*Pin, PulseWidth, State*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Pin* | Byte | Entrada | Número de pin |
| *PulseWidth* | UnsignedInteger | Entrada | Intervalo de tiempo, en unidades de 8 / 7 372 800 segundos (aprox. 1.085 µs). El rango es de 1 a 65 535 unidades. |
| *State* | Byte | Entrada | Especifica un pulso alto (1) o bajo (0) |

## Descripción

PulseOut envía un pulso de nivel lógico alto o bajo desde cualquier pin de entrada/salida disponible. El procedimiento espera hasta que se envíe el pulso antes de volver.

La resolución de PulseOut es 8 / 7 372 800 segundos (aprox.1.085 µs).

Nota – Nota -- PulseOut puede utilizarse por separado como medio de generar un retardo – es decir, sin afectar los pines físicos de entrada/salida. Esto se hace utilizando el pin 0 como el parámetros de los pines. El pin 0 se trata como si fuera un pin ficticio.

## Advertencia

Este procedimiento detiene todas las capacidades de multitarea durante la duración de la llamada. El reloj de tiempo real (RTC), la conmutación de tareas y el tráfico de la red quedan suspendidos durante este tiempo. Los pulsos de salida superiores a 1.95 milisegundos se traducirán en una pérdida de tiempo del reloj de tiempo real (RTC).

Así mismo, el comportamiento de PulseOut es indefinido si *PulseWidth* viola las limitaciones del rango.

## Ejemplo

```
Dim PulseWidth As Integer

' El ancho de los pulsos es 1.5 ms.
PulseWidth = 1382 ' Conversión de unidad: 1.5E-3/1.085E-6 = 1382

' Enviar un pulso alto al pin 17.
Call PulseOut(17, PulseWidth, 1)
```

# Procedimiento **Put1Wire**

## Sintaxis

**Sólo BX-24, BX-35**

Call Put1Wire(*PinNumber, BitValue*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *PinNumber* | Byte | Entrada | Número de pin. |
| *BitValue* | Byte | Entrada | Valor de bit. El rango es de 0 a 1. |

## Descripción

Transmite un solo bit utilizando el protocolo Dallas 1-Wire. Se transmite el bit en el número de pin especificado.

# Procedimiento **PutBit**

## Sintaxis

**Sólo BX-24, BX-35**

Call PutBit(*Operand, BitNumber, Value*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Cualquier variable o matriz | Entrada | Destino del bit. |
| *BitNumber* | Byte | Entrada | Número de bit (la numeración empieza en 0). El rango es de 0 a 255. |
| *Value* | Byte | Entrada/Salida | Valor del bit. El rango es de 0 a 1. |

## Descripción

PutBit configura el bit especificado al estado definido por *Value (Valor)0*. La numeración de los bits empieza en 0. Si el operando es una matriz, PutBit puede escribir en cualquier de los primeros 256 bits de la matriz.

## Ejemplo

```
' Ejemplo de PutBit para un byte sencillo.

    Dim A As Byte, B As Byte, C As Byte

    A = bx00100000

    Call PutBit(A, 2, 1) ' Aquí A = bx00100100
    Call PutBit(A, 5, 0) ' Aquí A = bx00000100

' Ejemplo de PutBit para una matriz Long de 32 bits.

    Dim L(1 To 2) as Long

    L(2) = 0

    ' Definir el primer bit del segundo elemento.
    Call PutBit(L, 32, 1)  ' Aquí, L(2) = 1.
```

# Procedimiento **PutDAC**

## Sintaxis

Call PutDAC(*Pin, NondimVolt, DACcounter*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Pin* | Byte | Entrada | Número de pin. |
| *NondimVolt* | Single | Entrada | Voltaje no dimensional. El rango es de 0.0 a 1.0. La resolución es aproximadamente 0.392 %. |
| *DACcounter* | Byte | Entrada/Salida | DACcounter es el valor que debe devolverse cada vez que se llame a la rutina de modo que el conversor DAC siga sincronizado. Si tiene múltiples convertidores DAC en ejecución simultanea, entonces deberá tener un DACcounter diferente para cada pin. |

## Descripción

PutDAC genera un voltaje pseudo-analógico de 8 bits en un pin de entrada/salida (/O). En sistemas de 5 voltios, el rango de voltaje es desde 0.0 V a 5.0 V, con una resolución de aproximadamente 19.6 mV.

Se temporiza un grupo rápido de pulsos de manera precisa para producir el voltaje deseado. Externamente, se necesita un circuito de filtro de paso bajo para filtrar la salida. PutDAC produce esta "explosión" de pulsos durante poco tiempo, y después coloca el pin en un estado de alta impedancia antes de devolver.

Se confía en el circuito de filtro externo para mantener el voltaje entre las llamadas. PutDAC debería llamarse de manera periódica para refrescar el pin y mantener el voltaje dentro de unas tolerancias. La tasa de refresco óptima depende de las características del circuito al que está conectado el pin. Puede considerar llamando a PutDAC en una tarea independiente si necesita refrescar el pin de manera continua.

Ver **DACPin** para el equivalente entero de PutDAC.

## Advertencia

PutDAC convierte el pin seleccionado en un pin de salida pin independiente de cualquier otra configuración. Así mismo, si el pin de salida no se refresca de manera periódica, no se mantendrá el voltaje analógico de salida.

## Ejemplo

```
Dim DACcounter As Byte
Const Pin As Byte = 16

' Establecer el pin 16 al 75 por ciento de la escala completa.
Call PutDAC(Pin, 0.75, DACcounter)
```

# Procedimiento **PutDate**

## Sintaxis

Call PutDate(*Year, Month, Day*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Year* | Integer | Entrada | Año. El rango es de 1999 a 2177. |
| *Month* | Byte | Entrada | Mes. |
| *Day* | Byte | Entrada | Día del mes. |

## Descripción

Establece la fecha. El día de la semana también se define automáticamente al llamar a PutDate (ver la función GetDayOfWeek).

# Procedimiento **PutEEPROM**

## Sintaxis

Call PutEEPROM(*Address, Value, Length*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Address* | Long | Entrada | Ubicación de inicio del destino en la memoria EEPROM. |
| *Value* | Cualquier tipo | Entrada | Ubicación de inicio del origen en la memoria RAM. |
| *Length* | Integer | Entrada | Número de bytes a transferir desde RAM hasta EEPROM. |

## Descripción

PutEEPROM transfiere los datos desde la memoria RAM a la memoria EEPROM. La memoria EEPROM es donde se almacena el programa BasicX. Dado que un programa en particular podría no utilizar toda la memoria disponible, PutEEPROM le permite utiliza el espacio restante para el almacenamiento de datos no volátiles.

PutEEPROM puede transferir un bloque grande de memoria de forma arbitraria en una sola operación; además el bloque puede aplicarse a múltiples variables en la memoria RAM.

## Advertencia

Escribir en el espacio de código en la memoria EEPROM puede corromper un programa en ejecución. Cualquier escritura debería hacerse en las direcciones más allá del final del programa. A fin de determinar la última dirección ocupada por el código, consulte la sección de memoria de código en el fichero de mapa MPP (generado al compilar un programa).

Tenga en cuenta que las memorias EEPROM tienen limitado el número de veces que se puede escribir antes de que se vuelvan inutilizables. En condiciones normales, los límites de escritura están entre 100.000 y 1.000.000. Compruebe que su programa no está bloqueado en bucle rápido escribiendo en la memoria EEPROM o se destruirá inmediatamente.

## Ejemplo

```
Dim Name As String * 20
Dim Address As String * 20
Sub Main()
    Name = "W.C.Fields"
    Address = "Chattanooga"
    ' Copiar 2 cadenas a 22 bytes por cadena (20
    ' caracteres más 2 bytes de sobrecarga por cadena).
    Call PutEEPROM(1000, Name, 22)
    Call PutEEPROM(1022, Address, 22)

End Sub
```

# Procedimiento **PutNetwork**

## Sintaxis (Sólo BX-01)

Call PutNetwork(*NodeAddress, MemoryAddress, Value, Result*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *NodeAddress* | UnsignedInteger | Entrada | Dirección de nodo del sistema remoto. |
| *MemoryAddress* | UnsignedInteger | Entrada | Dirección de la RAM de los datos a escribir. Consulte el apartado de los ficheros de mapas MPX si desea obtener más información acerca de las ubicaciones de las variables. |
| *Value* | Cualquier tipo escalar | Entrada | Origen de la copia. |
| *Result* | Byte | Salida | Resultado de la operación de red. Consultar los valores permitidos de la siguiente tabla. |

Valores permitidos para Resultado (*Result):*

| | | |
|---|---|---|
| bxNetOk | = 0 | Sin errores (No errors) |
| bxNetNoResponse | = 1 | No se han recibido respuesta del sistema remoto (No response from remote system) |
| bxNetBusy | = 255 | Comando de red en progreso (Network command in progress) |

## Descripción

PutNetwork copia una variable escalar en la ubicación de RAM especificada en un sistema BasicX.

La tarea que ejecuta el procedimiento PutNetwork se suspenderá hasta que el sistema remoto haya recibido la transferencia de los datos, o bien se haya intentado el procedimiento un número determinado de veces. A continuación, la tarea se reinicia y se devuelve el valor del resultado.

## Advertencia

Debe enviar los datos al sistema remoto con mucho cuidado. Si no envía los datos a la ubicación correcta, los datos del sistema remoto podrían corromperse y hacer que el sistema se vuelva inestable.

## Errores conocidos

Si otro nodo de red intenta transmitir un paquete de red simultáneamente, es posible que el procesador se quede bloqueado

# Procedimiento **PutNetworkP**

## Sintaxis (Sólo BX-01)

Call PutNetworkP(*NodeAddress, MemoryAddress, Value, Result*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *NodeAddress* | UnsignedInteger | Entrada | Dirección de nodo del sistema remoto. |
| *MemoryAddress* | UnsignedInteger | Entrada | Dirección de la EEPROM de los datos a escribir. Consulte el apartado de los ficheros de mapas MPX si desea más información acerca de la ubicación de las variables. |
| *Value* | Cualquier tipo escalar | Entrada | Origen de la copia. |
| *Result* | Byte | Salida | Resultado de la operación de red. Consulte los valores permitidos en la tabla siguiente. |

Valores permitidos para Resultado (*Resul)t:*

| | | |
|---|---|---|
| bxNetOk | = 0 | Sin errores (No errors) |
| bxNetNoResponse | = 1 | No se han recibido respuesta del sistema remoto (No response from remote system) |
| bxNetBusy | = 255 | Comando de red en progreso (Network command in progress) |

## Descripción

PutNetworkP copia una variables escalar en la ubicación de la memoria EEPROM (persistente) especificada en el sistema BasicX.

La tarea que ejecuta el procedimiento PutNetworkP se suspenderá hasta que el sistema remoto haya recibido la transferencia de los datos, o bien se haya intentado el procedimiento un número determinado de veces. A continuación, la tarea se reinicia y se devuelve el valor del resultado.

## Advertencia

Debe enviar los datos al sistema remoto con mucho cuidado. Si no envía los datos a la ubicación correcta, los datos del sistema remoto podrían corromperse y hacer que el sistema se vuelva inestable.

## Errores conocidos

Ver procedimiento PutNetwork.

# Procedimiento **PutNetworkPacket**

## Sintaxis (Sólo BX-01)

Call PutNetworkPacket(*Packet, Result*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Packet* | Matriz de bytes | Entrada | Paquete |
| *Result* | Byte | Salida | Resultado de la operación de red. Consultar los valores permitido en la siguiente tabla. |

Valores permitidos para Resultado (*Result):*

| | | |
|---|---|---|
| bxNetOk | = 0 | Sin errores (No errors) |
| bxNetNoResponse | = 1 | No se han recibido respuesta del sistema remoto (No response from remote system) |
| bxNetBusy | = 255 | Comando de red en progreso (Network command in progress) |

## Descripción

PutNetworkPacket es un comando utilizado normalmente por las funciones del sistema operativo que requieren en envío de un paquete de red un formato en concreto.

El procedimiento PutNetworkPacket presupone que conoce el formato del paquete y que ha creado un paquete formateado en la memoria, que se enviará directamente sin supervisión del sistema operativo.

## Advertencia

Este procedimiento lo utilizan las funciones del sistema operativo y normalmente ningún otro programa lo utiliza.

## Errores conocidos

Ver procedimiento PutNetwork.

# Procedimiento **PutNetworkQueue**

## Sintaxis (Sólo BX-01)

Call PutNetworkQueue(*NodeAddress, QueueAddress, Value, Result*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *NodeAddress* | UnsignedInteger | Entrada | Dirección del sistema remoto. |
| *QueueAddress* | UnsignedInteger | Entrada | Dirección de la memoria RAM de la cola del sistema remoto. Consulte el apartado de los ficheros de mapas MPX si desea obtener más información acerca de las ubicaciones de las variables. |
| *Value* | Cualquier tipo | Entrada | Los datos que se van a colocar en la cola remota. |
| *Result* | Byte | Salida | Resultado de una operación de red. Consulte los valores permitidos en la siguiente tabla. |

Valores permitidos para Resultado (*Resul)t:*

| | | |
|---|---|---|
| bxNetOk | = 0 | Sin errores (No errors) |
| bxNetNoResponse | = 1 | No se han recibido respuesta del sistema remoto (No response from remote system) |
| bxNetBusy | = 255 | Comando de red en progreso (Network command in progress) |

## Descripción

PutNetworkQueue coloque los datos en la cola de un sistema remoto a través de la red.

Este procedimiento es útil para los casos en los que múltiples sistemas BasicX envían los datos a un nodo común. Un ejemplo es un sistema de seguridad en el que la estación central estará a cargo de la monitorización de distintos eventos. Los sistemas remotos pueden enviar datos a una cola de la estación central siempre que se produzcan dichos eventos.

PutNetworkQueue coloca la dirección del nodo de transmisión en la cola delante de los datos. De esta manera el equipo remoto sabe quién envía los datos. Si el sistema remoto no necesita estos datos, el sistema debe extraerlos y descartarlos.

## Advertencia

Debe tener cuidado al enviar los datos a un sistema remoto. Si no envía los datos a la ubicación correcta, los datos del sistema remoto podrían corromperse y hacer que el sistema se vuelva inestable.

## Errores conocidos

Ver procedimiento PutNetwork.

# Procedimiento **PutPin**

## Sintaxis

Call PutPin(*Pin, State*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Pin* | Byte | Entrada | Número de pin. |
| *State* | Byte | Salida | Estado del pin. Consulte los valores permitidos en la siguiente tabla. |

Valores permitidos para Estado (*State):*

| | | |
|---|---|---|
| bxInputLow | = 0 | Salida de nivel lógico bajo (normalmente 0 voltios) |
| bxOutputHigh | = 1 | Salida de nivel lógico alto (normalmente 5 voltios) |
| bxInputTristate | = 2 | Tristate (Z, o alta impedancia) |
| bxInputPullup | = 3 | Pull-up (P, on-chip 120 k-Ohm pull-up) |

## Descripción

PutPin configura un pin de entrada/salida (I/O) con el estado Output low (0), Output high (1), Input tristate (Z), o Input pull-up (P).

PutPin le proporciona un control total sobre el estado de un pin. Puede transmitir un valor lógico alto o bajo. Así mismo, puede configurar el pin como tristate, que también es alta impedancia. Esto es gran utilidad en las comunicaciones con un bus bidireccional. El cuarto estado es pull-up, el cual conecta una resistencia pull-up en chip de aproximadamente 120 kΩ. Este estado es de gran utilidad al leer los datos de un dispositivo pasivo como por ejemplo un interruptor (switch).

PutPin normalmente se utiliza en conjunto con la función GetPin, en la que PutPin se utiliza para definir el estado del pin antes de leerlo.

## Ejemplo

```
' Configurar el pin de entrada/salida (I/O) 7 con un nivel lógico alto
(high), después esperar 1/2 segundo antes de volver a bajarlo (low).
Call PutPin(17, bxOutputHigh)
Call Sleep(0.5)
Call PutPin(17, bxInputLow)
```

# Procedimiento **PutQueue**

## Sintaxis

Call PutQueue(*Queue, Variable, Count*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Queue* | Matriz de bytes | Entrada/Salida | Cola en la que se insertan los datos. |
| *Variable* | Cualquier tipo | Entrada | Datos a insertar en la cola. |
| *Count* | Integer | Entrada | Número de bytes a insertar. |

## Descripción

PutQueue copia datos desde las variables de la memoria RAM a una cola. PutQueue puede atravesar los límites entre las variables a fin de transferir múltiples datos en una sola operación. Las variables no tienen que ser del mismo tipo al entrar y al salir (ver los ejemplos de códigos que se detallan a continuación). Tenga en cuenta que si se copia una matriz completa a la cola en una sola operación, el dato se transfiere empezando por el elemento más bajo.

Si la cola está llena, PutQueue suspenderá la tarea hasta que haya espacio suficiente para insertar los datos.

Las colas son un método muy eficaz para pasar datos de tarea a tarea o para almacenar los datos para un futuro procesamiento.

## Advertencia

Si no hay espacio suficiente en la cola, y ninguna de las tareas elimina nada de la cola, el procedimiento no devolverá ningún valor y la tarea se detendrá de manera indefinida.

## Ejemplo

```
Dim Oven(1 To 50) As Byte
Dim Pi As Single
Dim Fridge(1 To 4) As Byte

Sub Main()
    Call OpenQueue(Oven, 50)
    Pi = 3.14159

    ' Poner algunas Pi en el horno.
    Call PutQueue(Oven, Pi, 4)

    ' Poner cuatro piezas de Pi de tamaño byte en la nevera.
    Call GetQueue(Oven, Fridge, 4)

End Sub
```

# Procedimiento **PutQueueStr**

## Sintaxis

Call PutQueueStr(*Queue, StringSource*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Queue* | Matriz de bytes | Entrada/Salida | Cola en la que se copia la cadena. |
| *StringSource* | String | Entrada | La fuente de la cadena a copiar en la cola. |

## Descripción

PutQueueStr coloca una cadena en una cola.

Una aplicación típica para esta función es construir el equivalente de un comando "Print" (Imprimir), colocando el texto en una cola de un puerto serie para su transmisión a otro dispositivo.

## Advertencia

Si no hay espacio suficiente en la cola, y ninguna de las tareas elimina nada de la cola. El procedimiento PutQueueStr no devolverá ningún valor y la tarea se detendrá de manera indefinida. Una forma de evitar este problema es romper una cadena en partes más pequeñas y alimentar la cola de manera gradual.

## Ejemplo

```
Dim OCom(1 To 30) As Byte
Dim ICom(1 To 30) As Byte

Sub Main()

    Dim Howdy As String * 20

    Call OpenQueue(OCom, 30)
    Call OpenQueue(ICom, 30)
    Call OpenCom(2, 19200, ICom, OCom)

    Howdy = "Hello World!"
    Call PutQueueStr(OCom, Howdy)

    ' Adjuntar un retorno de carro y salto de línea
    Call PutQueueStr(OCom, Chr(13) & Chr(10))

End Sub
```

# Procedimiento **PutTime**

## Sintaxis

Call PutTime(*Hour, Minute, Second*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Hour* | Byte | Entrada | Horas. El rango es de 0 a 23. |
| *Minute* | Byte | Entrada | Minutos después de la hora. |
| *Second* | Single | Entrada | Segundos. La resolución es aprox. 1.95 ms. |

## Descripción

Fija la hora del día en un formato de 24 horas.

# Procedimiento **PutTimestamp**

## Sintaxis

Call PutTimestamp(*Year, Month, Day, Hour, Minute, Second*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Year* | Integer | Entrada | Año. El rango es de 1999 a 2177. |
| *Month* | Byte | Entrada | Mes. |
| *Day* | Byte | Entrada | Día. |
| *Hour* | Byte | Entrada | Horas. El rango es de 0 a 23. |
| *Minute* | Byte | Entrada | Minutos. |
| *Second* | Single | Entrada | Segundos. La resolución es aprox. 1.95 ms. |

## Descripción

Fija la fecha y la hora del día. La hora está en un formato de 24 horas. PutTimestamp define automáticamente el día de la semana (ver la función GetDayOfWeek).

# Procedimiento **PutXIO**

## Sintaxis (Sólo BX-01)

Call PutXIO(*Address, Value*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Address* | UnsignedInteger | Entrada | Dirección I/O, el rango es de 607 a 65 535. |
| *Value* | Byte | Entrada | Valor a enviar al puerto. |

## Descripción

El procedimiento PutXIO envía los datos a un puerto adicional de entrada/salida (I/O). BasicX soporta hasta 65.536 de estos puertos I/O, produciendo un total de 512 Kbits de entradas/salida (I/O).

Utilizar los mismos pines que la memoria RAM para la transmisión; la línea RD, WR y la línea de solicitud de entrada/salida (IO Request line), el sistema BasicX se dirige a los 65.536 puertos.

## Advertencia

Para el argumento *Address* no utilice valores inferiores a 607 (&H25F).

Este comando habilita los pines de RAM/XIO. Si tiene otras funciones o datos en estos pines, estos se sobrescribirán.

## Ejemplo

```
Dim Address As New UnsignedInteger
Dim Value As Byte

Address = &H3213
Value = &H47

' Transmitir los datos.
Call PutXIO(Address, Value)
```

# Procedimiento **PutXRAM**

## Sintaxis (Sólo BX-01)

Call PutXRAM(*Address, Buffer, Count*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Address* | UnsignedInteger | Entrada | Dirección de inicio en la memoria RAM adicional. El rango es de 608 a 65 535. |
| *Buffer* | Cualquier tipo | Entrada | Variable o matriz de la memoria RAM desde la que se envían los datos. |
| *Count* | UnsignedInteger | Entrada | Número de bytes a transferir. El rango es de 1 a 64 928. |

## Descripción

PutXRAM copia los datos de las variables locales de la memoria RAM ampliada. Las longitudes de la memoria RAM o local son de 64 KB.

PutXRAM puede transferir puede copiar un bloque grande de memoria de forma arbitraria en una sola operación; además el bloque puede aplicarse a múltiples variables en la memoria RAM.

## Ejemplo

```
Sub Main()

    Dim LocalData(1 To 20) As Single

    ' Escribir una matriz en XRAM, empezando en la ubicación
    ' 4096 (&H1000). Usar cuatro bytes por elemento
    ' para el tipo de punto flotante.
    Call PutXRAM( &H1000, LocalData, 20*4 )

    ' Recuperar la matriz de XRAM. La sintaxis es similar.
    Call GetXRAM( &H1000, LocalData, 20*4 )

End Sub
```

# Procedimiento **Randomize**

## Sintaxis

Call Randomize

## Argumentos

Ninguno

## Descripción

Randomize utiliza el reloj del sistema para fijar el valor del punto inicial (*seed)* para el generador de números aleatorios. Ver la función Rnd para obtener más detalles.

# Función RAMPeek

## Sintaxis

*F* = RAMPeek(*Address*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Address* | UnsignedInteger | Entrada | Dirección de memoria RAM |
| *F* | Byte | Salida | Valor del byte en la dirección indicada arriba. |

## Descripción

RAMPeek le permite leer cualquier de la memoria RAM, ignorando las reglas asociadas normalmente a los tipos de variables. Por ejemplo, puede consultar el tercer byte de una variable de 4 bytes de punto flotante, o mirar directamente los bytes de una cadena.

## Ejemplo

```
Dim Gbyte As Byte
Dim TestString As String * 32

Sub Main()

    ' Leer el byte en la ubicación 8756 (&h2234) de la memoria.
    Gbyte = RAMPeek(&h2234)

    TestString = "Hello World!"

    ' Leer el carácter 7 del resto de la cadena, que
    ' en realidad se emite 8 bytes después del principio
    ' de la cadena en la memoria.
    Gbyte = RAMPeek( MemAddress(TestString)+ 8 )

    ' En este punto, Gbyte es 87 (ASCII "W").

End Sub
```

# Procedimiento **RAMPoke**

## Sintaxis

Call RAMPoke(*Value, Address*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Value* | Byte | Entrada | Valor del byte a copiar en la memoria RAM |
| *Address* | UnsignedInteger | Entrada | Dirección de destino |

## Descripción

RAMPoke le permite escribir un byte en cualquier parte de la memoria RAM, ignorando las reglas asociadas normalmente a los tipos de variables. Por ejemplo puede modificar el byte superior de un entero, o bien modificar directamente los bytes de una cadena.

## Advertencia

La memoria RAM interna del chip BasicX ocupa las direcciones dentro del rango 0 a 607 (&H25F). **Cualquier transferencia a la RAM utilizada por el sistema operativo BasicX podría ocasionar un fallo del sistema.** Consulte el apartado acerca de la RAM de BasicX si desea obtener más información sobre este tema.

## Ejemplo

```
Sub Main()

    Dim TestString As String * 32
    Dim Gbyte As Byte

    TestString = "Hel o World!"

    ' Leer el carácter 3 de la cadena de prueba, que
    ' en realidad se emiten 4 bytes después del principio
    ' de la cadena de la memoria.
    Gbyte = RAMPeek( MemAddress(TestString) + 4 )

    ' En este punto, Gbyte es 108 (ASCII "l"). Copiar
    ' el byte en el siguiente carácter.
    Call RAMPoke( Gbyte, MemAddress(TestString) + 5 )

    ' La cadena lee ahora "Hello, World!"

End Sub
```

# Procedimiento **RCTime** (versión flotante)

## Sintaxis

Call RCTime(*Pin, State, Interval*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Pin* | Byte | Entrada | Número de pin |
| *State* | Byte | Entrada | Estado del pin – 0 (nivel lógico bajo) o 1 (nivel lógico alto) |
| *Interval* | Single | Salida | Intervalo de tiempo, en unidades de segundos. El rango válido es aproximadamente de 1.085 µs a 71.1 ms. El intervalo devuelve un valor de 0.0. |

## Descripción

RCTime mide el tiempo que un pin I/O permanece en un determinada estado. El pin se configura con el modo Input-tristate (alta impedancia) para la medición. EL intervalo devuelve un valor de 0.0. La resolución es aproximadamente 1.085 µs.

## Advertencia

RCTime dedica el procesador para buscar una transición. El reloj de tiempo real (RTC), la conmutación de las tareas y el tráfico de la red se suspenden durante este tiempo.

El procedimiento sobrescribe cualquier configuración anterior del pin dejando el pin en estado Input-tristate.

Si el pin no está en el estado especificado al llamar a RCTime, el procedimiento devuelve inmediatamente el valor más bajo (diferente de cero) permitido como Intervalo (*Interval),* aproximadamente 1.085E-6.

## Ejemplo

En este ejemplo se ilustra el uso del procedimiento f RCTime para medir el tiempo que tarda una resistencia en descargarse.

```
Dim TimeDelay As Single
Call PutPin(17, bxInputLow) ' Pull I/O pin 17 low.
' Esperar aprox. 8,7 microsegundos hasta que se descargue la
resistencia.
Call Delay(8.7E-6)
' Medir el tiempo que tarda la resistencia en cargar
' un punto inicial (seed). Fijar el pin 17 en modo Input-tristate y
' después medir el tiempo que permanece el pin en el nivel lógico bajo

Call RCTime(17, 0, TimeDelay)
```

# Función RCTime (Versión de entero)

## Sintaxis

*Interval* = RCTime(*Pin, State*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Pin* | Byte | Entrada | Número de pin |
| *State* | Byte | Entrada | Estado de pin – Estado del pin – 0 (nivel lógico bajo) o 1 (nivel lógico alto) |
| *Interval* | Integer | Salida | Intervalo de tiempo, en unidades de 8 / 7 372 segundos (aprox. 1.085 µs). El rango válido es aproximadamente de 1 a 32 767 unidades. El intervalo devuelve un valor de 0 o un valor negativo. |

## Descripción

RCTime mide el tiempo que un pin I/O permanece en un estado determinado. El pin se configura con el estado Input-tristate (alta impedancia) para la medición. El intervalo devuelve un valor 0 o un valor negativo.

## Advertencia

RCTime dedica el procesador para buscar una transición. El reloj de tiempo real (RTC), la conmutación de las tareas y el tráfico de la red se suspenden durante este tiempo.

El procedimiento sobrescribe cualquier configuración anterior del pin dejando el pin en estado Input-tristate.

Si el pin no está en el estado especificado al llamar a RCTime, el procedimiento devuelve inmediatamente un valor 1 como Intervalo (*Interval)*

## Ejemplo

Este ejemplo se ilustra el uso del procedimiento RCTime para medir el tiempo que tarda la resistencia en descargarse.

```
Dim TimeDelay As New UnsignedInteger

Call PutPin(17, bxInputLow) ' Pull I/O pin 3 low.

' Esperar aprox. 8,7 microsegundos para que la resistencia se descargue.
Call Sleep(8)

' Medir el tiempo que tarda la resistencia en cargar hasta un punto
' establecido. Configurar el pin 3 en modo Input-tristate y después
' medir el tiempo que permanece el pin en un nivel lógico bajo.
TimeDelay = RCTime(17, 0)
```

# Procedimiento **ResetProcessor**

## Sintaxis

Call ResetProcessor( )

## Argumentos

Ninguna.

## Descripción

ResetProcessor hace que el procesador BasicX se resetee y se reinicie en un intervalo de 17 milisegundos. Internamente, este procedimiento utiliza en realidad el temporizador watchdog para resetear el procesador.

# Función Rnd

## Sintaxis

*F* = Rnd

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Rnd devuelve un número aleatorio superior o igual a 0.0 e inferior a 1.0.

El procedimiento Rnd es un generador de números aleatorios congruenciales y multiplicativos que utiliza un número entero del punto inicial (*seed*) de 32 bits en la memoria estática. El procedimiento Randomize puede utilizarse para fijar este punto inicial (*seed*) basado en el valor del reloj del sistema.

De otro modo, también se tiene acceso directo al valor inicial (*seed*), que es una variable global proporcionada por el sistema denominada SeedPRNG. El valor inicial es un tipo Long de 32 bits.

# Función Semaphore

## Sintaxis

*F* = Semaphore(*Variable*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Variable* | Boolean | Entrada | Variable booleana en uso como semáforo. |
| *F* | Boolean | Salida | La función devuelve el valor *true* si el semáforo es propiedad de esta tarea; *false* si el semáforo ya lo está utilizando otra tarea. |

## Descripción

Semaphore es una función que permite a las tareas compartir variables y cooperar.

Los semáforos protegen los datos compartidos. Un semáforo es un mecanismo de señalización que permite a una tarea indicar a otras tareas la propiedad de un bloque determinado de datos. Cuando una tarea propietaria ha acabado de utilizar los datos, la tarea cambia el estado del semáforo, traspasando la propiedad del semáforo a otras tareas y permitiéndoles que utilicen los datos.

Debido a la naturaleza compleja de esta función. Le recomendamos que consulte toda la sección relacionada con el uso de semáforos.

## Advertencia

Si una tarea no puede definir el semáforo como *false* al acabar de utilizar los datos compartidos, otras tareas no podrán jamás utilizar los datos, y su sistema podría detenerse inmediatamente.

## Ejemplo

Consultar el fichero de ejemplo SemaphoreEx.bas.

# Procedimiento **SerialNumber**

## Sintaxis

Call SerialNumber(*Value*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Value* | Matriz de Byte (1 a 6) | Salida | Matriz que contiene la versión y los números de serie. Formato interno:<br><br>Byte 1 – Número de versión principal<br>Byte 2 -- Número de versión secundario<br><br>Bytes 3 - 6 – Número de serie de cuatro bytes (Sólo BX-01) |

## Descripción

Este procedimiento devuelve los números de versiones principal y secundario del chip BasicX. En los sistemas BX-01, el procedimiento también devuelve los datos únicos de números de serie en los bytes 3 - 6. En los sistemas BX-24, los bytes 3 - 6 están sin definir.

## Ejemplo

```
Dim SNC(1 to 6) As Byte
Dim MajorVersion As Byte
Dim MinorVersion As Byte
Dim SNumber(1 to 4) As Byte

' Leer los datos compuestos.
Call SerialNumber(SNC)

' Extraer los números de versión.
MajorVersion = SNC(1)
MinorVersion = SNC(2)

' Extraer el número de serie de 4 bytes (Sólo BX-01).
SNumber(1) = SNC(3)
SNumber(2) = SNC(4)
SNumber(3) = SNC(5)
SNumber(4) = SNC(6)
```

# Función Sin

## Sintaxis

*F* = Sin(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Función de seno. El operando está en unidades de radianes.

## Ejemplo

```
Dim F As Single
Const Pi As Single = 3.14159265

' 30 grados, convertidos a radianes.
F = Sin(Pi/6.0) ' Aquí F es 0.5
```

# Procedimiento **Sleep** (versión flotante)

## Sintaxis

Call Sleep(*SleepInterval*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *SleepInterval* | Single | Entrada | El intervalo del procedimiento de pausa (*sleep*) tiene un rango de aprox. 0.0 a 128.0 segundos. La resolución es aprox. 1.95 ms. |

## Descripción

Este procedimiento suspende la tarea actual aproximadamente durante el intervalo de tiempo especificado. Al final de *SleepInterval*, la tarea estará lista de nuevo. El tiempo real que tarde la tarea en volver a ejecutarse depende de lo ocupado que esté el sistema con otras tareas.

Una pausa de 0.0 es un método útil para permitir que otras tareas puedan ejecutarse, al tiempo que se reinicia inmediatamente si no hay otras tareas a la espera de ejecutarse.

Si la tarea está boqueada, el procedimiento Sleep desbloqueará la tarea (ver procedimiento LockTask).

## Advertencia

En realidad, el procedimiento Sleep espera un número determinado de marcaciones del reloj, lo que implica que *SleepInterval* no garantiza un retardo mínimo o máximo, dado que la duración exacta depende de la sensibilidad del programa ante un ciclo de marcaciones (si desea fijar un retardo mínimo garantizado, consulte el procedimiento Delay).

## Ejemplo

```
'Fijar el pin 1 con un nivel lógico alto (high)
Call PutPin(1, 1)

'Detiene momentáneamente esta tarea durante aprox. 1/2 segundo, y
después se reinicia
Call Sleep(0.5)

'Fijar el pin 1 con un nivel lógico bajo(low)
Call PutPin(1, 0)
```

# Procedimiento **Sleep** (Versión de entero)

## Sintaxis

Call Sleep(*SleepInterval*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *SleepInterval* | UnsignedInteger | Entrada | El intervalo de pausa (*sleep)* tiene un rango de 0 a 65 535. Las unidades son 1/512 segundos (aprox. 1.95 ms). |

## Descripción

Este procedimiento suspende la tarea actual aproximadamente durante el intervalo de tiempo especificado. Al final de *SleepInterval*, la tarea estará lista de nuevo. El tiempo real que tarde la tarea en volver a ejecutarse depende de lo ocupado que esté el sistema con otras tareas.

Una pausa de 0 es un método útil para permitir que otras tareas puedan ejecutarse, al tiempo que se reinicia inmediatamente si no hay otras tareas a la espera de ejecutarse.

Si la tarea está boqueada, el procedimiento Sleep desbloqueará la tarea (ver procedimiento LockTask).

## Advertencia

En realidad, el procedimiento Sleep espera un número determinado de marcaciones del reloj, lo que implica que *SleepInterval* no garantiza un retardo mínimo o máximo, dado que la duración exacta depende de la sensibilidad del programa a un ciclo de marcaciones (si desea fijar un retardo mínimo garantizado, consulte el procedimiento Delay).

## Ejemplo

```
' Fijar el pin 1 con un nivel lógico alto (high)
Call PutPin(1, 1)

' 'Detiene momentáneamente esta tarea durante aprox. 1/2 segundo, y
después se reinicia

Call Sleep(256)

' Fijar el pin 1 con un nivel lógico bajo (low)
Call PutPin(1, 0)
```

# Función ShiftIn

## Sintaxis

**Sólo BX-24, BX-35**

*F* = ShiftIn(*DataPin, ClockPin, NumberOfBits*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *DataPin* | Byte | Entrada | Número de pin de datos fuente. |
| *ClockPin* | Byte | Entrada | Número de pin de reloj. |
| *NumberOfBits* | Byte | Entrada | Número de bits. El rango es de 1 a 8. |
| *F* | Byte | Salida | Valor devuelto por función. |

## Descripción

Esta función transfiere hasta 8 bits de datos a través de la entrada *DataPin*. El sistema operativo lo registra automáticamente en cada bit al utilizar el *ClockPin* especificado. Para ser compatible con los dispositivos I2C, la tasa de bit es inferior a 400 kHz.

El orden de los bits es primero el bit MS, y el bit LS el último.

Antes de llamar a ShiftIn, el pin del reloj debe estar antes configurado en el nivel lógico correcto (alto o bajo).

## Ejemplo

```
Dim A As Byte

' Fijar el pin 1 con un nivel lógico bajo (low).
Call PutPin(17, bxInputLow)

' Transmitir 4 bits en A. El pin 16 se usa para la entrada de datos.
A = ShiftIn(16, 17, 4)
```

# Procedimiento **ShiftOut**

### Sintaxis

**Sólo BX-24, BX-35**

Call ShiftOut(*DataPin, ClockPin, NumberOfBits, Operand*)

### Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *DataPin* | Byte | Entrada | Número de pin de datos fuente |
| *ClockPin* | Byte | Entrada | Número de pin de reloj |
| *NumberOfBits* | Byte | Entrada | Número de bits. El rango es de 1 a 8. |
| *Operand* | Byte | Entrada | Fuente de los datos. |

### Descripción

Esta función transfiere hasta 8 bits de datos desde el *Operando* a través de la salida de *DataPin*. El sistema operativo lo registra automáticamente en cada bit al utilizar el *ClockPin* especificado. Para ser compatible con los dispositivos I2C, la tasa de bit es inferior a 400 kHz.

El orden de los bits es el bit MS primero y el bit LS en último lugar.

Antes de llamar al procedimiento ShiftIn, el pin del reloj debe fijarse antes en el nivel correcto (alto o bajo).

### Ejemplo

```
Dim A As Byte

' Fijar el pin del reloj con un nivel lógico bajo (low).
Call PutPin(17, bxOutputHigh)

' Transferir 4 bits desde A. El pin 16 se usa para la salida de datos.
Call ShiftOut(16, 17, 4, A)
```

# Procedimiento **SPICmd**

## Sintaxis

Call SPICmd(*Channel, PutCount, PutData, GetCount, GetData*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Channel* | Byte | Entrada | Número de canal de SPI. El rango es de 1 a 4. |
| *PutCount* | Byte | Entrada | Número de bytes a enviar al dispositivo. Cero indica que no hay datos. |
| *PutData* | Cualquier tipo | Entrada | Datos a enviar (si PutCount = 0, seguirá necesitando un argumento ficticio aquí). |
| *GetCount* | Byte | Entrada | Número de bytes a recibir desde el dispositivo. |
| *GetData* | Cualquier tipo | Entrada/Salida | Datos a recibir. |

## Descripción

BasicX tiene un bus de interfaz periférica serie (Serial Peripheral Interface - SPI) integrado en el hardware del chip. Utilizando este bus, los periféricos de otros fabricantes como Motorola y National Semiconductor pueden ser utilizados para funciones especiales imposibles de realizarse directamente a través del chip BasicX.

El bus SPI es un bus útil a través del cual el emisor y receptor pueden intercambiar datos de manera simultánea. Es decir, los datos pueden ir en ambas direcciones simultáneamente. El flujo de datos puede también unidireccional si se desea -- SPICmd le permite ambos casos.

Antes de llamar a SPICmd, deberá llamar a OpenSPI para inicializar el canal de SPI.

## Advertencia

Este comando está recomendado para usuarios que conozcan el bus SPI. El código BasicX normalmente lo recibe la memoria EEPROM de SPI, lo que implica que si el bus SPI no se ha manejado correctamente, las instrucciones no se recibirán correctamente.

# Función **Sqr**

## Sintaxis

*F* = Sqr(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Función de raíz cuadrada.

## jemplo

```
Dim F As Single

F = Sqr(9.0) ' F es 3.0
```

# Función StatusQueue

## Sintaxis

*F* = StatusQueue(*Queue*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Queue* | Matriz de bytes | Entrada | Nombre de la cola a comprobar. |
| *F* | Boolean | Salida | Devuelve un valor *true* si hay datos en la cola (*Queue)*. De lo contrario, devolverá un valor *false*. |

## Descripción

StatusQueue permite al programador comprobar si hay algún dato en una cola antes de que la tarea obtenga los datos. Si la tarea no comprueba la cola utilizando la función StatusQueue ay después intenta leer los datos de una cola vacía, la tarea se bloqueará hasta que los datos estén disponibles.

## Advertencia

Cuando se está utilizando una cola como un buffer de salida del puerto serie, StatusQueue no es la función adecuada para determinar si el buffer ha sido vaciado. En particular, la función StatusQueue puede indicar si la cola está vacía antes de finalizar una transmisión.

## Ejemplo

```
Dim Queue(1 To 30) As Byte

Sub Main()

    Dim Data As Byte

    Call OpenQueue(Queue, 30)
    Do
        ' Si el dato está en la cola, extraer un byte.
        If StatusQueue(Queue) Then
            Call GetQueue(Queue, Data, 1)
        End If
    Loop
End Sub
```

# Función Tan

## Sintaxis

*F* = Tan(*Operand)*

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *Operand* | Single | Entrada | Operando |
| *F* | Single | Salida | Valor devuelto por función |

## Descripción

Función de tangente. El operando está en unidades de radianes.

## Ejemplo

```
Dim F As Single

' Tan(Pi/4)
F = Tan(0.785398) ' F es 1.0.
```

# Función TaskIsLocked

## Sintaxis

*F* = TaskIsLocked( )

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *F* | Boolean | Salida | Si la tarea está bloqueada. |

## Descripción

TaskIsLocked le permite comprobar si la tarea actual está bloqueada. La función es útil si tiene un subprograma que necesita bloquear la tarea, y después restaurar es estado de bloqueo después de la devolución.

# Función Timer

## Sintaxis

*F* = Timer( )

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *F* | Single | Salida | Segundos en formato de punto flotante desde la medianoche. El rango es de 0.0 a 86 400.0 segundos. |

## Descripción

Devuelve el valor del tiempo transcurrido desde la medianoche. La resolución depende de la hora del día – el mejor caso es aprox. 1.95 ms (1/512 segundos) para los valores de tiempo pequeños.

## Ejemplo

```
Dim T1 As Single, T2 As Single, DT As Single

' Encontrar la hora de inicio.
T1 = Timer

Call TimedProcedure

' Encontrar la hora de finalización.
T2 = Timer

' Calcular el tiempo transcurrido.
DT = T2 - T1
```

# Función Trim

## Sintaxis

*F* = Trim(*StringVar*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *StringVar* | String | Entrada | Cadena de entrada |
| *F* | String | Salida | Cadena de salida |

## Descripción

Elimina los espacios en blanco de cabeza y cola de una cadena.

## Ejemplo

```
Dim Tx1 As String
Dim Tx2 As String

Tx1 = "   Hello, world   "

Tx2 = Trim(Tx1) ' Tx2 es "Hello, world"
```

# Función UCase

## Sintaxis

*F* = UCase(*StringVar*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *StringVar* | String | Entrada | Cadena de entrada |
| *F* | String | Salida | Cadena de salida |

## Descripción

Convierte una cadena a mayúsculas.

## Ejemplo

```
Dim Tx1 As String
Dim Tx2 As String

Tx1 = "abc"
Tx2 = UCase(Tx1)  ' Tx2 es "ABC"
```

# Procedimiento **UnlockTask**

## Sintaxis

Call UnlockTask( )

## Argumentos

Ninguno.

## Descripción

UnlockTask desbloquea una tarea. Este procedimiento invierte el efecto del procedimiento LockTask (bloquear una tarea impide al sistema operativo cambiar a otra tarea). Desbloquear la tarea hace que se vuelva activar la conmutación de las tareas.

Está permitido llamar al procedimiento UnlockTask si la tarea ya está desbloqueada – realizar múltiples llamadas al procedimiento UnlockTask tiene el mismo efecto que realizar sólo una si la tarea ya está desbloqueada. Por ejemplo, por norma general no se necesitan 2 llamadas a LockTask para desbloquear 2 llamadas a UnlockTask.

# Procedimiento **ValueS**

## Sintaxis

Call ValueS(*StringVar*, *Value*, *Success*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *StringVar* | String | Entrada | Cadena de entrada |
| *Value* | Single | Salida | Valor de devolución |
| *Success* | Boolean | Salida | Indicador de éxito (success) |

## Descripción

Convierte una cadena a un tipo Single. Si no se producen errores, el número se devolverá en Value y el indicador de éxito se fija como True. De lo contrario Value se fija a 0.0 y el indicador Success (Éxito) se fija a False.

El número de la cadena debe estar formado por dígitos numéricos, cuyos signos son opcionales para el número y exponente. Un punto decimal es también opcional. Se ignorarán los caracteres de control de cola y cabeza de la cadena (como los espacios o etiquetas).

## Ejemplo

```
Dim Tx As String
Dim Value As Single
Dim Success As Boolean

Tx = "  123 "
Call ValueS(Tx, Value, Success) ' Value es 123.0, Success es True

Tx = "-4.5E+03"
Call ValueS(Tx, Value, Success) ' Value es -4500.0, Success es True

‘Caracteres.
Tx = "&HFF"
Call ValueS(Tx, Value, Success) ' Valor es 0.0, Success es False
```

# Procedimiento **WaitForInterrupt**

## Sintaxis

Call WaitForInterrupt(*Interruptype*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *InterruptType* | Byte | Entrada | Tipo de interrupción. Consultar los valores permitidos en la tabla siguiente. |

Valores permitido para *InterruptType:*

| | | |
|---|---|---|
| bxComparatorToggle | = 0 | Comparador de estado de conmutación (Sólo BX-01) |
| bxComparatorFallingEdge | = 2 | Flanco descendiente del comparador (Sólo BX-01) |
| bxComparatorRisingEdge | = 3 | Flanco ascendente del comparador (Sólo BX-01) |
| bxPinLow | = 16 | Nivel lógico bajo en el pin de interrupción |
| bxPinFallingEdge | = 24 | Flanco descendiente en pin de interrupción |
| bxPinRisingEdge | = 28 | Flanco ascendente en pin de interrupción |

Número de pin de interrupción de BX-01: 13 (PDIP)
Número de pin de interrupción de BX-24: 11 (compartido con pin I/O)
Número de pin de interrupción de BX-35: 17 (PDIP)

## Descripción

WaitForInterrupt permite a una tarea responder de manera inmediata a un evento crítico del mundo exterior. Este procedimiento proporciona acceso a los interruptores de hardware integrados en el chip BasicX

WaitForInterrupt bloquea la llamada a la tarea hasta que se produzca el evento de activación. Cuando se produce el evento, se programará la ejecución inmediata de la tarea. El disparador tiene prioridad, incluso si otra tarea está ejecutándose y está bloqueada (consultar el procedimiento LockTask), en cuyo caso la otra tarea se desbloqueará temporalmente.

## Advertencia

Si no se genera ningún evento externo, la llamada de la tarea podría esperar de manera indefinida.

En el sistema BX-24, se comparte la línea de interrupción con el pin 11 de entrada/salida (I/O), lo que implica que el pin 11 debería estar configurado como Input-tristate o Input-pullup si quiere utilizar la línea de interrupción.

## Ejemplo

```
' Esperar un flanco ascendente en el comparador.
Call WaitForInterrupt( bxComparatorRisingEdge )
```

# Procedimiento **Watchdog**

## Sintaxis

Call Watchdog( )

## Argumentos

Ninguno.

## Descripción

Watchdog reinicia el temporizador watchdog antes de que se ponga a cero.

Antes de llamar al procedimiento Watchdog, es necesario llamar a OpenWatchdog para iniciar el temporizador watchdog. Consultar el apartado de OpenWatchdog si desea obtener más información.

# Procedimiento **X10Cmd**

## Sintaxis

**Sólo BX-24, BX-35**

Call X10Cmd(*PinOut, Pin60Hz, HouseCode, KeyCode, RepeatCycles*)

## Argumentos

| Elemento | Tipo | Dirección | Descripción |
|---|---|---|---|
| *PinOut* | Byte | Entrada | Pin de salida |
| *Pin60Hz* | Byte | Entrada | Pin de 60 Hz. |
| *HouseCode* | Byte | Entrada | Código House (Casa) |
| *KeyCode* | Byte | Entrada | Código Key (Clave) |
| *RepeatCycles* | Byte | Entrada | Número de ciclos de repetición. |

## Descripción

Transmite un comando X-10 command con la tasa de repetición especificado por *RepeatCycles*.

## Ejemplo

```
Const X10_P As Byte = &HC
Const X10_Dim As Byte = &H9
Const X10_Bright As Byte = &HB

Call X10Cmd(16, 17, X10_P, X10_Dim, 8)
Call Delay(1.0)
Call X10Cmd(16, 17, X10_P, X10_Bright, 8)
```