Machine Translated by Google

Gran vaca básica
CHIPINO



manual de programacion

Sample Do Not Copy

Machine Translated by Google

Publicado por Productos Electrónicos.

Esta es una versión en línea solo para uso personal.
Por favor, no copiar.
La editorial ofrece descuentos especiales en pedidos al por mayor de este libro.

Para información contactar:

Productos electrónicos
PO Box 251 Milford, MI
48381
www.elproducts.com
chuck@elproducts.com

El nombre y el logotipo de Microchip, MPLAB® y PIC® son marcas comerciales registradas de Microchip Technology Inc. en EE. UU. y otros países. chipKIT™ es una marca comercial de Microchip Technology Inc. en EE. UU. y otros países.

Todas las demás marcas comerciales mencionadas en este documento son propiedad de sus respectivas empresas.

Impreso en los Estados Unidos de América

Sample Version - Do Not Copy

Machine Translated by Google

Tabla de contenido

1.0 Introducción..... 7 1.1 Uso de Great Cow Basic ..... 9 1.2 Uso de Great Cow Basic ..... 13 1.3 Creando su primer proyecto ..... 15 1.4 Opciones de la línea de comandos..... 20 2.0 Conceptos básicos del compilador..... 21 2.1 Entradas/Salidas..... 21 2.3 Variables ..... 24 2.4 Fundición ..... 28 2.5 Definiciones/Constantes ..... 28 2.6 Constantes numéricas ..... 30 2.6.1 BCD a Decimal..... 30 2.6.2 De decimal a BCD ..... 30 2.7 Constantes de cadena ..... 31 2.8 Conversión de cadenas ..... 32 2.8.1 Cadena a hexadecimal ..... 32 2.8.2 Cadena a valor decimal..... 32 2.8. 3 Buscar cadena dentro de una cadena ..... 32 2.8.4 Convertir a minúsculas ..... 33 2.8.5 Convertir a mayúsculas ..... 33 2.8.6 Convertir número en cadena ..... 34 2.8.7 Recuperar caracteres de cadena (izquierda) ..... 34 2.8.8 Recuperar caracteres de cadena (medio) ..... 35 2.8.9 Recuperar caracteres de cadena (derecha)..... 35 2.9 Operaciones Matemáticas..... .. 35 2.10 Condiciones..... 37 2.11 Combinación de varias instrucciones ..... 38 2.12 Comentarios ..... 39 3.0 Directivas . ..... 41 3.1 #chip..... 41 3.2 #config ..... 41 3.3 #definir..... 41 3.4 #si ..... 42 3.5 #ifdef..... 43 3.6 #ifndef ..... 45 3.7 #incluir..... 45 3.8 #guión..... 46 3.9 #arranque..... 46 4.0 Funciones avanzadas ..... 47 4.1 Matrices..... 47 4.2 Funciones..... 48 4.3 Interrupciones .... 48 4.4 Tablas de consulta 49

Sample Version - Do Not Copy

Machine Translated by Google

4.5 Guiones 50 4.6
Subrutinas 51 4.7
Descripción general del teclado PS2 .....
55 4.8 Resumen aleatorio 56 4.9
Descripción general de I2C .....
57 4.10 Descripción general del temporizador .....
58 4.11 Resumen de siete segmentos. 58
4.12 Descripción general de PWM 60
4.13 Software son RS232 Descripción general 62
4.14 Descripción general del hardware RS232 63
4.15 Resumen del teclado 64 4.16
Vista general de la pantalla LCD .....
66 4.17 Resumen de tonos 70
4.18 Descripción general de SPI .....
71 4.19 Descripción general de GLCD ..
72 5.0 Comandos 75
Caja..... 75 Borrar
Temporizador 75
CLS 76
AtenuDirección ..... 77. ...
79 DisplayChar .....
80
Valor de visualización.....
81 Hacer..... 83
Fin 85
EPRead 85
Escribir EP..... 87 Salir
de Sub..... 89
CajaRellenada. 90
Para..... 90
Obtener..... 92
GLCDCLS 92
GLCDImprimir..... 93
GLCDDrawChar 93
GLCDWriteByte..... 94
GLCDReadByte. 94
Gosub ..... 94 Ir
a..... 96
HPWM 97
HSerImpresión..... 99
HSerReceive 104
HSerSend ..... 105 Recibir
I2C..... 107
I2CSend..... 109 Inicio
I2C 111
I2CSparar 112

Sample Version – Do Not Copy

Machine Translated by Google

Si entonces más 112
InitGLCD 114
InitSer..... 116
TemporizadorInicial0 117
TemporizadorInicial1 120
LLAVE 122
Interrupciones..... 125
IntOff..... 126
Inton ..... 127 Datos del
teclado ..... 127 Teclado en
bruto..... 130
LCDCreateChar 132
LCDHex..... 134
LCDWriteChar 135
Línea 137
Localizar..... 137 En
Interrupción 138
Mirar.. 143
Empuje 144
Bote 145
Imprimir..... 147 Borrado
de programa ..... 148 Lectura de
programa..... 149 Escritura de
programa 150
PS2ReadByte 150
PS2SetKBLeds 151
PS2WriteByte 152
Pset ..... 153 Pulso de
salida 153
Poner ..... 154 PWM
desactivado ..... 156 PWM
On .. .. 157 Salida
PWM..... 158
Aleatorio..... 159
Aleatorizar ..... 160 Leer
AD..... 161 Tabla de
lectura 162
Repetir 164
Rotar 165
Seleccionar..... 167
SerPrint 170
SerReceive 175
SerSend 176
Establecer ..... 177 Tono
corto..... 179 Modo
SPI 180

Sample Version - Do Not Copy

Machine Translated by Google

Transferencia SPI 181
Temporizador de inicio 184
Temporizador de parada 185
Tono 185
Espera 186 6.0
Ejemplos de proyectos 190
Proyecto 1 - Cruce de trenes 190
Proyecto 2 - Detección de un interruptor 194
Proyecto 3 – Creación de sonido ..... 197 Proyecto
4 - Detección de la luz ..... 200 Apéndice
A: características de la interfaz gráfica de usuario de PICkit 2 ..... 204 Herramienta UART 204
Conexión de la herramienta UART 205
Ventana de herramientas UART 206
Configuración de la velocidad en baudios y conexión 206
Modo ASCII ..... 207 Modo
hexadecimal 208
Ajustar texto 208
Registrar en archivo 208
Borrar pantalla ..... 209 Salir
de la herramienta UART 209
Herramienta lógica ..... 209 Modo de E/S lógica 210
Configuración de la E/S lógica de la herramienta lógica .....
210 Configuración de la dirección del pasador... ..... 211 213
Modo de analizador lógico .....
Conexión del analizador ..... 213 La
ventana del analizador lógico ..... 214 Pantalla
del analizador ..... 215 Cursores de la
pantalla del analizador ..... 216 Disparador
del analizador ..... 217 Adquisición
del analizador ..... 218 Ejecución del
analizador 220

Sample Version – Do Not Copy

Machine Translated by Google

1.0 Introducción

Great Cow BASIC (GCB) es un compilador BASIC para microcontroladores PIC. Te permite programar en BASIC en lugar de tener que aprender lenguaje ensamblador o C. ¡También es completamente de código abierto!

CHIPINO es un módulo de desarrollo de código abierto que facilita mucho la programación con lenguaje Great Cow Basic, Assembly o C. Tiene el mismo tamaño que los módulos Arduino populares, por lo que todos los escudos enchufables que funcionan con Arduino pueden funcionar con CHIPINO.

CHIPINO utiliza un cable de programación basado en el programador PICkit 2 de código abierto de Microchip. Esto permite que el CHIPINO se use con cualquier microcontrolador PIC de 28 pines en blanco. Esto también permite al usuario desenchufar el chip y convertirlo en un diseño permanente cuando el software esté completo.

Esto hace que la combinación de Great Cow Basic y CHIPINO sea una plataforma de desarrollo perfecta para crear productos electrónicos.

Great Cow BASIC ha sido escrito con tres objetivos principales: eliminar la necesidad de comandos de ensamblaje repetitivos, producir un código eficiente y facilitar el tomar el código escrito para un chip y ejecutarlo en otro. Oculta muchas de las partes más confusas de la programación de microcontroladores, lo que lo hace adecuado para principiantes y para aquellos a quienes no les gusta ensamblar o C. También es excelente para que los niños aprendan programación electrónica. ¿Y no somos todos niños por dentro?

La sintaxis de Great Cow BASIC se basa en la de QBASIC/FreeBASIC, pero con algunas modificaciones para adaptarse al sistema muy diferente que constituye un microcontrolador. Great Cow BASIC le permitirá programar la mayoría de los microcontroladores PIC de 8 bits (chips PIC10F, 12F, 16F y 18F).

Características:

- Declaraciones de control de flujo BASIC estándar: si, seleccionar mayúsculas y minúsculas, hacer, para, ir a • Compatibilidad con multiplicar, dividir, sumar, restar, operaciones booleanas y comparaciones. • Tipos de datos Bit, Byte, Word y String, además de matrices de bytes. • Subrutinas y Funciones

- Montaje en línea, en la mayoría de los casos sin directivas especiales • Tablas de datos

Código portátil y reutilizable:

- Admite la mayoría de los microcontroladores PIC de 8 bits • Escriba código para un PIC10F, 12F , Microcontrolador PIC de 8 bits 16F o 18F

Sample Version - Do Not Copy

Machine Translated by Google

• Recalcula automáticamente todos los comandos de retardo según la velocidad del reloj del chip

Capacidades de E/S:

• Rutinas LCD 2x16 estándar • Rutinas para A/D en chip, PWM, SPI, USART, EEPROM y temporizadores • Comunicaciones RS232: velocidades entre 300 y 19200 bps con el usuario paridad configurable, bits de inicio y parada. • Lectura de teclado PS/2 • Teclado 4x4

Otras características:

• Amplio archivo de ayuda y foro
• Varias traducciones
• Genera código ensamblador compatible con MPASM estándar para PIC. • ¡Gratis, tanto en la forma de cerveza gratis como en la forma de expresión libre! • Fuente abierta.

Great Cow Basic es creado por un grupo de colaboradores clave que incluyen:

Colaboradores del programa:
Hugh Considine - Desarrollador principal de Great Cow Basic
Stefano Bonomi - Subrutinas LCD de dos hilos
Geordie Millar - Subrutinas Swap y Swap4
Finn Stokes: rutina de multiplicación de 8 bits, código de acceso a la memoria del programa

Colaboradores de traducción (Solo admitimos inglés en este manual):
Stefano Delfiore - Italiano
Pablo Curvelo - Español
Murat Inceer - Turco

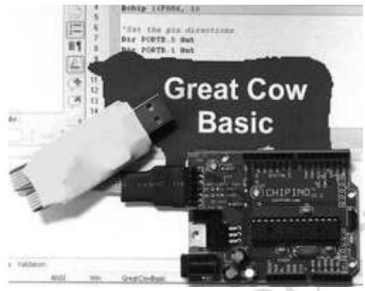
Otros colaboradores:
Russ Hensel - Great Cow Notas BÁSICAS
Chuck Hellebuyck - Este manual.
Frank Steinberg - GCB@SYN IDE para Great Cow Basic Alexy T. - SynWrite IDE utilizado para GCB IDE

Sample Version - Do Not Copy

Machine Translated by Google

1.1 Uso de Great Cow Basic

La mejor manera de comenzar a usar Great Cow Basic es con un kit de inicio completo. Puede obtener el módulo CHIPINO, el software Great Cow Basic para Windows, el cable del programador USB, el módulo CHIPINO, proyectos de software de muestra y este manual de libro electrónico en formato pdf en un paquete de inicio completo.



Con esta configuración, todo lo que necesita es una PC con Windows, XP, Vista o Win7 o una Mac con Parallels o VMWare y Windows.

El CHIPINO hace que sea fácil para un principiante usar Great Cow Basic debido a todos los escudos enchufables preconstruidos que se apilan en la parte superior. Hay numerosos ejemplos de código para este kit de inicio en www.greatcowbasic.com.

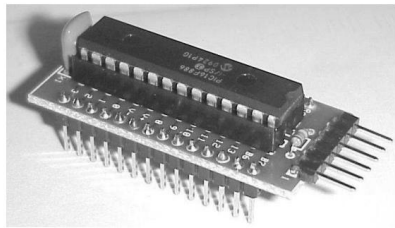


Pantalla LCD y pantalla de demostración

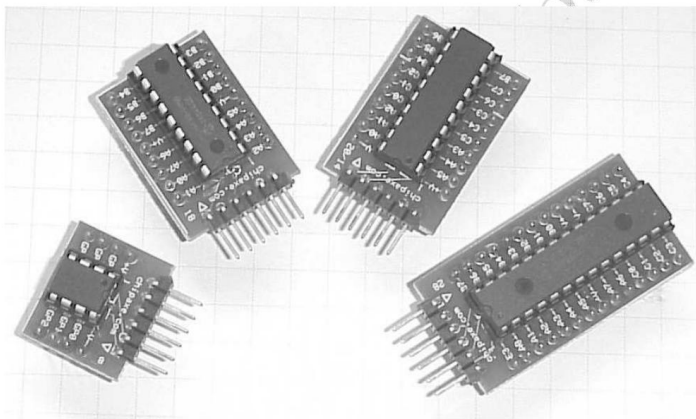
Sample Version - Do Not Copy

Machine Translated by Google

También puede usarlo con un módulo de placa de prueba CHIPINO.



También puede usar Great Cow Basic con los módulos CHIPAXE de Howtronics.com.
Estos módulos facilitan el uso de GCB con PIC Microchip de 8 pines, 14 pines, 18 pines, 20 pines y 28 pines.



También puede usar Great Cow Basic para programar un microcontrolador PIC de 8 pines y usarlo con Snap Circuits®. También hay adaptadores para conectar un módulo CHIPAXE a un módulo Snap Circuits®.

Sample Not Copy

Machine Translated by Google





Mínimo requerido Todo
lo que realmente necesita para usar Great Cow Basic es un microcontrolador PIC de Microchip y un programador. Funciona muy bien con un Microchip PICkit2 o uno de los muchos clones de PICkit 2 ya que el diseño de PICKit 2 también es de código abierto.



Kit PIC original 2



Clon PK2



PICkit 2 Clon

Machine Translated by Google

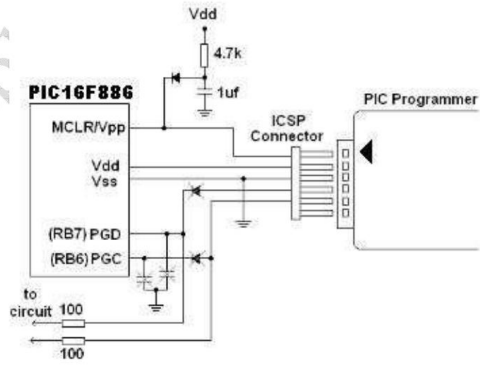
El compilador básico Great Cow es compatible con todos los PIC de Microchip de 8 bits, incluidas todas las partes que comienzan con PIC18F, PIC16F, PIC12F y PIC10F. Una parte muy común para empezar es el PIC16F886. Es una parte de 28 pines que tiene la mayoría de las características que tienen todos los demás PIC, que incluyen:

- Oscilador interno •
- Opción de resistencia MCLR interna •
- Convertidor analógico a digital (ADC) de 10 bits • 22 E/S • EEPROM • UART • Hardware SPI / I2C •
- Temporizadores • Y más



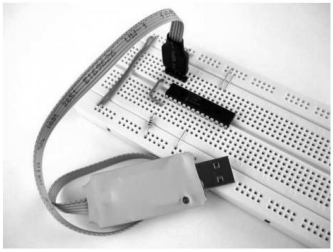
PIC16F886

Las conexiones al programador se muestran en la siguiente imagen. Los módulos CHIPINO se encargan de todas estas conexiones por usted, pero puede construirlo usted mismo en una placa de prueba como se muestra.



Programación en serie en circuito (ICSP)

Machine Translated by Google



PIC en una placa de pruebas

1.2 Uso de Great Cow Basic

El compilador Great Cow Basic se puede descargar del sitio web greatcowbasic.com como un paquete de software completo. Las instrucciones de instalación se encuentran a continuación.

Instalación

Requisitos:
PC con Windows con XP o superior (o Mac con Parallels/Windows,
Linux ejecutando Wine)
Puerto USB

Hay dos formas de instalar Great Cow Basic CHIPINO (GCBC) en su PC; Instalación Automática o Instalación Manual.

Instalación Automática

1) Descargue el archivo de configuración de GCBC y ejecútelo en su computadora. Instalará GCBC en su menú de inicio y también colocará un ícono en su escritorio.



Sample Version - Do Not Copy

Machine Translated by Google

Instalación manual 1) Si

está disponible, también puede descargar el archivo GCBCIDEVx.zip del sitio web greatcowbasic.com.



2) Descomprímalo en un directorio de su computadora.



3) Abra la carpeta GCBCIDEVx y verá varias carpetas.



4) Haga clic derecho en el símbolo de la vaca para crear un acceso directo en su escritorio.

Sample Do Not Copy

Machine Translated by Google



5) Ahora debería tener un ícono de GCBC en su escritorio.



6) La instalación está completa. Estás listo para crear tu primer proyecto.

Sample Version - Do Not Copy

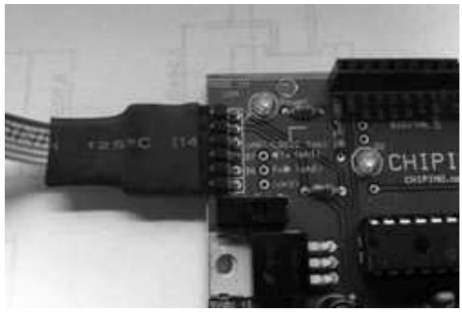
1.3 Creando tu primer proyecto

1) Conecte el cable de programación a un puerto USB abierto.

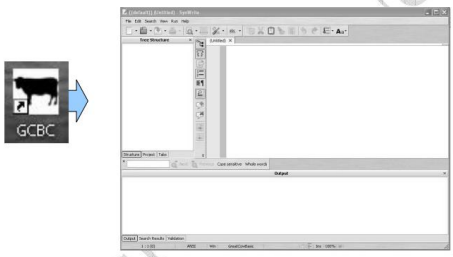


2) Conectar el cable de programación al CHIPINO. Asegúrese de que la flecha roja (junto al LED de encendido) esté línea en el cable se alinea con el ^ CHIPINO.

Machine Translated by Google



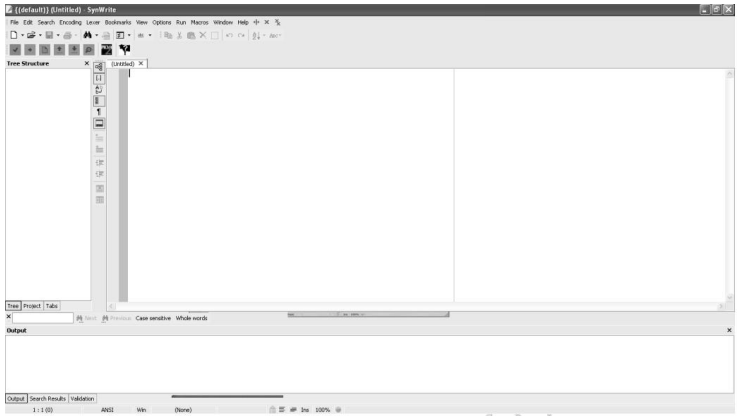
3) Haga doble clic en el icono de GCBC para iniciar el IDE de GCBC.



4) Verá una pantalla en blanco la primera vez que abra el IDE.

Sample Version Not Copy

Machine Translated by Google



En la parte superior verá la barra de herramientas de GCBC para crear proyectos de software.



Representan las siguientes funciones:

 Compile el archivo GCB e informe cualquier error (esto creará un archivo .hex).

 Compile y luego programe el CHIPINO a través del Cable de Programación PICkit2.

 Abre una nueva ventana en blanco.

 Abra el archivo GCB existente.

 Guarde el archivo GCB.

 Abra un archivo de terminal para la comunicación a través de la computadora

Sample Version Do Not

Machine Translated by Google

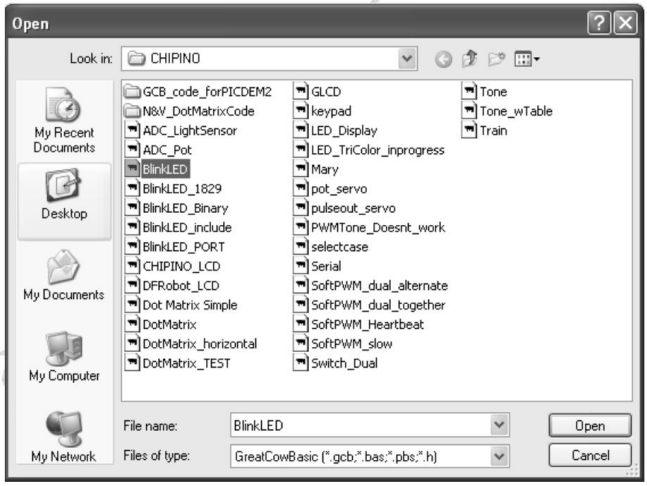
Puerto serial. Esto requiere un blindaje RS232 separado y un puerto RS232 en su PC o un cable convertidor de RS232 a USB.

 Abra el software PICkit 2 Standalone GUI para poder utilizar el PICkit 2 Cable de programación con Terminal o Analizador Lógico integrado. Consulte el Apéndice A para obtener más detalles sobre estas funciones.

 Abra el archivo de ayuda de GCB con detalles sobre los comandos y funciones de GCB.

5) Ahora abra el archivo BlinkLED.GCB de la carpeta CHIPINO.

Haga clic en y luego seleccione el archivo BlinkLED.gcb de la carpeta CHIPINO. Haga clic en el botón Abrir.



6) Debería ver el programa BlinkLED.gcb en la ventana del editor.

Sample - Do Not Copy

Machine Translated by Google

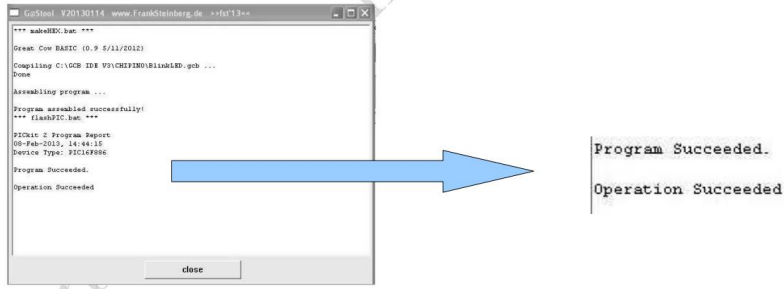
BlinkLED.gcb*

```
'A program to flash LED on PIN 13/PORTB5

'Chip model
#chip 16F886, 16

'Main routine
Start:
 'Turn one LED on, the other off
    SET PORTB.5 ON
    Wait 1 sec
 'Now toggle the LEDs
    SET PORTB.5 OFF
    Wait 1 sec
 'Jump back to the start of the program
Goto Start
```

7) Haga clic en o simplemente presione la tecla F5. El compilador básico Great Cow convertirá el archivo BlinkLED.gcb en los 1 y 0 que el microcontrolador, dentro del módulo CHIPINO, necesita ejecutar. Aparecerá una ventana que muestra que la compilación y la programación del CHIPINO fueron exitosas. El LED comenzará a parpadear en el CHIPINO.



Acabas de programar con éxito el CHIPINO por primera vez.

Sample Version - DO Not Copy

Machine Translated by Google

1.4 Opciones de la línea de comandos

Si usted es más del tipo técnico de usuario que quiere crear sus programas en el editor de texto y luego programar a través de la línea de comandos en lugar de a través del IDE, GCB en el corazón es un compilador de línea de comandos. Aquí se incluyen las opciones para este tipo de programación.

GCBASIC [/O:salida.asm] [/A:ensamblador] [/P:programador] [/K:{C|A}] [/V] [/L][/NP] nombre de archivo

Cambiar	Descripción	
/O:nombre de archivo	Establece el nombre del archivo de ensamblaje generado en nombre de archivo.	Predeterminado Mismo nombre que el archivo de entrada, pero con extensión .asm.
/A:archivo por lotes del ensamblador utilizado para llamar	ensamblador*. Si se proporciona / A:GCASM, GCBASIC utilizará su ensamblador interno.	El programa no se ensamblará.
/ P: archivo por lotes del programador utilizado para llamar	programador*. Este parámetro se ignora si el programa no está ensamblado.	El programa no se descargará.
/K:{C|A}	Mantenga el código original en la salida del ensamblado. /K:C guardará los comentarios, /K:A conservará todo el código de entrada.	No queda código original en la salida.
/V	Modo detallado: el compilador brinda información más detallada sobre sus actividades.	Ninguno
/L	Mostrar licencia y salir.	Ninguno
/NOTARIO PÚBLICO	No se detenga en los errores. Uso con IDE.	Haga una pausa cuando ocurra un error y espere a que el usuario presione una tecla.
Nombre del archivo	El archivo a compilar.	Ninguno

* Para los modificadores /A: y /P:, hay opciones especiales disponibles.

Si %FILENAME% está presente, será reemplazado por el nombre del archivo .asm.

Si %FN_NOEXT% será reemplazado por el nombre del archivo .asm pero sin extensión y %CHIPMODEL% será reemplazado por el nombre del chip. El nombre del chip será el mismo que el del archivo de datos del chip.

Sample Version - Do Not Copy

Machine Translated by Google

2.1 Entradas/Salidas

La mayoría de los pines de propósito general en un microcontrolador pueden funcionar en uno de dos modos: modo de entrada o modo de salida.

Al actuar como entrada, el pin se colocará en estado de alta impedancia. Luego, el microcontrolador detectará el pin y el programa puede leer el estado del pin y tomar decisiones basadas en él.

Cuando está en modo de salida, el microcontrolador conectará el pin a Vcc (el suministro positivo) o Vss (tierra o suministro negativo) según los comandos del programa GCB. Hay varios comandos que pueden establecer el estado del pin.

GCB intentará determinar la dirección de cada pin y configurarlo adecuadamente.
Sin embargo, si el pin se lee y se escribe durante el programa, entonces el pin debe configurarse en modo de entrada/salida por el programa, usando el comando Dir apropiado.

Sintaxis:
Dir port.bit {In | Out} (Forma Individual)
Puerto dir {En | Fuera | DirectionByte} (formulario de puerto completo)

Explicación: El
comando Dir se usa para establecer la dirección de los puertos del chip del microcontrolador. El formulario individual establece la dirección de un pin a la vez, mientras que el formulario de puerto completo establecerá todos los bits en un puerto. Esto controla el registro TRIS dentro del PIC si está familiarizado con el funcionamiento interno.

En el formulario individual, especifique el puerto y el bit (es decir, PORTB.4), luego la dirección, que es In o Out.

La forma completa del puerto es similar a la instrucción TRIS que ofrecen algunos chips PIC.
Para usarlo, proporcione el nombre del puerto (es decir, PORTA), y luego se debe escribir un byte en la variable TRIS. Esta forma de comando es para aquellos que están familiarizados con la arquitectura interna del chip PIC.

ADVERTENCIA: Los PIC usan 0 para salir y 1 para entrar. Cuando se usan IN y OUT, no hay problemas de compatibilidad.

Sample Version - Do Not Copy

Machine Translated by Google

```
Ejemplo: 'Este
    programa establece los bits 0 y 1 de PORTA en entrada y el resto en salida.
    'También configura todo PORTB para salida, excepto B1.
    'Se utiliza forma individual para PORTA:
    DIR PORTA.0 EN
    DIR PORTA.1 IN DIR
    PORTA.2 OUT DIR
    PORTA.3 OUT DIR
    PORTA.4 OUT DIR
    PORTA.5 OUT DIR
    PORTA.6 OUT DIR
    PORTA.7 OUT 'Todo el
    formulario de puerto utilizado para
    B: DIR PORTB b'00000010'


    'Todo el formulario de puerto utilizado para C:
    ENTRADA PUERTO DIR
```

2.2 Configuración

Cada chip PIC tiene una palabra CONFIG. Esta es un área de memoria en el chip que almacena configuraciones que gobiernan el funcionamiento del chip.

Los siguientes aspectos del chip se rigen por la palabra CONFIG:

•Selección del oscilador: ¿funcionará el chip desde un oscilador interno o se adjunta uno externo?

• Restablecimientos automáticos: ¿debería restablecerse el chip si la potencia cae demasiado? ¿Si detecta que está ejecutando el mismo código una y otra vez?

•Protección de código: ¿qué áreas de la memoria deben mantenerse ocultas una vez escritas?

•Uso de pines: ¿qué pines están disponibles para programar, restablecer el chip o emitir señales PWM?

Los ajustes de configuración exactos varían entre los chips. Para encontrar una lista de configuraciones válidas, consulte la hoja de datos del chip PIC que desea usar.

Todo esto puede ser bastante confuso; por lo tanto, GCB establecerá automáticamente algunos ajustes de configuración, a menos que se indique lo contrario:

•La programación de bajo voltaje (LVP) está desactivada. Esto permite que el pin PGM (generalmente B3 o B4) se use como un pin de E/S normal.

Sample Version - Do Not Copy

Machine Translated by Google

•El temporizador de vigilancia (WDT) está apagado. El WDT restablece el chip si ejecuta la misma pieza de código una y otra vez; esto puede causar problemas con algunas de las rutinas de demora más largas en GCBASIC.

•Master Clear (MCLR) está deshabilitado siempre que sea posible. En muchos chips más nuevos, esto permite que el pin MCLR (a menudo PORTA.5) se use como un puerto de entrada estándar. También elimina la necesidad de una resistencia pull-up en el pin MCLR.

•Se seleccionará un modo de oscilador, según las siguientes reglas en este orden de prioridad:

1. Si el PIC tiene un oscilador interno y el oscilador interno está capaz de generar la velocidad especificada en la línea #chip, entonces se utilizará el oscilador interno.
2. Si la velocidad del reloj es superior a 4 MHz, se selecciona el oscilador HS externo. 3. Si la velocidad del reloj es de 4 MHz o menos, se selecciona el modo de oscilador XT externo.

Tenga en cuenta que estos ajustes se pueden anular individualmente cuando sea necesario. Por ejemplo, si se necesita el temporizador de vigilancia, agregar la siguiente línea habilitará el temporizador de vigilancia, sin afectar ninguna otra configuración.

```
#config WDT = ENCENDIDO
```

Uso de la configuración

Una vez que se han determinado las opciones CONFIG necesarias, agregarlas al programa es fácil. En una nueva línea, escriba "#config" y luego enumere las opciones deseadas separadas por comas, como en esta línea:

```
#config OSC = RC, BODEN = APAGADO
```

GCB también admite este formato en chips de la serie 10/12/16:

```
#config INTOSC_OSC_NOCLKOUT, BODEN_OFF
```

Sin embargo, para compatibilidad ascendente con chips 18F, debe usar la primera opción de configuración.

Es posible tener varias líneas #config en un programa, por ejemplo, una en el programa principal y una en cada uno de varios archivos #include. Sin embargo, se debe tener cuidado para asegurarse de que la configuración de un archivo no entre en conflicto con la de otro.

Sample Version - Do Not Copy

Machine Translated by Google

2.3 Variables

Una variable es un área de memoria en el microcontrolador que se puede usar para almacenar un número o una serie de letras. Esto es útil para muchos propósitos, como tomar la lectura de un sensor y actuar en consecuencia, o contar la cantidad de veces que el robot ha realizado una tarea en particular.

Cada variable debe recibir un nombre, como "MyVariable" o "PieCounter".
Elegir un nombre para una variable es fácil: simplemente no incluya espacios ni ningún símbolo (que no sea _), y asegúrese de que el nombre tenga al menos 2 caracteres (letras y/o números).

Tipos de variables Hay
varios tipos diferentes de variables, y cada tipo puede almacenar un tipo diferente de información. Estos son los tipos de variables que Great Cow BASIC puede usar actualmente:

Tipo de variable	Información que esta variable puede almacenar	Ejemplos de usos para este tipo de variable
Poco	Un poco (0 o 1)	Indicadores para rastrear si una pieza de código se ha ejecutado o no
Byte	Un número entero entre 0 y 255	Almacenamiento de datos de propósito general, como contadores
Palabra	Un número entero entre 0 y 65535	Almacenamiento de números extra grandes
Entero	Un número entero entre - 32768 y 32767	Cualquier cosa donde ocurrirá un número negativo
Largo	Un número entero entre 0 y 2^32 (4290 millones)	Almacenar números muy, muy grandes
Formación	Una lista de números enteros que van del 0 al 255	Registros de lecturas de sensores
Cadena	Una serie de letras, números y símbolos.	Mensajes que se van a mostrar en una pantalla

Sample Version - Do Not Copy

Uso de variables Las
variables de byte se crean automáticamente en Great Cow Basic. Simplemente ingrese el nombre de la variable en el comando donde se necesita la variable y el compilador creará la variable automáticamente.

Otros tipos de variables deben "dimensionarse" primero. Esto implica usar el comando DIM para decirle a Great Cow BASIC que se trata de algo más que una variable de byte.

Machine Translated by Google

```
Dim Var como Word 'Variable VAR ahora puede contener de 0 a 65,535
Indicador atenuado como bit 'La variable de bandera tendrá un 0 o un 1
```

Una característica clave de las variables es que es posible hacer que el microcontrolador verifique una variable y solo ejecute una sección de código si es un valor dado. Esto se puede hacer con el comando IF.

Alias de variables Algunas

variables son alias, que se utilizan para hacer referencia a ubicaciones de memoria utilizadas por otras variables. Estos son útiles para unir variables de byte predefinidas para formar variables de palabra.

Los alias no son como punteros en muchos idiomas: siempre deben hacer referencia a la misma variable o variables y no se pueden cambiar.

Un alias útil es HighByte, Low Byte para variables de palabra.

Ejemplo:

```
Dim BigVar como alias de palabra HighByte, LowByte

Byte alto = 2
Byte bajo = 100
```

También puede acceder a bits de una variable como esta. Sin embargo, recuerde que el primer bit es el bit 0, no el bit 1. Por lo tanto, los ocho bits de una variable de byte son del 0 al 7.

```
Si Var.2 = 1, entonces ' Pruebe el bit 3 de la variable Var para ver si está establecido en 1
configure D13 en End ' Luz LED en D13 si está configurado
If
```

Configuración de variables

Sintaxis:

```
variable = datos
```

Sample Version - Do Not Copy

Machine Translated by Google

Explicación:

La variable se establecerá en datos. Los datos pueden ser un valor fijo (como 157), otra variable o una suma.

Si los datos son un valor fijo, debe ser un número entero entre 0 y 255 inclusive.

Si los datos son un cálculo, entonces pueden tener cualquiera de los siguientes operandos:

+ (añadir)

- (restar o negar si no hay ningún valor antes)

* (multiplicar)

/ (dividir)

% (módulo)

& (y)

| (o)

(xor)

! (no)

= (igual)

<> (no igual)

< (menor que)

> (mayor que)

<= (menor o igual)

>= (mayor o igual)

Sample Version - Do Not Copy

Machine Translated by Google

Los 6 operandos finales son para verificar condiciones. Devolverán FALSO (0) si la condición es falsa, o VERDADERO (255) si la condición es verdadera.

Los operadores And, Or, Xor y Not funcionan tanto como operadores bit a bit como lógicos.

GCBASIC entiende el orden de las operaciones. Si hay varios operandos presentes, se procesarán en este orden:

1. Soportes
2. Operaciones unarias (not y negate)
3. Multiplicar/Dividir/Módulo
4. Sumar/Restar
5. Operadores condicionales
6. Y/O/Xo

Hay varios modos en los que se pueden configurar las variables. GCBASIC utilizará automáticamente un modo diferente para cada cálculo, dependiendo del tipo de variable que se establezca. Si se establece una variable de byte, se utilizará el modo de byte; si se establece una variable de palabra, se utilizará el modo de palabra. Si se está configurando un byte pero el cálculo involucra números mayores de 255, se puede usar el modo palabra agregando [PALABRA] al comienzo de uno de los valores en el cálculo. Esto se conoce como fundición.

Si lo prefiere, puede agregar "LET" al comienzo de la línea. No alterará la ejecución del programa, pero está incluido para aquellos que están acostumbrados a incluirlo en otros dialectos BASIC.

Ejemplo:

```
'Este programa es para ilustrar la configuración de variables.

ardilla listada = 46        'Establece la variable Ardilla en 46

Animal = ardilla listada          'Establece la variable Animal al valor de la variable Ardilla
```

Sample Version - Do Not Copy

Machine Translated by Google

```
Oso = 2 + 3 * 5              'Establece la variable Bear en el resultado de 2 + 3 * 5, 17.

Oveja = (2 + 3) * 5 'Establece la variable Oveja al resultado de (2 + 3) * 5, 25.

Animal = 2 * Oso             'Establece la variable Animal al doble del valor de Oso.

GranVar = 321                'LargeVar debe configurarse como una palabra; consulte DIM.

Temp = LargeVar / [WORD]5 'Tenga en cuenta el uso de [WORD] para asegurarse de que el cálculo se realiza correctamente
```

2.4 Fundición

La conversión cambia el tipo de una variable o valor. Colocar el tipo al que se debe convertir el valor entre corchetes le indicará al compilador que lo convierta. Por ejemplo, esto hará que dos variables de byte sean tratadas como variables de palabra por el código de adición:

```
Dim MyWord como Word
MiPalabra = [palabra]VarByte + OtraVarByte
```

¿Por qué hacer esto? Si no hay conversiones, GCBASIC agregará los dos valores usando el código de adición de bytes y luego convertirá el resultado en una palabra para almacenar en MyWord. Supongamos que ByteVar es 150 y AnotherByteVar es 231. Cuando se agrega, esto llegará a 381, que se desbordará, dejando 125 en el resultado. Sin embargo, cuando se agrega el molde, GCBASIC tratará a ByteVar como si fuera una palabra, por lo que utilizará el código de adición de palabras. Esto hará que se calcule el resultado correcto.

A menudo, se utilizará un molde al calcular un promedio:

```
MiPromedio = ([palabra]Valor1 + Valor2) / 2
```

También es posible lanzar el segundo valor:

```
MiPromedio = (Valor1 + [palabra]Valor2) / 2
```

El resultado será exactamente el mismo.

2.5 Definiciones/Constantes

Sintaxis:

```
#define Buscar Reemplazar
```

Sample Version - Do Not Copy

Machine Translated by Google

Explicación:
#define buscará en el programa Buscar y lo reemplazará con el valor dado para Reemplazar.

Una definición o constante es un tipo de directiva que le dice al compilador que busque una palabra determinada y la reemplace con otra palabra o número. Las definiciones son útiles para situaciones en las que es necesario modificar fácilmente una rutina. Por ejemplo, se podría usar una definición para especificar la cantidad de tiempo para ejecutar una alarma una vez activada.

También es posible usar defines para especificar puertos; por lo tanto, defines se puede usar para ayudar en la creación de código que se puede adaptar fácilmente para ejecutarse en un PIC diferente con diferentes puertos.

GCBASIC hace un uso considerable de define internamente. Por ejemplo, el código LCD usa defines para establecer los puertos que debe usar para comunicarse con la pantalla LCD.

Uso de define

Para crear una definición es cuestión de usar la directiva #define. Estos son algunos ejemplos de definiciones:

#define Línea 34

Line es una constante simple: GCBASIC encontrará "Line" en el programa y lo reemplazará con el número 34. Esto podría usarse en un programa de seguimiento de línea, para que sea más fácil calibrar el programa para diferentes condiciones de iluminación.

#define Luz PORTB.0

La luz es un puerto: representa un pin particular en el chip PIC. Esto sería útil si el programa tuviera muchas líneas de código que controlaran la luz, y existiera la posibilidad de que el puerto al que estaba conectada la luz tuviera que cambiar en el futuro.

#define LightOn Activar PORTB.0

Sample Version - Do Not Copy

Machine Translated by Google

LightOn es una definición utilizada para hacer que el programa sea más legible. En lugar de escribir "Set PORTB.0 on" una y otra vez, sería posible escribir "LightOn" y hacer que el compilador haga el trabajo duro.

2.6 Constantes numéricas

GCB permite definir constantes numéricas en tres bases: decimal, binaria y hexadecimal. Los decimales son los predeterminados y no requieren prefijo. Los números binarios y hexadecimales se pueden escribir en un par de formatos, pero cada uno requiere un prefijo.

decimales:
170

Binario:
0b10100101 ' Equivalente binario a 170
o
b'10100101'

Maleficio:
H'A5' ' Equivalente hexadecimal a 170
o
0xA5

2.6.1 BCD a decimal A veces

es necesario formatear números en formato decimal codificado en binario. GCB no tiene la función incorporada, pero puede agregarla fácilmente con esta función a continuación. Simplemente agregue esto a su programa GCB y luego llámelo cuando lo necesite.

```
Función DecToBcd(va) como Byte
DecToBcd=(va/10)*16+va%10
función final
```

Ejemplo:
BCDNumber = DecToBcd(number) 'Convertir número de variable a valor BCD

2.6.2 De decimal a BCD

A veces es necesario convertir números del formato decimal codificado en binario a decimal. GCB no tiene la función incorporada, pero puede agregarla fácilmente con esta función a continuación. Simplemente agregue esto a su programa GCB y luego llámelo cuando lo necesite.

Sample Version - Do Not Copy

Machine Translated by Google

```
Función BcdToDec(va) como byte
BcdADec=(va/16)*10+va%16
función final
```

Ejemplo:
DecNumber = BcdToDec(number) 'Convertir número variable a valor decimal

2.7 Constantes de cadena Las
variables de cadena son útiles para muchos comandos de comunicación, como enviar a una pantalla LCD. Las cadenas son caracteres que se colocan entre comillas.

Si una variable se establece igual a una cadena, se utilizará el valor equivalente ASCII.

"A" se leerá como el valor ASCII de 65 "d" se
leerá como el valor ASCII de 100

Las cadenas se pueden combinar en una sola frase entre comillas.
Por ejemplo:

La cadena "Hola" es lo mismo que "H", "e", "l", "l", "o"

Así es como las cadenas pueden ser útiles para enviar cartas a una pantalla LED.

```
'Este programa mostrará "Hola" en una pantalla LED
'La pantalla está conectada a PORTB

Dim mensaje como cadena
Mensaje = "Hola"
Para Contador = 1 a 6
        DisplayChar 1, mensaje (contador)
        Espere 250ms
Próximo
```

También puede combinar cadenas como variables.

```
Dim TestString$ como cadena
DIM OtherString$ como cadena

TestString$ = “Hola” “ 'Contiene hola
OtraCadena$ = CadenaPrueba$ + World” 'OtherString$ contiene Hello World

'En la pantalla LCD
Imprimir TestString$ 'Imprimir Hola
Imprimir OtherString$ 'Imprimir Hola Mundo
```

Sample Version - Do Not Copy

Machine Translated by Google

2.8 Conversión de cadenas

Estos comandos facilitan la conversión de variables y constantes de cadenas a otros formatos.

2.8.1 Cadena a

hexadecimal La función hexadecimal convertirá un número a formato hexadecimal. El número de entrada debe ser una variable de byte o un número fijo entre 0 y 255 inclusive. Después de ejecutar la función, la variable de cadena stringvar contendrá un número hexadecimal de 2 dígitos.

Sintaxis:

```
stringvar = Hex(número)
```

Ejemplo:

```
'Usa hexadecimal para mostrarse como hexadecimal
Para CurrentLocation = 0 a 255
        HSerPrint Hex (Ubicación actual)
Próximo
```

2.8.2 Cadena a valor decimal La

función Val extraerá un número de una variable de cadena y lo almacenará en una variable de palabra. Un uso potencial es leer números que se envían en formato ASCII a través de una conexión en serie.

Sintaxis:

```
var = Val(cadena)
```

Ejemplo:

```
'Variables para bytes recibidos
Dim DataIn como cadena

'Convertir nivel de salida a variable numérica
Nivel de salida = Val (Entrada de datos)

'Producción
HPWM 1, 32, nivel de salida
```

2.8.3 Buscar cadena dentro de una

cadena La función Instr buscará una cadena para encontrar la ubicación de otra cadena dentro de ella. source es la cadena para buscar dentro, y find es la cadena para buscar. La función devolverá la ubicación de búsqueda dentro de la fuente, o 0 si la fuente no contiene búsqueda.

Sintaxis:

Sample Version - Do Not Copy

Machine Translated by Google

ubicación = Instr(fuente, buscar)

Ejemplo:
Dim TestData As String
TestData = "¡Hola, mundo!"

'Muestra la ubicación de "mundo" dentro de la cadena 'Devolverá 8, porque "w" en mundo es el
octavo carácter 'de "¡Hola, mundo!"

HSerPrint Instr(TestData, "mundo")
HSerImprimirCRLF

'Muestra la ubicación de "planeta" dentro de la cadena 'Mostrará 0, porque "planeta" no aparece
dentro de 'la cadena "¡Hola, mundo!"

HSerPrint Instr(TestData, "planeta")
HSerImprimirCRLF

2.8.4 Convertir a minúsculas

La función LCase convertirá todas las letras en la fuente de la cadena a minúsculas y devolverá el resultado.

Sintaxis:
salida = LCase(fuente)

Ejemplo:
'Establecer modelo de
chip #chip 16F1936

'Configurar conexión serial de hardware #define
USART_BAUD_RATE 9600 #define USART_BLOCKING

'Rellena una cadena con un mensaje Dim TestData As
String TestData = "¡Hola, mundo!"

'Mostrar la cadena en minúsculas 'Mostrará "¡hola, mundo!"

HSerPrint LCase(TestData)
HSerImprimirCRLF

2.8.5 Convertir a mayúsculas La

función UCase convertirá todas las letras de la cadena fuente a mayúsculas y devolverá el resultado.

Sample Version - Do Not Copy

Machine Translated by Google

Sintaxis:
salida = UCase (fuente)

Ejemplo:
```
Dim TestData As String TestData =
"¡Hola, mundo!"

'Mostrar la cadena en mayúsculas 'Mostrará "¡HOLA,
MUNDO!"
HSerPrint UCase(TestData)
```

2.8.6 Convertir número en cadena La

función Str convertirá un número en una cadena. número puede ser cualquier variable de byte o palabra, o un número fijo entre 0 y 65535 inclusive. La variable de cadena stringvar contendrá el mismo número, representado como una cadena.

Esta función es especialmente útil si es necesario agregar un número al final de una cadena, o si se ha creado una rutina de envío de datos personalizada pero solo admite la salida de variables de cadena.

Sintaxis:
stringvar = Str(número)

Ejemplo:
```
'Tomar una lectura A/D
SensorReading = ReadAD(AN0)

'Crear una variable de cadena
Dim OutVar como cadena

'Llene la cadena con la lectura del sensor
OutVar = Str(Lectura del sensor)

'Enviar
HSerImprimir OutVar
HSerImprimirCRLF
```

2.8.7 Recuperar caracteres de cadena (izquierda)

La función Left extraerá los caracteres de conteo más a la izquierda del origen de la cadena de entrada y los devolverá en una nueva cadena.

Sintaxis:
salida = Izquierda (fuente, conteo)

Ejemplo:
```
Dim TestData As String TestData =
"¡Hola, mundo!"
```

Sample Version - Do Not Copy

Machine Translated by Google

```
'Mostrar los 5 caracteres más a la izquierda
'Mostrará "Hola"
HSerPrint Izquierda (TestData, 5)
```

2.8.8 Recuperar caracteres de cadena (medio)

La función Mid se usa para extraer caracteres del medio de una variable de cadena. fuente es la variable de la que se extrae, inicio es la posición del primer carácter que se extrae y recuento es el número de caracteres que se extraen. Si no se especifica el recuento, se devolverán todos los caracteres desde el principio hasta el final de la cadena de origen.

Sintaxis:

```
salida = Mid (fuente, inicio, conteo)
```

Ejemplo:

```
'Extraer "gato". La c está en la posición 5 y se necesitan 3 letras HSerPrint "El animal es un

HSerPrint Mid(TestData, 5, 3)

'Extraer la acción. "sat" comienza en la posición 9.
HSerImpresión "El animal"
HSerPrint Mid(TestData, 9)
HSerImprimirCRLF
```

2.8.9 Recuperar caracteres de cadena (derecha)

La función Derecha extraerá los caracteres de conteo más a la derecha del origen de la cadena de entrada y los devolverá en una nueva cadena.

Sintaxis:

```
salida = Derecha (fuente, recuento)
```

Ejemplo:

```
Dim TestData As String TestData =
"¡Hola, mundo!"

'Mostrar los 6 caracteres más a la derecha 'Mostrará
"world!"
HSerImprimir Derecha(TestData, 6)
HSerImprimirCRLF
```

2.9 Operaciones Matemáticas Si los datos

son un cálculo, entonces pueden tener cualquiera de los siguientes operandos:

Sample Version - Do Not Copy

Machine Translated by Google

+ (sumar)

- (restar o negar si no hay ningún valor antes) (multiplicar) /
* (dividir) % (módulo) ++ (incrementar) (decrementar)

--

Operadores bit a bit y lógicos:

& (y) | (o) #
(xor) ! (no)
= (igual) <>
(no igual) <
(menor que)
> (mayor que) <=
(menor que o igual)
>= (mayor que o igual)

Los 6 operandos finales son para verificar condiciones. Devolverán FALSO (0) si la condición es falsa, o VERDADERO (255) si la condición es verdadera.

Los operadores And, Or, Xor y Not funcionan tanto como operadores bit a bit como lógicos.

GCBASIC entiende el orden de las operaciones. Si hay varios operandos presentes, se procesarán en este orden:

1. Corchetes
2. Operaciones unarias (no y negar)
3. Multiplicar/Dividir/Módulo 4.
Sumar/Restar 5. Operadores
condicionales 6. Y/O/Xor

Hay varios modos en los que se pueden configurar las variables. GCBASIC utilizará automáticamente un modo diferente para cada cálculo, dependiendo del tipo de variable que se establezca. Si se establece una variable de byte, se utilizará el modo de byte; si se establece una variable de palabra, se utilizará el modo de palabra.

Sample Version - Do Not Copy

Machine Translated by Google

Si se está configurando un byte pero el cálculo involucra números mayores de 255, se puede usar el modo palabra agregando [PALABRA] al comienzo de uno de los valores en el cálculo. Esto se conoce como fundición (sección 2.4).

Si lo prefiere, puede agregar "LET" al comienzo de la línea. No alterará la ejecución del programa, pero está incluido para aquellos que están acostumbrados a incluirlo en otros dialectos BASIC.

Ejemplo:

```
'Este programa es para ilustrar la configuración de variables.

ardilla listada = 46            'Establece la variable Ardilla en 46

Animal = Ardilla 'Establece la variable Animal al valor de Ardilla

Oso = 2 + 3 * 5          'Establece la variable Oso al resultado de 2 + 3 * 5, 17

Ovejas = (2 + 3) *       5 'Establece la variable Ovejas al resultado de (2 + 3) * 5, 25

animales = 2 * Oso 'Establece la variable Animal al doble del valor de Oso

GranVar = 321                   'LargeVar debe configurarse como una palabra; consulte DIM

Temp = LargeVar / [WORD]5 'LargeVar es una variable de tamaño de palabra.
'Tenga en cuenta el uso de [PALABRA] para asegurarse de que el cálculo se realiza correctamente
```

Sample Version - Do Not Copy

2.10 Condiciones

En GCBASIC (y en la mayoría de los demás lenguajes de programación), una condición es una declaración que puede ser verdadera o falsa. Las condiciones se utilizan cuando el programa debe tomar una decisión.

Una condición generalmente se da como un valor o variable, un operador relativo (como = o >) u otro valor o variable. Se pueden combinar varias condiciones para formar una condición mediante el uso de operadores lógicos como AND y OR.

GCBASIC admite estos operadores relativos:

Símbolo	Significado
=	Igual
<>	No es igual
<	Menos que

Machine Translated by Google

>	Mas grande que
<=	Menos que o igual a
>=	Igual o mayor que

Además, estos operadores lógicos se pueden utilizar para combinar varias condiciones en una sola:

Nombre	Abreviatura	Condición verdadera si
Y	&	ambas condiciones son verdaderas
O	|	al menos una condición es verdadera
XOR	#	una condición es verdadera
NO	!	la condición no es cierta

NOT es ligeramente diferente a los otros operadores lógicos, ya que solo necesita otra condición. Otros operadores aritméticos pueden combinarse en condiciones para cambiar los valores antes de compararlos.

GCBASIC tiene dos condiciones integradas: VERDADERO, que siempre es verdadero, y FALSO, que siempre es falso. Estos se pueden utilizar para crear bucles infinitos.

También es posible probar bits individuales en condiciones. Para hacer esto, especifique el bit a probar, luego 1 o 0 (o ON y OFF respectivamente). Actualmente no hay forma de combinar pruebas de bits con otras condiciones: NOT, AND, OR y XOR no funcionarán.

Condiciones de ejemplo:

Condición	Comentarios
temperatura = 0	La condición es verdadera si Temp = 0
Sonda <> 0	La condición es verdadera si el sensor no es 0
Lectura1 > Lectura2	Verdadero si Lectura1 es mayor que Lectura2
Modo = 1 Y Tiempo > 10	Verdadero si el modo es 1 y el tiempo es más de 10
Calor > 5 O Humo > 2	Verdadero si Calor es más de 5 o Humo es más de 2
Luz >= 10 Y (NO Tiempo > 7)	Verdadero si la luz es 10 o más y el tiempo es 7 o menos
Temp.0 ENCENDIDO	Verdadero si Temp bit 0 está activado

Sample Version - Do Not Copy

2.11 Combinación de varias instrucciones

Es posible combinar varias instrucciones en una sola línea, separándolas con dos puntos. Por ejemplo, este código:

Machine Translated by Google

```
Activar PORTB.0 Activar
PORTB.1 Esperar 1 s
Activar PORTB.0 Desactivar
PORTB.0 Desactivar PORTB.0
```

También podría escribirse como:

```
Activar PORTB.0: Activar PORTB.1
Espera 1 segundo
Desactivar PORTB.0: Desactivar PORTB.0
```

En la mayoría de los casos, no hará ninguna diferencia si los comandos comparten una línea o no. Sin embargo, se debe tener especial cuidado con los comandos "IF", como este código:

```
Desactivar PORTB.0
Desactivar PORTB.1
Si la temperatura es > 10, entonces configure PORTB.0 en: Configure PORTB.1 en espera 1
s
```

Será equivalente a esto:

```
Desactivar PORTB.0
Desactivar PORTB.1
Si Temp > 10, entonces configure
        PORTB.0 en
        Activar PORTB.1
Terminara si
Espera 1 s
```

Además, los comandos utilizados para iniciar y finalizar subrutinas, tablas de datos y funciones deben estar solos en una línea. Por ejemplo, esto es INCORRECTO:

```
Sub Algo: Desactivar PORTB.0: Finalizar sub
```

2.12 Comentarios La

adición de comentarios a su programa GCB se realiza con un apóstrofo antes de la línea de comentarios. También puede comentar secciones de código si lo desea simplemente colocando un apóstrofe al comienzo de cada línea. El SynGCB IDE tiene una función para hacer esto automáticamente. Comentará o comentará en bloques de código que resalte. Hay un icono para comentar y comentar en la barra de herramientas central.



Comentar y descomentar iconos

Example Version - Do Not Copy

Machine Translated by Google

En algunos programas de muestra de GCB también puede ver líneas de comentarios que comienzan con un punto y coma (;) o el término REM.

```
;configuraciones de chips
Rem Esta es una línea de comentarios.
```

Ninguno de estos son reconocidos automáticamente por el formateo automático de IDE, por lo que parecerán líneas de comando, pero el compilador Great Cow Basic los reconocerá como líneas comentadas y no los incluirá en la compilación final.

Es mejor usar el apóstrofe para todas las líneas comentadas para que el IDE las muestre como líneas de comentarios no resaltadas. Esto hará que su código sea más fácil de leer.
Los íconos de la barra de herramientas para comentar/quitar comentarios agregarán apóstrofes o los eliminarán.

Ejemplo

```
'La cantidad de pines para flashear #define
FlashPins 2

'Puede crear un encabezado usando un apóstrofe antes de cada línea. 'Esta es una excelente
manera de describir su programa. 'También puede usarlo para describir las conexiones de hardware.

'Puede colocar comentarios encima del comando o en la misma línea
Dir PORTB Out 'Inicializar PORTB a todas las salidas

'Lazo principal
hacer
    PUERTOB = 0 'Todos los pines apagados
    Espera 1 segundo 'Retraso 1 segundo
    PUERTOB = 0xFF 'Todos los pines en
    Espera 1 s 'Retraso 1 segundo
Bucle
```

Sample Version - Do Not Copy

Machine Translated by Google

3.0 Directivas

3.1 #chip

Sintaxis:
    #modelo de chip, velocidad

Explicación: La
directiva #chip se usa para especificar el modelo de chip y la velocidad para la que debe compilar GCBASIC. model es el modelo del chip microcontrolador, algo parecido a "16F819". speed es la velocidad del chip en MHz, y se requiere para las rutinas de retardo y PWM.

Si se omite la velocidad, GCBASIC utilizará la velocidad de reloj más alta que el oscilador interno pueda soportar. Si el chip no tiene un oscilador interno, entonces GCBASIC supondrá que el chip se ejecuta a su máxima velocidad de reloj posible utilizando un cristal externo.

Ejemplos:
#chip 12F509, 4 #chip
18F4550, 48 #chip
16F88, 0.125 #chip
tiny2313, 1 #chip mega8,
16

3.2 #config

Sintaxis: #config
    opción1, opción2, ... , opciónn

Explicación: la
directiva #config se usa para especificar opciones de configuración para el chip. Hay una explicación detallada de #config en la sección Configuración de la ayuda.

3.3 #definir

Sintaxis:
    #define Buscar Reemplazar

Sample Version - Do Not Copy

Machine Translated by Google

Explicación:
#define buscará en el programa Buscar y lo reemplazará con el valor dado para Reemplazar.

3.4 #si

Sintaxis:

```
#if Condición
...
#terminara si
```

Explicación: la
directiva #if se usa para evitar que se compile una sección de código a menos que Condition sea verdadera.

La condición tiene la misma sintaxis que la condición en un comando if normal de GCBASIC. La única diferencia es que usa constantes en lugar de variables.
Ejemplo:

```
'Este programa pulsará un número ajustable de pines en PORTB 'El número de
pines es controlado por la constante FlashPins #chip 16F88, 8


'La cantidad de pines para flashear #define
FlashPins 2

'Inicializar
Dir PUERTO Salida

'Bucle principal
Hacer
    #if FlashPins >= 1 PulseOut
        PORTB.0, 250 ms #endif #if FlashPins
    >= 2 PulseOut PORTB.1, 250 ms #endif #if
    FlashPins >= 3 PulseOut PORTB.2, 250
        ms #endif #if FlashPins >= 4 PulseOut
    PORTB.3, 250ms #endif



Bucle
```

Sample Version - Do Not Copy

Machine Translated by Google

3.5 #ifdef

Sintaxis:

```
#ifdef Constante | Valor constante | Var(NombreVariable)
...
#terminara si
```

Explicación: la
directiva #ifdef se usa para habilitar selectivamente secciones de código. Hay varias formas en las que se puede utilizar: •Comprobar si una constante está definida •Comprobar si una constante está definida y tiene un valor particular •Comprobar si existe una variable de sistema •Comprobar si se ha definido un bit de sistema

La ventaja de usar #ifdef en lugar de una serie equivalente de sentencias IF es la cantidad de código que se descarga en el chip. #ifdef controla qué código se compila y descarga, IF controla qué se ejecuta una vez en el chip. #ifdef debe usarse siempre que se deba verificar el valor de una constante.

GCBASIC también es compatible con la directiva #ifndef, que es lo opuesto a la directiva #ifdef, eliminará el código que deja #ifdef y viceversa.

(Nota: el código de las siguientes secciones no se compilará, ya que le faltan las directivas #chip y los comandos DIR. Está destinado a actuar solo como ejemplo).

Código de habilitación si se define una constante

Ejemplo de sintaxis:

```
#definir Parpadeo1

#ifdef Blink1 PulseOut
        PORTB.0, 1 seg Espere 1 seg #endif
        #ifdef Blink2 PulseOut PORTB.1, 1 seg
Espere 1 seg #endif
```

Este código pulsará PORTB.0, pero no PORTB.1. Esto se debe a que Blink1 se ha definido, pero Blink2 no. si la línea

```
#definir Parpadeo2
```

se agregó al comienzo del programa, entonces ambos pines serían pulsados. El valor de la constante definida no es importante y se puede dejar fuera de #define.

Sample Version - Do Not Copy

Machine Translated by Google

Código de habilitación si se define una constante y tiene un valor dado

Ejemplo de sintaxis:

```
#define PinsToFlash 2

#ifdef PinsToFlash 1,2,3
        PulseOut PORTB.0, 1 seg #endif
#ifdef PinsToFlash 2,3

        PulseOut PORTB.1, 1 seg #endif
#ifdef PinsToFlash 3 PulseOut PORTB.2, 1
seg #endif
```

Este programa usa una constante llamada PinsToFlash que controla cuántas luces se pulsan. PORTB.0 se pulsa cuando PinsToFlash es igual a 1, 2 o 3, PORTB.1 se pulsa cuando PinsToFlash es igual a 2 o 3, y PORTB.2 parpadea cuando PinsToFlash es 3.

Código de habilitación si se define una variable del sistema

Ejemplo de sintaxis:

```
#ifdef NoVar(ANSEL)
        FIJAR ADCON1.PCFG3 APAGADO
        FIJAR ADCON1.PCFG2 ENCENDIDO
        FIJAR ADCON1.PCFG1 ENCENDIDO
        FIJAR ADCON1.PCFG0 APAGADO
#endif
#ifdefVar(ANSEL)
        ANSEL = 0
#terminara si
```

La sección de código anterior se ha copiado directamente de ad.h. Se usa para deshabilitar la función A/D de los pines, para que puedan usarse como puertos de E/S digital estándar. Si ANSEL no se declara como una variable del sistema para un chip en particular, entonces el programa usa ADCON1 para controlar los modos de puerto. Si se define ANSEL, entonces el chip es más nuevo y sus puertos se pueden configurar en digital borrando ANSEL.

Código de habilitación si se define un bit de sistema

Similar al anterior, excepto con Bit y NoBit en lugar de Var y NoVar respectivamente.

Sample Version - Do Not Copy

Machine Translated by Google

3.6 #ifndef

Sintaxis:

```
#ifndef Constante | Valor constante | Var(NombreVariable)
...
#terminara si
```

Explicación: la
directiva #ifndef se usa para habilitar selectivamente secciones de código. Es lo opuesto a la directiva #ifdef: eliminará el código en los casos en que #ifdef lo dejaría, y dejará el código donde #ifdef lo eliminaría.

3.7 #incluir

Sintaxis:

```
#incluye nombre de archivo
```

Explicación:
#include le dice a GCBASIC que abra otro archivo, lea todas las subrutinas y constantes y luego las copie en el programa actual.

Hay dos formas de inclusión: absoluta y relativa.

Absoluto se usa para hacer referencia a archivos en el directorio C:\Program Files\GCBASIC\include. El nombre del archivo se especifica entre los símbolos < y >. Por ejemplo, para incluir el archivo "srf04.h", la directiva es:

```
#incluye <srf04.h>
```

Relativo se usa para leer archivos en la misma carpeta que el programa seleccionado actualmente. Los nombres de archivo se dan entre comillas, como:

```
#include "micódigo.h"
```

donde mycode.h es el nombre del archivo que se va a leer.

No es esencial que el nombre del archivo de inclusión termine en .h; lo importante es que el nombre dado a GCBASIC sea el nombre exacto del archivo que se incluirá.

Aquellos que estén familiarizados con #include en ensamblador o C deben tener en cuenta que #include en GCBASIC funciona de manera diferente a #include en la mayoría de los otros lenguajes :

Sample Version - Do Not Copy

Machine Translated by Google

el código no se inserta en la ubicación de #include, sino al final del programa actual.

Si se instala un convertidor de archivos adecuado , se puede usar #include para incluir archivos que no sean GCBASIC. GCBASIC detectará que la extensión del archivo incluido coincide con un convertidor y ejecutará el convertidor apropiado para importar el archivo.

3.8 #script

Sintaxis:

```
#script
    [scriptcommand1]
    [scriptcommand2]
    ...
    [comando de secuencia de
comandos] #endscript
```

Explicación: el
bloque #script se usa para crear pequeñas secciones de código que GCBASIC ejecuta durante la compilación. Se incluye una explicación detallada y un ejemplo en el artículo de Scripts.

3.9 #inicio

Sintaxis:

```
#subnombre de inicio
```

Explicación:
#startup se usa en archivos de inclusión para insertar automáticamente rutinas de inicialización. Si se utiliza una definición o subrutina del archivo en el programa, se llamará a la subrutina especificada.

Hay algunas limitaciones en esta directiva. Solo puede ocurrir una vez dentro de un archivo y no se pueden especificar parámetros para la subrutina que se va a llamar.

Sample Version - Do Not Copy

Machine Translated by Google

4.0 Funciones avanzadas

4.1 Matrices

Una matriz es un tipo especial de variable, que puede almacenar varios valores a la vez. Es esencialmente una lista de números en la que cada uno puede ser abordado individualmente mediante el uso de un "índice". El índice es un valor entre paréntesis inmediatamente después del nombre de la matriz.

Ejemplos de nombres de matrices son:

Matriz/Índice	Significado
pescado(10)	Elemento 10 de una matriz llamada Fish
Registro de datos(2)	El segundo número en una matriz llamada DataLog
Lista de botones (Temp)	Un elemento en la matriz "ButtonList" que se selecciona de acuerdo con el valor de la variable "Temp"

Uso de matrices Para utilizar una matriz, se especifica su nombre, luego el índice. Las matrices se pueden usar en cualquier lugar donde se pueda usar una variable normal.

Configuración de una matriz completa a la vez Es posible configurar varios elementos de una matriz con una sola línea de código. Este breve ejemplo muestra cómo:

```
Dim TestVar(10)
TestVar = 1, 2, 3, 4, 5, 6, 7, 8, 9
```

El elemento 0 de TestVar se establecerá en el número de elementos de la lista, que en este caso es 9. Cada elemento de la matriz se cargará con el valor correspondiente en la lista; por lo tanto, en el ejemplo, TestVar(1) establecerse en 1, TestVar(2) en 2, y así sucesivamente.

Si hay muchos elementos en la matriz, puede ser mejor usar una tabla de búsqueda para almacenar los elementos y luego copiar algunos de los elementos de datos en una matriz más pequeña según sea necesario.

Sample Version - Do Not Copy

Machine Translated by Google

4.2 Funciones

Las funciones son un tipo especial de subrutina que puede devolver un valor. Esto significa que cuando el nombre de la función se usa en lugar de una variable, GCBASIC llamará a la función, obtendrá un valor de ella y luego pondrá el valor en la línea de código en el lugar de la variable.

Las funciones también pueden tener parámetros; estos se tratan exactamente de la misma manera que los parámetros de las subrutinas. La única excepción es que se requieren corchetes alrededor de cualquier parámetro al llamar a una función.

Uso de funciones

Este programa usa una función llamada AverageAD para tomar dos lecturas analógicas y luego tomar una decisión basada en el promedio:

```
'Seleccionar chip
#chip 16F88, 20

'Define puertos
#define LED PORTB.0 #define
Sensor AN0

'Establecer direcciones de puerto
dir LED apagado
dir PORTA.0 en

'Código principal

Hacer
        Desactivar PORTB.0
        Si AverageAD > 128, entonces configure PORTB.0 en espera
        10 ms
Bucle

Función PromedioAD 'Obtenga
        2 lecturas, divida por 2, almacene en PromedioAD 'Observe el lanzamiento, el
        resultado de ReadAD debe convertirse en 'una palabra antes de agregar, o el resultado puede
        desbordarse.
       PromedioAD = ([palabra]ReadAD(Sensor) + ReadAD(Sensor)) / 2

función final
```

Sample Version - Do Not Copy

4.3 Interrupciones

Las interrupciones son una característica de muchos microcontroladores. Permiten que el microcontrolador pause (interrumpa) temporalmente el código que está ejecutando y luego comience a ejecutarse

Machine Translated by Google

otra pieza de código cuando ocurre algún evento. Una vez que se ha ocupado del evento, volverá a donde estaba y continuará ejecutando el programa.

Muchos eventos pueden desencadenar una interrupción, como que un temporizador llegue a su límite, que se reciba un mensaje en serie o que un pin especial en el microcontrolador reciba una señal.

Uso de interrupciones
Hay dos formas de usar interrupciones en GCBASIC. La primera forma es usar el comando On Interrupt. Esto habilitará automáticamente una interrupción dada y ejecutará una subrutina particular cuando ocurra la interrupción.

La otra forma de lidiar con las interrupciones es crear una subrutina llamada Interrupción. GCBASIC llamará a esta subrutina cada vez que ocurra una interrupción, y luego su código puede verificar los bits de "bandera" para determinar qué interrupción ha ocurrido y qué se debe hacer al respecto. Si utiliza este enfoque, deberá habilitar las interrupciones deseadas manualmente. También es esencial que su código borre los bits de bandera, o de lo contrario, la rutina de interrupción se llamará repetidamente.

También es posible alguna combinación de estos dos métodos: el código generado por On Interrupt con verificación para ver si la interrupción es una que reconoce. Si se reconoce la interrupción, On Interrupt se encargará de ella; si no, se llamará a la subrutina Interrupt para que se ocupe de la interrupción.

La forma recomendada es usar On Interrupt, ya que es más eficiente y más fácil de configurar.

Durante algunas secciones del código, es deseable que no ocurra ninguna interrupción. Si este es el caso, utilice el comando IntOff para deshabilitar las interrupciones al comienzo de la sección e IntOn para volver a habilitarlas al final. Si se producen eventos de interrupción mientras las interrupciones están deshabilitadas, se procesarán tan pronto como se vuelvan a habilitar. Si el programa no utiliza interrupciones, GCBASIC eliminará automáticamente IntOn e IntOff.

Sample Version - Do Not Copy

4.4 Tablas de consulta

Una tabla de búsqueda es una lista de valores que se almacenan en la memoria del programa del chip, a la que se puede acceder mediante el comando ReadTable.

La ventaja de las tablas de búsqueda es que son eficientes en memoria, en comparación con un conjunto equivalente de declaraciones IF.

Machine Translated by Google

Uso de tablas de búsqueda
Primero, se debe crear la tabla. El código para crear una tabla de búsqueda es simple: una línea que tiene "Tabla" y luego el nombre de la tabla, una lista de números (hasta 254) y luego "Tabla final".

Una vez que se crea la tabla, el comando ReadTable se usa para leer datos de ella.
El comando ReadTable requiere el nombre de la tabla que se va a leer, la ubicación del elemento a recuperar y una variable para almacenar el número recuperado.

Las tablas de búsqueda pueden almacenar valores de bytes o palabras. GCBASIC detectará automáticamente el tipo de tabla en función de los valores que contenga.

El elemento 0 de una tabla de búsqueda almacena el tamaño de la tabla. Si el comando ReadTable intenta leer más allá del final de la tabla, se devolverá el valor 0.

4.5 Guiones

Un script es una pequeña sección de código que ejecuta Great Cow BASIC cuando comienza a compilar un programa. Su uso principal es realizar los cálculos que se requieren para ajustar el programa para diferentes chips de velocidad.

Los scripts no se compilan ni se descargan en el microcontrolador: GCBASIC los lee, los ejecuta, los elimina del programa y luego permite que los resultados que hayan calculado se usen como constantes en el programa.

Dentro de un script, las constantes se tratan como variables. Los scripts pueden leer los valores de las constantes y configurarlos para que contengan nuevos valores.

Uso de secuencias
de comandos Las secuencias de comandos comienzan con "#script" y terminan con "#endscript". En su interior, pueden constar de 3 comandos: •Si •Asignación (=) •Error

If es similar al comando If en el código GCBASIC normal, excepto que no tiene una cláusula Else.
Se utiliza para comparar los valores de las constantes.

= es idéntico al de los programas GCBASIC. La constante que se va a configurar va del lado izquierdo del =, y el nuevo valor va del lado derecho.

Sample Version - Do Not Copy

Machine Translated by Google

Error se utiliza para mostrar un mensaje de error. Cualquier cosa después del comando Error se muestra al final de la compilación y se guarda en el registro de errores del programa.

Script de ejemplo

Este script se utiliza en el archivo pwm.h. Toma los valores de las constantes PWM_Freq, PWM_Duty y ChipMHz, y usa las ecuaciones que se muestran en las hojas de datos de PIC para calcular los valores correctos para las variables relevantes del sistema.

```
#script PR2Temp = int((1/PWM_Freq)/(4*(1/(ChipMHz*1000))))

T2PR = 1 Si
PR2Temp > 255 Entonces
    PR2Temp = Int((1 / PWM_Freq) / (16 * (1 / (ChipMHz * 1000))))
    T2PR = 4 Si
    PR2Temp > 255 Entonces
        PR2Temp = Int(( 1 / PWM_Freq) / (64 * (1 / (ChipMHz * 1000))))
        T2PR = 16 Si
        PR2Temp > 255 Entonces Error
            Valor de frecuencia PWM no válido Terminar si
        Terminar si Terminar si



DutyCycle = (PWM_Duty * 10.24) * PR2Temp / 1024 DutyCycleH =
(DutyCycle Y 1020) / 4 DutyCycleL = DutyCycle Y 3 #endscript
```

Después de ejecutar este script, los valores PR2Temp, DutyCycleH y DutyCycleL se utilizan como constantes para configurar las variables requeridas.

4.6 Subrutinas

Una subrutina es un pequeño programa dentro del programa principal. Las subrutinas se utilizan normalmente cuando una tarea debe repetirse varias veces en diferentes partes del programa principal.

Hay dos usos principales para las subrutinas:

•Mantener los programas ordenados y fáciles de leer •Reducir el tamaño de los programas al permitir que se reutilicen secciones comunes de código.

Cuando el PIC llega a una subrutina, guarda su ubicación en el programa actual antes de saltar al inicio o llamar a la subrutina. Una vez que llega al final de la subrutina, regresa al programa principal y continúa ejecutando el código donde lo dejó anteriormente.

Sample Version - Do Not Copy

Machine Translated by Google

Normalmente, es posible que las subrutinas llamen a otras subrutinas. Hay límites para la cantidad de veces que una subrutina puede llamar a otra subrutina, que varían de un chip a otro: Familia PIC 10F*, 12C5*, 12F5*, 16C5*, 16F5*

	Ancho de instrucción	Número de suscriptores llamados
	12	1
12C*, 12F*, 16C*, 16F*, excepto los anteriores	14	7
18F*, 18C*	dieciséis	31

Estos límites se deben a la cantidad de memoria en el PIC que guarda su ubicación antes de saltar a una nueva subrutina. Algunos comandos de GCBASIC son subrutinas, por lo que siempre debe permitir 2 o 3 llamadas a subrutinas más de las que tiene su programa.

Otra característica de las subrutinas es que pueden aceptar parámetros. Estos son valores que se pasan del programa principal a la subrutina cuando se llama y luego se devuelven cuando finaliza la subrutina.

Uso de subrutinas

Llamar a una subrutina es muy simple: todo lo que se necesita es el nombre de la subrutina y luego una lista de parámetros. Este código llamará a una subrutina llamada "Buzz" que no tiene parámetros:

```
Zumbido
```

Si el sub tiene parámetros, deben aparecer después del nombre de la subrutina. Este sería el comando para llamar a una subrutina llamada "MoveArm" que tiene tres parámetros:

```
MoveArm NuevoX, NuevoY, 10
```

Si una subrutina tiene parámetros, puede optar por colocarlos entre paréntesis, así:

```
MoveArm (NuevoX, NuevoY, 10)
```

Todo lo que esto hace es cambiar la apariencia del código; no hace ninguna diferencia en lo que hace el código. Decide cuál cumple con tus preferencias personales y luego mantenlo.

Sample Version - Do Not Copy

Machine Translated by Google

Crear subrutinas

Crear una subrutina es casi tan simple como usar una. Debe haber una línea al principio que tenga "sub", y luego el nombre de la subrutina. Además, debe haber una línea al final de la subrutina que diga "end sub". Para crear una subrutina llamada "Buzz", este es el código requerido:

```
zumbido secundario

'el código para la subrutina va aquí

final sub
```

Si la subrutina tiene parámetros, entonces deben enumerarse después del nombre. Por ejemplo, para definir el sub "MoveArm" usado anteriormente, use este código:

```
sub MoveArm(ArmX, ArmY, ArmZ)

'el código para la subrutina va aquí

final sub
```

En el sub anterior, ArmX, ArmY y ArmZ son todas variables. Si se utiliza la llamada desde arriba, las variables tendrán estos valores al inicio de la subrutina:

```
BrazoX = NuevoX
Ejército = NuevoY
Brazo Z = 10
```

Cuando la subrutina haya terminado de ejecutarse, GCBASIC copiará los valores donde sea posible. NewX se establecerá en ArmX y NewY en ArmY. GCBASIC no intentará establecer el número 10 en ArmZ.

Controlar la dirección en la que se mueven los datos

Es posible indicar a GCBASIC que no vuelva a copiar el valor después de llamar a la subrutina. Si una subrutina se define así:

```
sub MoveArm(En ArmX, En ArmY, En ArmZ)

'el código para la subrutina va aquí

final sub
```

Entonces GCBASIC copiará los valores a la subrutina, pero no los copiará.

Sample Version Do Not Copy

Machine Translated by Google

También se puede evitar que GCBASIC vuelva a copiar los valores agregando "Out" antes del nombre del parámetro. Esto se usa en las rutinas de lectura de EEPROM: no tiene sentido copiar un valor de datos en la subrutina de lectura, por lo que se ha usado Out para evitar perder tiempo y memoria. La rutina EPRead se define como "Sub EPRead (Dirección de entrada, Datos de salida).

Muchas secciones antiguas de código usan "#NR" al final de la línea donde se especifican los parámetros. El "#NR" significa "Sin retorno", y cuando se usa tiene el mismo efecto que agregar "In" antes de cada parámetro. No se recomienda el uso de "#NR", ya que no brinda el mismo nivel de control.

Uso de diferentes tipos de datos para parámetros

Es posible utilizar cualquier tipo de variable como parámetro de una subrutina. Simplemente agregue "Como" y luego el tipo de datos al final del nombre del parámetro. Por ejemplo, para hacer que todos los parámetros para las variables de palabra de la subrutina MoveArm, use este código:

```
sub MoveArm (ArmX como palabra, ArmY como palabra, ArmZ como palabra)
...
final sub
```

Parámetros opcionales

A veces, el mismo valor puede usarse una y otra vez para un parámetro, excepto en un caso particular. Si esto ocurre, se puede especificar un valor predeterminado para el parámetro, y luego solo se necesita dar un valor para ese parámetro en una llamada si es diferente al predeterminado.

Por ejemplo, suponga que se requiere una subrutina para crear un pitido de error. Normalmente emite un tono de 440 Hz, pero a veces se requiere un tono diferente. Para crear el sub, se usaría este código:

```
Sub ErrorBeep (Tono de salida opcional como palabra = 440)
    Tono fueraTono, 100
Finalizar sub
```

Tenga en cuenta el "Opcional" antes del parámetro y el " = 440" después. Esto le dice a GCBASIC que si no se proporciona ningún parámetro, establezca el parámetro OutTone en 440.

Si se llama usando esta línea:

```
pitido de error
```

luego se emitirá un pitido de 440 Hz. Si se llama usando esta línea:

Sample Version - Do Not Copy

Machine Translated by Google

ErrorBeep 1000

entonces el subwoofer producirá un tono de 1000 Hz.

Cuando se utilizan varios parámetros, es posible hacer que cualquier número de ellos sea opcional. Si los parámetros opcionales están al final de la llamada, entonces no es necesario especificar ningún valor. Si están al principio o en el medio, debe insertar comas para permitir que GCBASIC le indique dónde están los parámetros opcionales.

Sobrecarga

Es posible tener 2 subrutinas con el mismo nombre, pero diferentes parámetros. Esto se conoce como sobrecarga y GCBASIC seleccionará automáticamente la subrutina más adecuada para cada llamada.

Un ejemplo de esto es la rutina Imprimir en las rutinas LCD. En realidad, hay 3 subrutinas de impresión; uno tiene un parámetro de byte, otro un parámetro de palabra y otro un parámetro de cadena. Si se utiliza este comando:

Imprimir 100

Luego se llamará a la subrutina Imprimir (byte). Sin embargo, si se utiliza este comando:

Imprimir 30112

Luego se llamará a la subrutina Imprimir (palabra). Si no hay una coincidencia exacta para una llamada en particular, GCBASIC utilizará la opción que requiera la menor conversión de tipos de variables. Por ejemplo, si se utiliza este comando:

Imprimir PUERTO.0

Se utilizará la impresión de bytes. Esto se debe a que el byte es el tipo más cercano al parámetro de un solo bit.

Sample Version - Do Not Copy

4.7 Descripción general del teclado PS2

Introducción

Estas rutinas facilitan la comunicación con un dispositivo PS/2, en particular un teclado.

Machine Translated by Google

Constantes relevantes

Estas constantes afectan el funcionamiento de las rutinas PS/2:

Nombre constante	Control S	Valor por defecto
Datos de PS2	Pin conectado a la línea de datos PS/2	N / A
PS2Reloj	Pin conectado a la línea de reloj PS/2.	N / A
PS2_DELAY	Esta constante se puede establecer en un retraso, como 10 ms. Si se establece, se agregará un retraso al final de cada byte enviado o recibido.	no establecido

Simple Version - Do Not Copy

4.8 Resumen aleatorio

Introducción:

Estas rutinas permiten a GCBASIC generar números pseudoaleatorios.

El generador utiliza un registro de desplazamiento de retroalimentación lineal de 16 bits para producir números pseudoaleatorios. Los 8 bits más significativos del LFSR se utilizan para proporcionar un número aleatorio de 8 bits.

Al compilar un programa, GCBASIC generará una semilla inicial para el generador. Sin embargo, esta semilla será la misma cada vez que se ejecute el programa, por lo que la secuencia de números producidos por un programa determinado siempre será la misma.
Para evitar esto, existe una subrutina Randomize. Se puede proporcionar con una nueva semilla para el generador (lo que hará que el generador se mueva a un punto diferente en la secuencia). Alternativamente, Randomize se puede configurar para obtener una semilla de alguna otra fuente, como un temporizador, cada vez que se ejecuta.

Machine Translated by Google

Constantes relevantes:

Estas constantes se utilizan para controlar la configuración de las rutinas de generación de tonos. Para configurarlos, coloque una línea en el archivo del programa principal que use #define para asignar un valor a la constante particular.

Nombre constante	Control S	Valor por defecto
RANDOMIZE_SEED Fuente de la semilla aleatoria si Randomize se llama sin un parámetro		Temporizador0

Sample Version - Do Not Copy

4.9 Descripción general de I2C

Introducción:

Estas rutinas permiten que los programas GCBASIC envíen y reciban mensajes I2C.
Se pueden configurar para actuar como maestro o esclavo, y también se puede modificar la velocidad.

No se requiere ningún módulo I2C de hardware para estas rutinas: toda la comunicación se maneja en el software. Sin embargo, estas rutinas no funcionarán en PIC de instrucción de 12 bits (chips 10F, 12F5xx y 16F5xx).

Constantes relevantes:

Estas constantes controlan la configuración de las rutinas I2C del software.

Constante	Control S	Valor por defecto
MODO_I2C	Modo de rutinas I2C (Maestro o Esclavo)	Maestro
I2C_DATOS	Pin en microcontrolador conectado a datos I2C	N / A
I2C_CLOCK	Pin en el microcontrolador conectado al reloj I2C	N / A
I2C_BIT_DELAY	Tiempo por un bit (utilizado para la detección de reconocimiento)	2 nosotros
I2C_CLOCK_DELAY Tiempo para que el pulso del reloj		1 nosotros

Machine Translated by Google

	permanecer alto	
I2C_END_DELAY	Tiempo entre pulsos de reloj	1 nosotros
I2C_USE_TIMEOUT Establecido	en verdadero si el modo esclavo Las rutinas I2C deberían dejar de esperar al maestro y salir después de un tiempo de espera ocurre.	no establecido

4.10 Descripción general del temporizador

Se proporcionan varias funciones para leer el valor actual del temporizador. Ellos son:

•Temporizador0

•Temporizador1

•Temporizador2

•Temporizador3

•Temporizador4

•Temporizador5

No todas estas funciones están disponibles en todos los chips. Por ejemplo, si un chip solo tiene 3 temporizadores, solo estarán disponibles Timer0, Timer1 y Timer2. Timer0 y Timer2 devuelven valores de bytes, mientras que Timer1, Timer3, Timer4 y Timer5 devolverán palabras.

Consulte la hoja de datos de su microcontrolador para determinar la cantidad y el tamaño de los temporizadores disponibles.

Simple Version - Do Not Copy

4.11 Resumen de siete segmentos

Introducción

Las rutinas de pantalla de 7 segmentos facilitan que los programas GCBASIC muestren números y letras en pantallas LED de 7 segmentos.

Machine Translated by Google

Hay dos formas de configurar las rutinas de visualización de 7 segmentos. Una opción es conectar los cables de las pantallas en un orden particular y luego establecer las constantes DisplayPortn y DispSelectn. La otra opción es conectar la/s pantalla/s de la manera que sea más fácil y luego establecer las constantes DISP_SEG_x y DISP_SEL_x. La primera opción (configurar DisplayPortx y DispSelectn) generará un código un poco más eficiente.

Configuración mediante DISP_SEG_x y DISP_SEL_x

Al configurar el código de 7 segmentos con constantes DISP_SEG_x, estos deben configurarse:

Nombre constante	Control S	Valor por defecto
DISP_SEG_x	Controla el ping de salida utilizado para controlar el segmento x de las pantallas. Hay 7 de estas constantes, llamadas DISP_SEG_A a DISP_SEG_G. Se debe configurar uno para cada segmento.	N / A
DISP_SEG_DOT	Especifica el pin de salida utilizado para controlar el punto decimal en las pantallas.	N / A
DISP_SEL_x	El comando utilizado para seleccionar la pantalla n. Se utiliza para controlar los pines de direccionamiento cuando se multiplexan varias pantallas.	N / A

Sample Version - Do Not Copy

Nota: En lugar de configurar DISP_SEL_x, es posible configurar DispSelectn y usarlos junto con DISP_SEG_x.

Machine Translated by Google

Configuración usando DisplayPortn y DispSelectn

Para configurar las rutinas de visualización de 7 segmentos suministradas con GCBASIC usando DisplayPortn, es necesario establecer estas constantes:

Nombre constante	Control S	Valor por defecto
DisplayPortn	Controla el puerto de salida utilizado para controlar la pantalla n. n es A, B, C o D, correspondientes a las pantallas 1, 2, 3 y 4, respectivamente.	N / A
DispSeleccionar	El comando utilizado para seleccionar la pantalla n. Se utiliza para controlar los pines de direccionamiento cuando se multiplexan varias pantallas.	nop

Para configurar las rutinas de esta manera, las pantallas deben estar conectadas de la siguiente manera:

Pin del puerto del microcontrolador	Segmento de visualización
0	A
1	B
2	C
3	D
4	mi
5	F
6	GRAMO

Sample Version - Do Not Copy

4.12 Descripción general de PWM

Introducción:

Las rutinas descritas en este capítulo permiten la generación de señales de modulación de ancho de pulso. Estos permiten que el microcontrolador controle la velocidad de un

Machine Translated by Google

motor, o el brillo de una luz. Las rutinas también se pueden usar para generar la señal de frecuencia adecuada para impulsar un LED infrarrojo para aplicaciones de control remoto.

Las rutinas PWMOn, PWMOff y HPWM utilizan el módulo de generación de PWM en el microcontrolador. Solo funcionarán en algunos microcontroladores; consulte el artículo sobre cada comando para obtener más información. PWMOn y HPWM harán que el módulo PWM en el microcontrolador comience a generar la señal PWM, que luego continuará generándose hasta que se ejecute la instrucción PWMOff.

PWMOut no utiliza ningún hardware especial. Sin embargo, solo se genera una señal mientras se ejecuta el comando PWMOut: cuando el programa pasa al siguiente comando, la señal se detiene.

Constantes relevantes:

Estas constantes se utilizan para controlar la configuración del módulo de modulación de ancho de pulso del chip PIC. Para configurarlos, coloque una línea en el archivo del programa principal que use #define para asignar un valor a la constante particular.

Tenga en cuenta que hay dos conjuntos de constantes: uno para Hardware PWM y otro para Software PWM. El hardware PWM requiere un módulo CCP en el chip PIC. El software PWM no tiene requisitos con respecto al PIC.

Hardware PWM

Estas constantes solo son necesarias para PWMOn. HPWM y PWMOff no requieren constantes para funcionar.

Nombre constante	Control S	Valor por defecto
PWM_Freq	Especifica la frecuencia de salida del módulo PWM en KHz.	38
PWM_deber	Establece el ciclo de trabajo de la salida del módulo PWM. Dado como porcentaje.	50

Sample Version - Do Not Copy

Machine Translated by Google

Hardware PWM solo está disponible a través del pin "CCP1" o "CCP". Esta es una limitación de hardware de los microcontroladores PIC.

Software PWM

Nombre constante	Control S	Valor por defecto
PWM_Delay	El período PWM. La duración de cualquier demora utilizada se multiplicará por 255. Si no se especifica ningún valor, no se insertarán retrasos en la rutina PWM.	No definido - sin demora
PWM_Fuera	El puerto puerto físico en el PIC que corresponde al canal n. n puede representar 1, 2, 3 o 4.	No definida

Son posibles más de 4 canales, pero para esto se debe modificar la rutina PWMOut en include\lowlevel\stdbasic.h.

Sample Version - Do Not Copy

4.13 Descripción general del software RS232

Introducción:

Estas rutinas permiten que el microcontrolador envíe y reciba datos RS232.

Todas las funciones se implementan mediante software, por lo que no se requiere hardware especial en el microcontrolador. Sin embargo, si el microcontrolador tiene un módulo serial de hardware (generalmente denominado UART o USART) y las líneas de datos seriales están conectadas a los pines apropiados, las rutinas de hardware deben usarse para código más pequeño, confiabilidad mejorada y velocidades de transmisión más altas.

Machine Translated by Google

Constantes relevantes:

Estas constantes se utilizan para controlar la configuración de las rutinas de comunicación serie RS232. Para configurarlos, coloque una línea en el archivo del programa principal que use #define para asignar un valor a la constante particular.

Nombre(s) constante(s)	Control S	Valor por defecto
Enviar bajo, enviar bajo, EnviarCbajo	Estos se utilizan para definir los comandos utilizados para enviar un bit bajo (0) en los canales serie A, B y C, respectivamente.	nop (se debe establecer)
EnviarAlto, EnviarBAlto, enviarCalto	Estos se utilizan para definir los comandos utilizados para enviar un bit alto (1) en los canales serie A, B y C, respectivamente.	nop (se debe establecer)
RecALbajo, RecBLbajo, RecCL bajo	La condición que se cumple cuando se recibe un bit bajo	Sys232Temp.0 APAGADO (se debe establecer)
RecAalta, RecBalta, recchigh	La condición que se cumple cuando se recibe un bit alto	Sys232Temp.0 ENCENDIDO (se debe establecer)

Sample Version - Do Not Copy

4.14 Descripción general del hardware RS232

Introducción

Estas subrutinas permiten que los programas GCBASIC se comuniquen más fácilmente usando RS232.

Estas rutinas basadas en hardware están diseñadas para usarse en microcontroladores con módulos RS232 incorporados, normalmente denominados en las hojas de datos como módulos USART o UART. Para usarlos, las líneas de datos RS232 deben estar conectadas a los pines del microcontrolador utilizado por el módulo serial. Si las líneas RS232 están conectadas en otro lugar, o el microcontrolador no tiene un módulo RS232, entonces se deben usar las rutinas basadas en software.

Machine Translated by Google

Constantes relevantes

Estas constantes afectan el funcionamiento de las rutinas del hardware RS232:

Nombre constante	Control S	Valor por defecto
USART_BAUD_RATE Velocidad	en baudios (en bps) a la que operarán las rutinas.	N / A
USART_BLOCKING Si se establece, esta constante hará que	las rutinas USART se retrasen hasta que se puedan enviar o recibir datos. Si no se establece, los datos se almacenarán en búfer y el hardware los enviará cuando sea posible.	no establecido

Sample Version - Do Not Copy

4.15 Descripción general del teclado

Introducción

Las rutinas del teclado permiten que un programa lea desde un teclado de matriz de 4 x 4.

Hay dos formas de configurar las rutinas del teclado. Una opción es conectar los cables del teclado en un orden particular y luego establecer la constante KeypadPort. La otra opción es conectar el teclado de la manera que sea más fácil y luego configurar las constantes KEYPAD_ROW_x y KEYPAD_COL_x.
La primera opción (configurar KeypadPort) generará un código un poco más eficiente.

Configuración usando KEYPAD_ROW_x y KEYPAD_COL_x

Estas constantes deben establecerse:

Nombre constante	Control S	Valor por defecto
TECLADO_ROW_1	El pin en el microcontrolador que se conecta al pin de la Fila 1 en el teclado	N / A
TECLADO_ROW_2	El pin en el microcontrolador que se conecta a la Fila 2	N / A

Machine Translated by Google

	pin en el teclado	
TECLADO_ROW_3	El pin en el microcontrolador que se conecta al pin de la Fila 3 en el teclado	N / A
TECLADO_ROW_4	El pin del microcontrolador que se conecta al pin de la fila 4 en el teclado	N / A
TECLADO_COL_1	El pin en el microcontrolador que se conecta al pin Col 1 en el teclado	N / A
TECLADO_COL_2	El pin en el microcontrolador que se conecta al pin Col 2 en el teclado	N / A
TECLADO_COL_3	El pin en el microcontrolador que se conecta al pin Col 3 en el teclado	N / A
TECLADO_COL_4	El pin en el microcontrolador que se conecta al pin Col 4 en el teclado	N / A

Si usa un teclado de 3 x 3, no configure las constantes KEYPAD_ROW_4 o KEYPAD_COL_4.

Sample Version - Do Not Copy

Configuración usando KeypadPort

Al configurar el código del teclado usando la constante KeypadPort, solo KeypadPort debe configurarse:

Nombre constante	Control S	Valor por defecto
TecladoPuerto	El puerto en el chip del microcontrolador al que está conectado el teclado.	N / A

Machine Translated by Google

Para que esto funcione, el teclado debe estar conectado de la siguiente manera:

Pin del puerto del microcontrolador	Conector de teclado
0	Fila 1
1	Fila 2
2	Fila 3
3	Fila 4
4	columna 1
5	columna 2
6	columna 3
7	columna 4

Nota: Para usar un teclado de 3 x 3 en este modo, se deben levantar los pines del microcontrolador para cualquier columna no utilizada.

Sample Version - Do Not Copy

4.16 Vista general de la pantalla LCD

Introducción:

Las rutinas de esta sección permiten que los programas GCBASIC interactúen con pantallas de cristal líquido alfanuméricas basadas en el IC HD44780. Esto cubre la mayoría de las pantallas de 16 x 2 y similares.

Estas rutinas permiten que las pantallas se conecten al microcontrolador de muchas maneras diferentes:

Modo de conexión	Conexiones requeridas
0	Ninguno es requerido directamente por las rutinas. Sin embargo, las rutinas LCD deben estar provistas de otras subrutinas que manejarán la comunicación. Esto es útil para comunicarse con LCD conectados a través de RS232 o I2C.
2	Líneas de datos y reloj. Este modo se usa cuando la pantalla LCD está conectada a través de un IC de registro de desplazamiento 74LS174, como se detalla en http://www.rentron.com/Myke1.htm.

Machine Translated by Google

4	R/W, RS, Habilitar y las 4 líneas de datos más altas (DB4 a DB7).
8	R/W, RS, Habilitar y las 8 líneas de datos. Todas las líneas de datos deben estar conectadas al mismo puerto de E/S, en orden secuencial. Por ejemplo, DB0 a PORTB.0, DB1 a PORTB.1 y así sucesivamente, con DB7 yendo a PORTB.7.

Usando el modo de 0 bits:

Para usar el modo de conexión de 0 bits, se debe proporcionar una subrutina para escribir un byte en la pantalla LCD. Opcionalmente, también se puede dar otra subrutina para leer un byte de la pantalla LCD. Si no hay forma de leer desde la pantalla LCD, entonces se debe establecer la constante LCD_NO_RW.

En el modo de 0 bits, la constante LCD_RS se establecerá automáticamente en una variable de bit de repuesto. Los comandos LCD de nivel superior (como Imprimir y Localizar) lo configurarán, y el código responsable de escribir en la pantalla LCD debe leerlo y luego configurar el pin RS en la pantalla LCD de manera adecuada.

Este código es un ejemplo de cómo usar el modo de 0 bits. Envía mensajes a otro microcontrolador, que ha sido programado para leer los mensajes y alternar los pines de una pantalla LCD de manera adecuada:

```
#define LCD_IO 0

#define LCDWriteByte MySendToLCD

#define LCD_NO_RW


Sub MySendToLCD (en MyLCDByte)
    'Usa I2C
    'Envía un byte de dirección (128)
    'Luego un byte de control, donde el bit 4 es el estado del pin RS
    Luego, un byte de datos, que se envía a los pines de datos de la pantalla LCD.
```

Sample Version - Do Not Copy

Machine Translated by Google

```
    Byte de control = 0
    Si LCD_RS = Encendido Entonces ControlByte.4 = Encendido
    I2CInicio
    I2C Enviar 128
    I2CSend ControlByte
    I2CSend MyLCDByte
    I2CSparar
    'Necesito dar tiempo para que el receptor del mensaje actualice la pantalla LCD
    Espera 5ms

Finalizar sub
```

Si se tuviera que leer la pantalla LCD, LCDReadByte se establecería con el nombre de una función que lee la pantalla LCD y devuelve el byte de datos de la pantalla LCD.

Constantes relevantes:

(Nota: esta sección no se aplica en el modo de 0 bits)

Estas constantes se utilizan para controlar la configuración de las rutinas de pantalla de cristal líquido incluidas con GCBASIC. Para configurarlos, coloque una línea en el archivo del programa principal que use #define para asignar un valor a la constante particular.

Algunas constantes son necesarias para los modos de 4 y 8 bits, algunas son necesarias para el modo de 4 bits y otras son necesarias para el modo de 8 bits. Cuando se utiliza el modo de 2 bits, solo se deben configurar tres constantes; todas las demás se pueden ignorar. Verifique la columna "Modos" para determinar si debe establecer una constante.

Sample Version - Do Not Copy

Machine Translated by Google

Nombre constante	Control S	Valor por defecto	Modos
LCD_IO	El modo de E/S. Pueden ser 2, 4 u 8.	8	2, 4, 8
LCD_DB	El pin de datos utilizado en el modo de 2 bits.	N/A - Debe establecerse	2
LCD_CB	El pin de reloj utilizado en el modo de 2 bits.	N/A - Debe establecerse	2
LCD_RS	Especifica el pin de salida que está conectado a la selección de registro en la pantalla LCD.	N/A - Debe establecerse	4, 8
LCD_RW	Especifica el pin de salida que está conectado a lectura/escritura en la pantalla LCD. El pin R/W se puede desactivar*.	N/A: debe configurarse (a menos que R/W esté deshabilitado)	4, 8
LCD_Habilitar	Especifica el pin de salida que está conectado a lectura/escritura en la pantalla LCD.	N/A - Debe establecerse	4, 8
LCD_DATA_PORT	Puerto de salida utilizado para interactuar con el bus de datos LCD	N/A - Debe establecerse	solo 8
LCD_DB4	Pin de salida utilizado para interactuar con el bit 4 del bus de datos LCD	N/A - Debe establecerse	4 solo
LCD_DB5	Pin de salida utilizado para interactuar con el bit 5 del bus de datos LCD	N/A - Debe establecerse	4 solo
LCD_DB6	Pin de salida utilizado para interactuar con el bit 6 del bus de datos LCD	N/A - Debe establecerse	4 solo
LCD_DB7	Pin de salida utilizado para interactuar con el bit 7 del bus de datos LCD	N/A - Debe establecerse	4 solo

Sample Version - Do Not Copy

Machine Translated by Google

*El pin R/W se puede desactivar configurando la constante LCD_NO_RW. Si se hace esto, no es necesario que el R/W esté conectado al chip, y no es necesario configurar la constante LCD_RW. ¡Asegúrese de que la línea R/W en la pantalla LCD esté conectada a tierra si no se usa!

4.17 Resumen de tonos

Introducción:

Estas rutinas generan tonos de una determinada frecuencia y duración.

Nota: Si se requiere una frecuencia exacta o se necesita un programa más pequeño, no se deben usar estas rutinas. En su lugar, debe utilizar un código como este:

```
Repetir conteo
    PulseOut SoundOut, punto
    Período de espera nosotros
Finalizar repetición
```

Establezca el conteo y el período en los valores apropiados:

•período a 1000000 / frecuencia deseada / 2

•Cuenta hasta la duración/período deseado.

Constantes relevantes:

Estas constantes se utilizan para controlar la configuración de las rutinas de generación de tonos. Para configurarlos, coloque una línea en el archivo del programa principal que use #define para asignar un valor a la constante particular.

Sample Version - Do Not Copy

Machine Translated by Google

Nombre constante	Control S	Valor por defecto
Apagado	El pin de salida para producir sonido en	N/A - Debe establecerse

4.18 Resumen de SPI

Disponibilidad de comandos:
Disponible en microcontroladores PIC con módulo Synchronous Serial Port (SSP).
Estos comandos controlan la comunicación SPI.

Sintaxis:
Modo SPIMode

SPIMode establece el modo del módulo SPI dentro del chip PIC. Estos son los posibles Modos SPI:

Nombre del modo	Descripción
maestrolento	Modo maestro, el reloj SPI es 1/64 de la frecuencia del PIC.
Maestro	Modo maestro, el reloj SPI es 1/16 de la frecuencia del PIC.
MasterFast	Modo maestro, el reloj SPI es 1/4 de la frecuencia del PIC.
Esclavo	Modo esclavo
Esclavo	Modo esclavo, con el pin de selección de esclavo habilitado.

Sintaxis:
Transferencia SPI tx, rx

Este comando envía y recibe simultáneamente un byte de datos utilizando el protocolo SPI. Se comporta de manera diferente dependiendo de si el PIC se ha configurado para actuar como maestro o esclavo.

Cuando se opera como maestro, el comando SPITransfer iniciará una transferencia.
Los datos en tx se enviarán al esclavo, mientras que el byte que se almacena en el búfer en el esclavo

Sample Version - Do Not Copy

Machine Translated by Google

se leerá en rx.

En modo esclavo, el comando SPITransfer pausará el programa hasta que el maestro inicie una transferencia. En este punto, enviará los datos en tx mientras lee la transmisión del maestro en la variable rx.

4.19 Descripción general de GLCD

Los comandos GLCD se utilizan para controlar una pantalla de cristal líquido gráfica basada en el chip controlador KS0108. Suelen ser pantallas de 128x64 píxeles. Pueden dibujar elementos gráficos habilitando o deshabilitando píxeles.

GCB hace que esto sea más fácil de controlar con los siguientes comandos.

Configuración: debe incluir el archivo glcd.h en la parte superior de su programa. El archivo debe estar entre paréntesis.

#incluir <GLCD.h>

Define:
Hay varias conexiones que deben definirse para usar los comandos GLCD con una pantalla KS0108. El pin de E/S es el pin del microcontrolador PIC que está conectado a ese pin específico en la pantalla LCD gráfica.

```
#define pin de E/S GLCD_RW              'Conexión de pin de lectura/
escritura #define GLCD_RESET I/O pin 'Restablecer conexión de pin
#define GLCD_CS1 I/O pin                'Conexión de pin CS1
#define GLCD_CS2 pin de E/S             'Conexión de pin CS2
#define pin de E/S GLCD_RS              'Conexión de pin RS
#define GLCD_ENABLE pin de E/S 'Habilita conexión de pin #define
GLCD_DB0 pin de E/S                     'Conexión de pin de datos
0 #define GLCD_DB1 pin de E/S           'Conexión de pin de datos
1 #define GLCD_DB2 pin de E/S           'Conexión del pin de datos
2 #define GLCD_DB3 pin de E/S           'Conexión del pin de datos
3 #define GLCD_DB4 pin de E/S           'Conexión del pin de datos
4 #define GLCD_DB5 pin de E/S           'Conexión del pin de datos
5 #define GLCD_DB6 pin de E/S           'Conexión del pin de datos
6 #define GLCD_DB7 pin de E/S           'Conexión de pin de datos 7
```

Special Version - Do Not Copy

Machine Translated by Google

Comandos

InitGLCD 'Inicializar pines de puerto

GLCDCLS 'Borrar pantalla

GLCDPrint (en PrintLocX, en PrintLocY, PrintData como cadena)

GLCDDrawChar (en CharLocX, en CharLocY, en CharCode)

Caja (En LíneaX1, En LíneaY1, En LíneaX2, En LíneaY2, Opcional En LíneaColor = 1)

CajaRellenada(En LíneaX1, En LíneaY1, En LíneaX2, En LíneaY2, Opcional En LíneaColor = 1)

Línea (En LíneaX1, En LíneaY1, En LíneaX2, En LíneaY2, Opcional En LíneaColor = 1)

PSet(En GLCDX, En GLCDY, En GLCDState)

GLCDWriteByte (en LCDByte)

GLCDReadByte

Sample Version - Do Not Copy

Machine Translated by Google

Sample Version - Do Not Copy

Machine Translated by Google

5.0 Comandos

Caja

Sintaxis:
Cuadro (LineX1, LineY1, LineX2, LineY2, LineColour opcional = 1)

Explicación:
Dibuja un cuadro en una pantalla LCD gráfica desde la esquina superior de la ubicación X1, Y1 hasta la ubicación X2, Y2.

Véase también el comando CuadroRellenado

ClearTimer

Sintaxis:

ClearTimer TemporizadorNo

Explicación:

ClearTimer se utiliza para borrar el temporizador especificado.

Ejemplo:

Consulte el artículo InitTimer1 para ver un ejemplo.

Sample Version - Do Not Copy

Machine Translated by Google

CLS

Sintaxis:

CLS

Explicación:

El comando CLS borra el contenido de la pantalla LCD y devuelve el cursor a la esquina superior izquierda de la pantalla.

Ejemplo:

```
'Un programa Flashing Hello World para GCBASIC

'Configuración general de hardware
#chip 16F877A, 20

'Configuración de conexión LCD
#definir LCD_IO 8
#define LCD_DATA_PORT PUERTO
#define LCD_RS PUERTO.0
#define LCD_RW PUERTO.1
#define LCD_Habilitar PUERTO.2

'Rutina principal
Hacer
Imprimir "Hola mundo"
```

Sample Version - Do Not Copy

Machine Translated by Google

```
Espera 1 segundo

CLS

Espera 1 segundo

Bucle
```

Oscuro

Dim tiene dos usos, los cuales implican grandes variables. Se puede utilizar para definir matrices y declarar variables de más de 1 byte.

Sintaxis:

Para Variables > 1 byte:

Dim variable [, variable2[, variable3]] [Como tipo] [Alias othervar [,othervar2]] [En la ubicación]

Para arreglos:

Matriz atenuada (tamaño) [En la ubicación]

Explicación:

El comando Dim variable se utiliza para informar a GCBASIC de variables que tienen más de 1 byte, o para crear nombres alternativos para otras variables.

El comando Dim array también configura variables de matriz. Al usarlo para esto, el tamaño debe ser un valor constante hasta 80.

tipo especifica el tipo de variable que se va a crear. Diferentes tipos de variables pueden contener valores en diferentes rangos y usar diferentes cantidades de RAM. Consulte el artículo Variables para obtener más información.

Sample Version Do Not Co

Machine Translated by Google

Cuando se incluyen múltiples variables en una línea, GCBASIC las configurará todas al tipo que se especifica al final de la línea. Si no se especifica ningún tipo, GCBASIC convertirá la variable en un byte.

Alias crea una variable utilizando la misma ubicación de memoria que una o más variables. Se usa principalmente internamente en GCBASIC para tratar las variables del sistema como una palabra. Por ejemplo, este comando se utiliza para crear una variable de palabra, formada por las dos ubicaciones de memoria utilizadas para almacenar el resultado de una conversión A/D:

Se puede colocar una variable en una ubicación específica en la memoria de datos del chip usando la opción En. Se usará la ubicación ya sea una ubicación válida o no, pero se generará una advertencia si GCBASIC ya ha asignado la memoria, o si la memoria no parece ser válida. Esto se puede usar para periféricos que tienen búferes de varios bytes en la RAM.

Dim ADResult como palabra Alias ADRESH, ADRESL

Ejemplo:

```
'Este programa configurará una matriz y una variable de palabra


lista de datos tenue (10)

tenue Lectura como palabra


Lectura = 21978

Lista de datos (1) = 15
```

Sample Version - Do Not Copy

Machine Translated by Google

Dirección

Sintaxis:

Dir puerto.bit {En | Out} (Forma Individual)

Puerto dir {En | Fuera | DirectionByte} (formulario de puerto completo)

Disponibilidad de comandos:

Disponible en todos los microcontroladores. Sin embargo, algunos microcontroladores PIC de gama baja (chips 10F, 12F5x y 16F5x) solo aceptarán el formulario de puerto completo.

Explicación:

El comando Dir se usa para establecer la dirección de los puertos del chip del microcontrolador. El formulario individual establece la dirección de un pin a la vez, mientras que el formulario de puerto completo establecerá todos los bits en un puerto.

En el formulario individual, especifique el puerto y el bit (es decir, PORTB.4), luego la dirección, que es In o Out.

La forma completa del puerto es similar a la instrucción TRIS que ofrecen algunos chips PIC. Para usarlo, proporcione el nombre del puerto (es decir, PORTA), y luego se debe escribir un byte en la variable TRIS. Esta forma de comando es para aquellos que están familiarizados con la arquitectura interna del chip PIC.

ADVERTENCIA: Los PIC usan 0 para salir y 1 para entrar. Cuando se usan IN y OUT, no hay problemas de compatibilidad.

Sample Version - Do Not Copy

Machine Translated by Google

Ejemplo:

```
'Este programa pone los bits 0 y 1 de PORTA en entrada y el resto en salida.
'También configura todo PORTB para salida, excepto B1.
'Se utiliza forma individual para PORTA:
DIR PORTA.0 EN
DIR PORTA.1 EN
DIR PORTA.2 SALIDA
DIR PORTA.3 SALIDA
DIR PORTA.4 SALIDA
DIR PORTA.5 SALIDA
DIR PORTA.6 SALIDA
DIR PORTA.7 SALIDA
'Todo el formulario de puerto utilizado para B:
DIR PUERTO b'00000010'
'Todo el formulario de puerto utilizado para C:
ENTRADA PUERTO DIR
```

Sample Version - Do Not Copy

DisplayChar

Sintaxis:

DisplayChar (pantalla, carácter)

Explicación:

Machine Translated by Google

Este comando mostrará el carácter ASCII dado en una pantalla LED de siete segmentos. display es el número de la pantalla a usar, y character es el carácter ASCII a mostrar.

Ejemplo:

```
'Este programa mostrará "Hola" en una pantalla LED

'La pantalla debe estar conectada a PORTB



#chip 16F877A, 20

#define DisplayPortA PUERTOB



Mensaje tenue como cadena

Mensaje = "Hola"

Para Contador = 1 a 6

    DisplayChar 1, mensaje (contador)

    Espere 250ms

Próximo
```

Mostrar valor

Sintaxis:

Visualización de DisplayValue, datos

Explicación:

Sample Version - Do Not Copy

Machine Translated by Google

Este comando mostrará el valor dado en una pantalla LED de siete segmentos. display es el número de la pantalla a usar, y data es el valor entre 0 y 9 para mostrar

Ejemplo:

```
'Este programa contará de 0 a 99 en dos pantallas LED

#chip 16F819, 8

#config osc = int



#define DISP_SEG_A PORTB.0

#define DISP_SEG_B PORTB.1

#define DISP_SEG_C PORTB.2

#define DISP_SEG_D PORTB.3

#define DISP_SEG_E PORTB.4

#define DISP_SEG_F PORTB.5

#define DISP_SEG_G PORTB.6



#define DISP_SEL_1 PORTA.0

#define DISP_SEL_2 PORTA.1


Hacer

Para contador = 0 a 99



 'Obtener los 2 dígitos

Número = Contador

número1 = 0
```

Sample Version - Do Not Copy

Machine Translated by Google

```
Si Número >= 10 Entonces
Núm1 = Número / 10
 'SysCalcTempX almacena el resto después de la división
Número = SysCalcTempX
Terminara si
Núm2 = Número


 'Mostrar los dígitos
 'Cada DisplayValue borrará al otro (multiplexación)
 'Así que deben ser llamados con la frecuencia suficiente para que el parpadeo
 'No puede ser visto.
Repetir 500
Valor de pantalla 1, número 1
Espera 1 ms
Valor de visualización 2, Num2
Espera 1 ms
Finalizar repetición
Próximo
Bucle
```

Sample Version - Do Not Copy

Hacer	

Sintaxis:

Hacer [{Mientras | Hasta} condición]

Machine Translated by Google

...

código de programa

...

Bucle [{Mientras | Hasta} condición]

Explicación:

El comando Do hará que el código entre Do y Loop se ejecute repetidamente mientras la condición sea verdadera o hasta que la condición sea verdadera, dependiendo de si se ha especificado Mientras o Hasta.

Tenga en cuenta que el tiempo o hasta y la condición solo se pueden especificar una vez, o no se pueden especificar en absoluto. Si no se especifican, el código se repetirá sin cesar.

Ejemplo:

```
'Este código parpadeará una luz hasta que se presione el botón

#chip 12F629, 4

#config osc = int



#define BOTÓN GPIO.3

#define LUZ GPIO.5



Dir BOTÓN Entrada

Dir LUZ Fuera



Hacer hasta BOTÓN = 1
```

Sample Version - Do Not Copy

Machine Translated by Google

```
PulseOut LUZ, 1 s

Espera 1 s

Bucle
```

Fin

Sintaxis:

Fin

Explicación:

Cuando se usa el comando Finalizar, el programa dejará de ejecutarse inmediatamente. Hay muy pocos casos en los que se necesita este comando; por lo general, el programa debe ser un bucle sin fin.

Ejemplo:

```
'Este programa encenderá la luz roja, pero no la luz verde

Activar ROJO

Fin

Activar VERDE
```

EPRead

Sintaxis:

EPLeer ubicación, tienda

Sample Version - Do Not Co

Machine Translated by Google

Disponibilidad de comandos:

Disponible en todos los chips PIC con memoria de datos EEPROM.

Explicación:

EPRead se utiliza para leer información del almacenamiento de datos EEPROM con el que están equipados muchos chips de microcontroladores. La ubicación representa la ubicación desde la que se leen los datos y varía de un chip a otro. store es la variable en la que almacenar los datos después de haberlos leído de la EEPROM.

Ejemplo:

```
'Programa para encender y apagar una luz

'Recordaré el último estado



#chip diminuto2313, 1

#define Botón PORTB.0

#define Luz PORTB.1



Botón de entrada de directorio

Dir luz apagada



'Cargar estado guardado

EPRead 0, estado de luz



Si LightStatus = 0 Entonces

Apagar luz
```

Sample Version - Do Not Copy

Machine Translated by Google

```
Demás
Enciende la luz
Terminara si

Hacer
'Esperar a que se presione el botón
Botón Esperar mientras = Encendido
Botón Esperar mientras = Apagado

'Alternar valor, registro
Estado de luz = ! Estado de luz
EPWrite 0, estado de luz

'Actualizar luz
Si LightStatus = 0 Entonces
Apagar luz
Demás
Enciende la luz
Terminara si
Bucle
```

Sample Version - Do Not Copy

EPWrite

Sintaxis:

EPEscribir ubicación, datos

Machine Translated by Google

Disponibilidad de comandos:

Disponible en todos los chips PIC con memoria de datos EEPROM.

Explicación:

EPWrite se utiliza para escribir información en el almacenamiento de datos de la EEPROM, de modo que un programador en la PC pueda acceder a ella más tarde o mediante el comando EPRead. La ubicación representa la ubicación desde la que se leen los datos y varía de un chip a otro. Los datos son los datos que se escribirán en la EEPROM y pueden ser un valor o una variable.

Ejemplo:

```
#chip 16F819, 8

#config osc = int


 'Establecer la dirección del pin de entrada

Dir PORTA.0 Entrada

 'Bucle para tomar lecturas hasta que la EEPROM esté llena

Para dirección actual = 0 a 255

 'Haz una lectura y regístrala

EPWrite dirección actual, ReadAD(AN0)


 'Espere 10 minutos antes de obtener otra lectura

Espera 10 minutos


Próximo
```

Sample Version - Do Not Copy

Machine Translated by Google

Salir de Sub

Sintaxis:

Salir de Sub

Explicación:

Este comando hará que el programa salga de la subrutina en la que se encuentra actualmente, como lo haría si llegara al final de la subrutina.

Ejemplo:

```
#chip diminuto13, 1

#define SENSOR PORTB.0
#define ZUMBADOR PUERTO B.1
#define PUERTO DE LUZ.2
Dir SENSOR Entrada
Dir ZUMBADOR Salida
Dir LUZ Fuera

Hacer
Ladrón
Bucle

'Subrutina de alarma antirrobo
```

Sample Version - Do Not Copy

Machine Translated by Google

```
sub ladrón

Si SENSOR = 0 Entonces

Desactivar ZUMBADOR

Desactivar LUZ

Salir de Sub

Terminara si

Activar ZUMBADOR

Encender LUZ

Finalizar sub
```

cajallena

Sintaxis:
CajaRellenada(LíneaX1,LíneaY1, LíneaX2, LíneaY2, ColorDeLínea Opcional = 1)

Explicación: dibuja
un cuadro lleno en una pantalla LCD gráfica desde la esquina superior de la ubicación X1, Y1 hasta la ubicación X2, Y2.

Véase también Comando de cuadro

Para

Sintaxis:

```
Para contador = inicio Para finalizar [Incremento de paso]
...
código de programa
...

Próximo
```

Sample Version - Do Not Copy

Machine Translated by Google

Explicación:

El comando For es ideal para situaciones en las que se debe ejecutar un fragmento de código un número determinado de veces y en las que es necesario realizar un seguimiento de cuántas veces se ha ejecutado el código. Cuando el comando For se ejecuta por primera vez, el contador se pone en marcha. Luego, cada vez que el programa se repite, se agrega un incremento al contador, hasta que el contador es igual al final. Luego, el programa continúa más allá del Siguiente.

El paso y el incremento son opcionales. Si no se especifica Step, GCBASIC incrementará el contador en 1 cada vez que se ejecute el código.

Ejemplo:

```
'Este código hará parpadear una luz verde 6 veces.

#chip 16F88, 8

#config Osc = Int

#define PUERTO LED.0

Dirección LED de salida

Para LoopCounter = 1 a 6

Led PulseOut, 1 s

Espera 1 s

Próximo
```

Sample Version - Do Not Copy

Machine Translated by Google

Conseguir

Sintaxis:

var = Obtener (Línea, Columna)

Disponibilidad de comandos:

Disponible en todos los microcontroladores, excepto si la pantalla LCD está conectada mediante el modo de 2 bits.

Explicación:

La función Obtener lee el código de caracteres ASCII en una ubicación determinada en la pantalla LCD.

GLCDCLS

Sintaxis:

GLCDCLS

Explicación:

Borra la pantalla de un LCD gráfico.

Sample Version - Do Not Copy

Machine Translated by Google

GLCDImprimir

Sintaxis:
GLCDPrint(PrintLocX, PrintLocY, PrintData)

PrintLocX es la ubicación de coordenadas X para los datos

PrintLocY es la ubicación de la coordenada Y para los datos

PrintData es una variable String o String de los datos a mostrar

Explicación:
Imprime caracteres de cadena en una ubicación específica en la pantalla GLCD.
En una pantalla GLCD de 128x64,
X suele ser de 0 a 128 Y suele ser
de 0 a 64

GLCDDrawChar

Sintaxis:
GLCDDrawChar(CharLocX, CharLocY, CharCode)

CharLocX es la ubicación de la coordenada X para el personaje.

CharLocY es la ubicación de la coordenada Y para el carácter

CharCode es el número ASCII del carácter a mostrar. Puede ser hexadecimal decimal o binario.

Explicación:
muestra un carácter ASCII en una ubicación X e Y especificada.
En un LCD gráfico de 128x64 X
= 1 a 128 Y = 1 a 64

Sample Version - Do Not Copy

Machine Translated by Google

GLCDWriteByte

Sintaxis:
GLCDWriteByte (LCDByte)

Explicación:
Escribe un byte de datos en la memoria LCD gráfica

GLCDReadByte

Sintaxis:
GLCDWriteByte

Explicación:
Lee un byte de datos de la memoria LCD gráfica

Gosub

Sintaxis:

etiqueta gosub

Explicación:

El comando Gosub se usa para saltar a una etiqueta como una subrutina, de manera similar a Goto. La diferencia es que Return se puede usar para volver a la línea de código después de Goto.

Sample Version - Do Not Copy

Machine Translated by Google

Nota: Gosub no debe usarse si se puede evitar. No es necesario llamar a una subrutina que ha sido definida usando Sub, solo escriba el nombre de la subrutina.

Ejemplo:

```
'Este programa hará parpadear un LED en el bit 0 del puerto y reproducirá un pitido en
'porta bit 4. hasta que se apague el robot.

#chip 16F628A, 4' Cambia esto para que se adapte a tu circuito

#define SOUNDOUT PORTA.4
#define PUERTO DE LUZ.0
Dir LUZ Fuera

Hacer
'Linterna
PulseOut LUZ, 1 s
Espera 1 s
'Bip
Gosub Play Beep
Bucle
Reproducir pitido:
Tono 200, 10
Tono 100, 10
Devolver
```

Sample Version - Do Not Copy

Machine Translated by Google

Ir a

Sintaxis:

Ir a etiqueta

Explicación:

El comando Ir a hará que el robot salte a la línea especificada y continúe ejecutando el programa desde allí. El comando Ir a es principalmente útil para salir de los bucles; si necesita crear un bucle infinito, use el comando Do en su lugar.

Tenga cuidado con cómo usa Goto. Si se usa demasiado, puede hacer que los programas sean muy difíciles de leer.

Para definir una etiqueta, coloque el nombre de la etiqueta solo en una línea, con solo dos puntos (:) después.

Ejemplo:

```
'Este programa hará parpadear la luz hasta que se presione el botón
'apagado. Observe la etiqueta denominada SWITCH_OFF.

#chip 16F628A, 4' Cambia esta linea para que se adapte a tu circuito

#define BOTÓN PORTB.0
#define PUERTO DE LUZ.1
Dir BOTÓN Entrada
Dir BOTÓN Fuera
```

Sample Version - Do Not Copy

Machine Translated by Google

```
Hacer

PulseOut LUZ, 500 ms

Si BOTÓN = 1 Entonces Ir a SWITCH_OFF

Espere 500ms

Si BOTÓN = 1 Entonces Ir a SWITCH_OFF

Bucle


APAGAR:

Desactivar LUZ

'El chip entrará en modo de bajo consumo cuando finalice el programa
```

HPWM

Sintaxis:

canal HPWM, frecuencia, ciclo de trabajo

Disponibilidad de comandos:

Solo disponible en microcontroladores PIC con módulo Capture/Compare/PWM (CCP).

Explicación:

Este comando configura el módulo PWM de hardware del chip PIC para generar una forma de onda PWM de la frecuencia y el ciclo de trabajo dados. Una vez que se llama a este comando, se emitirá el PWM hasta que se llame a PWMOff. Si solo necesita una frecuencia y un ciclo de trabajo en particular, debe usar PWMOn y las constantes PWM_Freq y PWM_Duty en su lugar.

Sample Version - Do Not Copy

Machine Translated by Google

el canal es 1 o 2, y corresponde a los pines CCP1 y CCP2 respectivamente. En chips con un solo puerto CCP, siempre se usa el pin CCP o CCP1 y se ignora el canal. (Debe establecerse en 1 de todos modos para permitir futuras actualizaciones a chips PIC más potentes).

frecuencia establece la frecuencia de la salida PWM. Se mide en KHz. El valor máximo permitido es 255 KHz. El valor mínimo varía según la velocidad del reloj. 1 KHz es el mínimo en chips de 16 MHz o menos y 2 Khz es el mínimo posible en chips de 20 MHz. En situaciones que no requieren una frecuencia PWM específica, la frecuencia PWM debe ser igual a aproximadamente 1 quinientosima parte de la velocidad de reloj del PIC (es decir, 40 Khz en un chip de 20 MHz, 16 KHz en un chip de 8 MHz). Esto proporciona la mejor resolución de ciclo de trabajo posible.

ciclo de trabajo especifica el ciclo de trabajo deseado de la señal PWM y varía de 0 a 255, donde 255 es el ciclo de trabajo del 100 %.

Ejemplo:

```
'Este programa alterará el brillo de un LED usando
'PWM de hardware.

'Seleccione modelo de chip y velocidad
#chip 16F877A, 20

'Establecer el pin CCP1 en modo de salida
Salida PUERTO DIR.2

'Código principal
```

Sample Version - Do Not Copy

Machine Translated by Google

```
hacer
 'Sube el brillo durante 2,5 segundos
Para brillante = 1 a 255
HPWM 1, 40, brillante
espera 10ms
próximo
 'Reduce el brillo durante 2,5 segundos
Para brillante = 255 a 1
HPWM 1, 40, brillante
espera 10ms
próximo
bucle
```

HSerImprimir

HSerPrint se utiliza para enviar un valor a través de la conexión en serie. El valor puede ser una cadena, una palabra o un byte: SerPrint es muy similar a Print. El canal es la conexión en serie para enviar datos.

Sintaxis:

valor HSerPrint

Disponibilidad de comandos:

Disponible en todos los microcontroladores con módulo USART o UART.

Nota:
HSerPrint no enviará ningún carácter de nueva línea. Si el chip está enviando a una terminal, estos comandos deben seguir cada SerPrint:

Sample Version - Do Not Copy

Machine Translated by Google

HSerPrint 13
HSerPrint 10

O

HSerImprimirCRLF

Ejemplo:

Este programa mostrará cualquier valor recibido a través de la conexión serial. Si se recibe "pot", se enviará el valor del sensor analógico.

'Nota: Esto ha sido adaptado del ejemplo de SerPrint.

'Configuración de chips

#chip 18F2525, 8

#config Osc = Int

'Configuración de pantalla LCD

#definir LCD_IO 4

#define LCD_RS PORTC.7

#define LCD_RW PUERTO.6

#define LCD_Habilitar PUERTO.5

#define LCD_DB4 PORTC.4

#define LCD_DB5 PORTC.3

#define LCD_DB6 PUERTO.2

#define LCD_DB7 PUERTO.1

'Configuración USART

Sample Version - Do Not Copy

Machine Translated by Google

```
#define USART_BAUD_RATE 9600


'Potenciómetro
#define POT_PORT PORTA.0
#define POT_AN AN0


'Establecer direcciones de pines
Dir SerOutPort Salida
Dir SerInPort In
Dir POT_PORT Entrada


'Crear variables de búfer para almacenar mensajes recibidos
Búfer atenuado como cadena
Dim OldBuffer como cadena
Tamaño de búfer = 0


'Mostrar mensajes de prueba
Imprimir "Probador en serie"
Espera 1 s
HSerPrint "Prueba GCBASIC RS232"
HSerSend 13
HSerSend 10
Espera 1 s


'Bucle principal
```

Sample Version - Do Not Copy

Machine Translated by Google

```
Hacer
 'Obtener un byte de la terminal
HSerReceive Temp

 'Si se presionó la tecla Intro, tratar con el contenido del búfer
Si temperatura = 13 Entonces
Búfer(0) = Tamaño de Búfer

 'Intenta ejecutar comandos en el búfer
Si no es ExecCommand (búfer), entonces
 'Mostrar mensaje en la línea inferior, último mensaje en la parte superior.
CLS
Imprimir búfer antiguo
Localizar 1, 0
Búfer de impresión
 'Guardar el mensaje para la próxima vez
OldBuffer = Búfer
Terminara si

Tamaño de búfer = 0
Terminara si
 'Código de retroceso, borra el último carácter en el búfer
Si temperatura = 8 Entonces
Si BufferSize > 0 Entonces BufferSize -= 1
Terminara si
```

Sample Version - Do Not Copy

Machine Translated by Google

```
'Recibido código ASCII entre 32 y 127, agregar al búfer
Si Temp >= 32 y Temp <= 127 Entonces
Tamaño del búfer += 1
Búfer (Tamaño del búfer) = Temperatura
Terminara si
Bucle


'Toma una lectura del sensor y la envía a la terminal
Lectura del sensor de envío secundario
HSerPrint "Lectura del sensor:"
HSerPrint ReadAD10(POT_AN)
HSerSend 13
HSerSend 10
Finalizar sub
'Comprobará el búfer para un comando
'Si se encuentra el comando, ejecútelo y devuelva verdadero
'Si no, devuelve falso
Función ExecCommand (CmdIn como cadena)
ComandoEjecutivo = Falso
'Si el comando recibido es "pot", muestra el valor del potenciómetro
Si CmdIn = "olla" Entonces
EnviarSensorLectura
ComandoEjecutivo = Verdadero
Terminara si
función final
```

Sample Version - Do Not Copy

Machine Translated by Google

HSerRecibir

Sintaxis:

Utilizado como subrutina:

Salida HSerReceive

Usado como función:

salida = HSerReceive

Disponibilidad de comandos:

Disponible en todos los microcontroladores con módulo USART o UART.

Explicación:

Este comando leerá un byte del módulo de hardware RS232. Se puede utilizar como subrutina o como función. Si se usa como una subrutina, se debe proporcionar una variable para almacenar el valor recibido. Si se usa como una función, devolverá el valor recibido.

Ejemplo:

```
'Este programa leerá un valor del USART y lo mostrará en PORTB.

#chip 16F877A, 20

'Configuración USART
```

Sample Version - Do Not Copy

Machine Translated by Google

```
#define USART_BAUD_RATE 9600


'Establecer PORTB en entrada
Dir PUERTO Salida
'Establece el pin de recepción de USART en la entrada
Dir PUERTO.7 Entrada


'Bucle principal
Hacer
'Obtener y mostrar el valor
'Si no hay datos nuevos, HSerReceive devolverá el valor antiguo.
HSerReceive PORTB
'También podría escribir:
'PORTB = HSerReceive
Bucle
```

Sample Version - Do Not Copy

HSerEnviar

Sintaxis:

HSerEnviar datos

Disponibilidad de comandos:

Disponible en todos los microcontroladores con módulo USART o UART.

Machine Translated by Google

Explicación:

Este comando enviará un byte dado por datos utilizando el módulo de hardware RS232.

Ejemplo:

```
'Este programa enviará el estado de PORTB a través del hardware
'módulo serie.

#chip 16F877A, 20

'Configuración USART
#define USART_BAUD_RATE 9600

'Establecer PORTB en entrada
Dir PORTB In
'Configura el pin de transmisión USART a la salida
Dir PUERTO.6 Salida

'Bucle principal
Hacer
'Enviar valor PORTB a través de USART
HSerSend PUERTOB
'Pequeño retraso para que el receptor procese el mensaje
Espera 10ms
Bucle
```

Sample Version - Do Not Copy

Machine Translated by Google

I2CRecibir

Sintaxis:

I2CRecibir datos

I2CRecibir datos, acuse de recibo

Disponibilidad de comandos:

Disponible en todos los microcontroladores excepto en los PIC de instrucción de 12 bits (10F, 12F5xx, microprocesadores 16F5xx)

Explicación:

El comando I2CReceive enviará datos a través de la conexión I2C. Si ack es VERDADERO, o no se da ningún valor para ack, entonces I2CReceive enviará un ack.

Si está en modo maestro, I2CReceive leerá los datos inmediatamente. Si está en modo esclavo, I2CReceive esperará a que el maestro envíe los datos antes de leer.

Ejemplo:

```
'Este programa lee un registro I2C y enciende un LED si está por encima de 100.
'Se leerá desde la dirección 83, registro 1.

'Configuración de chips
#chip 18F2620, 8

'Configuración I2C
```

Sample Version - Do Not Copy

Machine Translated by Google

```
#define I2C_MODE Maestro

#define I2C_DATA PORTC.0

#define I2C_PUERTO DE RELOJ.1


'Configuraciones misceláneas

#define PUERTO LED.0


'Bucle principal

Hacer

'Enviar inicio

I2CInicio


'Solicitar valor

I2C Enviar 83

I2CSenviar 1


'Leer valor

I2CRecibir ValorEn


'Enviar detener

I2CSparar


'Enciende el LED si se recibe un valor > 100

Establecer LED apagado

Si ValueIn > 100, entonces encienda el LED
```

Sample Version - Do Not Copy

Machine Translated by Google

'Demora

Espere 20 ms

Bucle

I2CSend

Sintaxis:

I2CEnviar datos

I2CEnviar datos, acuse de recibo

Disponibilidad de comandos:

Disponible en todos los microcontroladores excepto en los PIC de instrucción de 12 bits (10F, 12F5xx, microprocesadores 16F5xx)

Explicación:

El comando I2CSend enviará datos a través de la conexión I2C. Si ack es VERDADERO, o no se da ningún valor para ack, entonces I2CSend esperará un bit de reconocimiento del receptor antes de continuar.

Si está en modo maestro, I2CSend enviará los datos inmediatamente. Si está en modo esclavo, I2CSend esperará a que el maestro solicite los datos antes de enviar.

Ejemplo:

'Este programa actuará como un convertidor analógico a digital I2C

'Cuando se solicitan datos de la dirección 83, registra del 0 al

Sample Version - Do Not Copy

Machine Translated by Google

'3, devolverá el valor de AN0 a AN3.

modelo de chip

#chip 16F88, 8

'Configuración I2C

#define I2C_MODE Esclavo

#define I2C_CLOCK PORTB.0

#define I2C_DATA PORTB.1

'Bucle principal

Hacer

'Esperar condición de inicio

I2CInicio

'Obtener dirección

Dirección de recepción I2C

Si Dirección = 83 Entonces

'Si la dirección era la dirección de este dispositivo, responda

Registro de recepción I2C

OutValue = ReadAD(Registrar)

I2CEnviarValor

Terminara si

Sample Version - Do Not Copy

Machine Translated by Google

I2CSparar

Espera 5ms

Bucle

I2CInicio

Sintaxis:

I2CInicio

Disponibilidad de comandos:

Disponible en todos los microcontroladores excepto en los PIC de instrucción de 12 bits (10F, 12F5xx, microprocesadores 16F5xx)

Explicación:

Si las rutinas I2C están operando en modo Maestro, este comando enviará una condición de inicio. Si las rutinas están en modo Esclavo, pausará el programa hasta que el maestro envíe una condición de inicio. Debe colocarse al comienzo de cada transmisión I2C.

Si el manejo de interrupciones está habilitado, este comando lo deshabilitará.

Ejemplo:

Consulte I2CSend e I2CReceive para ver un ejemplo.

Sample Version - Do Not Copy

Machine Translated by Google

I2CSparar

Sintaxis:

I2CSparar

Disponibilidad de comandos:

Disponible en todos los microcontroladores excepto en los PIC de instrucción de 12 bits (10F, 12F5xx, microprocesadores 16F5xx)

Explicación:

Cuando está en modo Maestro, este comando enviará una condición de parada I2C y volverá a habilitar las interrupciones si I2CStart las deshabilitó. En modo Esclavo, volverá a habilitar las interrupciones.

Se debe llamar a I2CStop al final de cada transmisión I2C.

Ejemplo:

Consulte I2CSend e I2CReceive para ver un ejemplo.

Si entonces más

Sintaxis:

Forma corta:

Si condición Entonces comando

Forma larga:

Sample Version - Do Not Copy

Machine Translated by Google

Si condición Entonces

...

código de programa

...

Terminara si

Usando Else:

Si condición Entonces

código para ejecutar si es verdadero

Demás

código para ejecutar si es falso

Terminara si

Explicación:

El comando If es el comando más común utilizado para tomar decisiones. Si la condición es verdadera, se ejecutará el comando (corto) o el código de programa (largo). Si es falso, el robot saltará al código ubicado en la línea siguiente (corta) o después de End If (forma larga).

Si se usa Else, la condición entre If y Else se ejecutará si la condición es verdadera, y el código entre Else y End If se ejecutará si la condición es falsa.

Ejemplo:

'Enciende o apaga una luz dependiendo de un sensor de luz

#chip 12F683, 8

Sample Version - Do Not Copy

Machine Translated by Google

```
#config osc = int


#define LUZ GPIO.1

#define SENSOR AN3

#define SENSOR_PORT GPIO.4


Dir LUZ Fuera

Dir SENSOR_PORT Entrada


Hacer

Si ReadAD(SENSOR) > 128 Entonces

Desactivar LUZ

Demás

Encender LUZ

Terminara si

Bucle
```

InitGLCD

Sintaxis:

InitGLCD

Explicación:

Esto inicializa la pantalla LCD gráfica para su funcionamiento. Aquí están los pasos que toma:

Sample Version - Do Not Copy

Machine Translated by Google

```
'Establecer direcciones de pines

Dir GLCD_RS Salida
Dir GLCD_RW Salida
Dirección GLCD_ENABLE Salida
Dir GLCD_CS1 Salida
Dir GLCD_CS2 Salida
Dirección GLCD_RESET Salida

'Reiniciar

Desactivar GLCD_RESET
Espera 1 ms
Activar GLCD_RESET
Espera 1 ms

'Seleccionar ambas fichas

Activar GLCD_CS1
Activar GLCD_CS2

'Encender

Desactivar GLCD_RS
GLCDWriteByte 63

'Establecer Z en 0

GLCDWriteByte 192

'Deseleccionar fichas

Desactivar GLCD_CS1
Desactivar GLCD_CS2

'Pantalla clara

GLCDCLS
```

Sample Version - Do Not Copy

Machine Translated by Google

InitSer

Sintaxis:

InitSer canal, velocidad, inicio, datos, parada, paridad, invertir

Explicación:

Este comando configurará las comunicaciones seriales. Los parámetros son los siguientes:

el canal es 1, 2 o 3, y se refiere a los puertos de E/S que se utilizan para la comunicación.

rate es la tasa de bits, que viene dada por la letra r y luego la tasa deseada en bps. Las unidades aceptables son r300, r600, r1200, r2400, r4800, r9600 y r19200.

start proporciona el número de bits de inicio, que suele ser 1. Para hacer que el PIC espere el bit de inicio antes de continuar con la recepción, agregue 128 a start. (Nota: puede ser deseable usar la constante WaitForStart aquí).

data le dice al programa cuántos bits de datos se van a enviar o recibir. En la mayoría de las situaciones es 8, pero puede oscilar entre 1 y 8, inclusive.

parada es el número de bits de parada. Si el bit de inicio 7 está activado, este número será ignorado

la paridad se refiere a un sistema de comprobación de errores utilizado por muchos dispositivos. Puede ser impar (donde siempre debe haber un número impar de bits altos), par (donde el número de bits altos siempre debe ser par) o ninguno (para sistemas que no usan paridad).

invertir puede ser "normal" o "invertir". Si está en "invertir", los bits altos se cambiarán a bajos y los bajos a altos.

Ejemplo:

Consulte SerSend para ver un ejemplo de InitSer

Sample Version DO Not Copy

Machine Translated by Google

InitTimer0

Sintaxis:

fuente InitTimer0, preescalador

Disponibilidad de comandos:

Disponible en todos los microcontroladores con módulo Timer 0.

Explicación:

InitTimer0 configurará el temporizador 0, de acuerdo con la configuración proporcionada. la fuente puede ser Osc o Ext.

En un microcontrolador PIC, el preescalador puede tener una de las siguientes configuraciones:

•PS0_2

•PS0_4

•PS0_8

•PS0_16

•PS0_32

•PS0_64

•PS0_128

•PS0_256

Corresponden a un prescaler de entre 1/2 y 1/256 de la velocidad del oscilador.
El preescalador se aplicará al oscilador o a la entrada del reloj externo.

Sample Version - Do Not Copy

Machine Translated by Google

Cuando el temporizador se desborda de 255 a 0, se generará una interrupción Timer0Overflow. Esto se puede usar junto con On Interrupt para ejecutar una sección de código periódicamente.

Ejemplo:

```
'Este código usará el temporizador 0 y On Interrupt para generar un pulso
'Señal de modulación de ancho, que permitirá aumentar la velocidad de un motor
ser fácilmente controlado.

#chip 16F88, 8
#config osc = int

#define PUERTO DE MOTORB.0

'Llamar a la rutina de inicialización
InitMotorControl

'Rutina principal
Hacer
'Aumenta la velocidad al máximo durante 2,5 segundos
Para Velocidad = 0 a 100
MotorSpeed = Velocidad
Espere 25 ms
Próximo
'Mantener la velocidad
Espera 1 s
```

Sample Version - Do Not Copy

Machine Translated by Google

```
'Disminuir la velocidad a cero durante 2,5 segundos
Para Velocidad = 100 a 0
MotorSpeed = Velocidad
Espere 25 ms
Próximo
'Mantener la velocidad
Espera 1 s
Bucle


'Rutina de configuración
Sub InitMotorControl
'Borrar variables
Velocidad del motor = 0
Contador PWM = 0


'Añadir un controlador para la interrupción
En Interrupción Timer0Overflow Call PWMHhandler


'Configurar el temporizador
InitTimer0 Osc, PS0_1/2
'El temporizador 0 se inicia automáticamente en un PIC
Finalizar sub


'Sub PWM
'Esto se llamará cuando el temporizador 0 se desborde
```

Sample Version - Do Not Copy

Machine Translated by Google

```
Manejador PWM secundario

Si MotorSpeed > PWMCounter Entonces

Activar MOTOR

Demás

Desactivar MOTOR

Terminara si

Contador PWM += 1

Si PWMCounter = 100 Entonces PWMCounter = 0

Finalizar sub
```

InitTimer1

Sintaxis:

fuente InitTimer1, preescalador

Disponibilidad de comandos:

Disponible en todos los microcontroladores con módulo Timer 1.

Explicación:

InitTimer1 configurará el temporizador 1, de acuerdo con la configuración dada. la fuente puede ser , Osc Ext o ExtOsc (ExtOsc solo está disponible en microcontroladores PIC). En un PIC, el preescalador puede tener una de las siguientes configuraciones:

•PS1_1

•PS1_2

Sample Version - Do Not Copy

Machine Translated by Google

•PS1_4

•PS1_8

Ejemplo:

```
'Este ejemplo medirá el tiempo que un interruptor permanece encendido

#chip 16F819, 20

#define Interruptor PORTA.0



Dir cambiar en

Recuento de datos = 0

InitTimer Osc, PS1_8



Dim TimerValue como palabra



Hacer

ClearTimer 1

Esperar hasta que el interruptor = encendido

Temporizador de inicio 1

Esperar hasta que el interruptor = Apagado

detener el temporizador 1



'Leer el temporizador

ValorTemporizador = Temporizador1



'Registrar el valor del temporizador
```

Sample Version - Do Not Copy

Machine Translated by Google

```
EPWrite(DataCount, TimerValue_H)

EPWrite(DataCount + 1, TimerValue)

Recuento de datos += 2

Bucle
```

INKEY

Sintaxis:

salida = INKEY

Explicación:

La función INKEY leerá la última tecla presionada desde un teclado PS/2 y devolverá un valor ASCII correspondiente a la tecla. Si no se presiona ninguna tecla, INKEY devolverá 0.

También controlará las teclas Caps Lock, Num Lock y Scroll Lock, y actualizará los LED de estado según corresponda.

Ejemplo:

```
'Un programa para aceptar mensajes de un teclado estándar PS/2

'Cualquier tecla presionada se mostrará en una pantalla LCD.


'Configuración de hardware

#chip 18F4620, 20


'Configuración de conexión LCD
```

Sample Version - Do Not Copy

Machine Translated by Google

```
#definir LCD_IO 4

#define LCD_DB4 PUERTO.4

#define LCD_DB5 PUERTO.5

#define LCD_DB6 PUERTO.6

#define LCD_DB7 PUERTO.7

#define LCD_RS PUERTO.0

#define LCD_RW PUERTO.1

#define LCD_Habilitar PUERTO.2


'Configuración de conexión PS/2

#define PS2Clock PORTC.1

#define PS2Data PORTC.0

#define PS2_DELAY 10ms


'Configurar registro de claves

Dim KeyLog(32)

Recuento de datos = 0

Registro de teclas (1) = 32


Principal:

'Leer la última tecla presionada

Entrada de tecla = ENTRADA

'Si no se presiona ninguna tecla, intente leer de nuevo

Si KeyIn = 0 Entonces Ir a Principal
```

Sample Version - Do Not Copy

Machine Translated by Google

```
'Escape pulsado - borrar mensaje
Si KeyIn = 27 Entonces
Recuento de datos = 0
Para DataPos = 1 a 32
KeyLog(DataPos) = 32
Próximo
Ir a datos de visualización
Terminara si


'Retroceso presionado - eliminar el último carácter
Si KeyIn = 8 Entonces
Si DataCount = 0 Entonces Ir a Principal
Registro de teclas (recuento de datos) = 32
Recuento de datos = Recuento de datos - 1
Ir a datos de visualización
Terminara si


'De lo contrario, agregue el carácter al búfer
Si KeyIn >= 31 y KeyIn <= 127 Entonces
Recuento de datos = Recuento de datos + 1
KeyLog(DataCount) = Entrada de tecla
Terminara si
Mostrar datos:
'Mostrar búfer clave
'LCDWriteChar se usa en lugar de Print para un mayor control
CLS
```

Sample Version - Do Not Copy

Machine Translated by Google

```
Para DataPos = 1 a DataCount

Si DataPos = 17, busque 1, 0

LCDWriteChar KeyLog(DataPos)

Próximo
```

Ir a principal

Interrupciones

Acerca de las
interrupciones Las interrupciones son una característica de muchos microcontroladores. Permiten que el microcontrolador pause (interrumpa) temporalmente el código que está ejecutando y luego comience a ejecutar otra pieza de código cuando ocurre algún evento. Una vez que se ha ocupado del evento, volverá a donde estaba y continuará ejecutando el programa.

Muchos eventos pueden desencadenar una interrupción, como que un temporizador llegue a su límite, que se reciba un mensaje en serie o que un pin especial en el microcontrolador reciba una señal.

Uso de interrupciones
Hay dos formas de usar interrupciones en GCBASIC. La primera forma es usar el comando On Interrupt. Esto habilitará automáticamente una interrupción dada y ejecutará una subrutina particular cuando ocurra la interrupción.

La otra forma de lidiar con las interrupciones es crear una subrutina llamada Interrupción. GCBASIC llamará a esta subrutina cada vez que ocurra una interrupción, y luego su código puede verificar los bits de "bandera" para determinar qué interrupción ha ocurrido y qué se debe hacer al respecto. Si utiliza este enfoque, deberá habilitar las interrupciones deseadas manualmente. También es esencial que su código borre los bits de bandera, o de lo contrario, la rutina de interrupción se llamará repetidamente.

También es posible alguna combinación de estos dos métodos: el código generado por On Interrupt con verificación para ver si la interrupción es una que reconoce. Si se reconoce la interrupción, On Interrupt se encargará de ella; si no, se llamará a la subrutina Interrupt para que se ocupe de la interrupción.

Sample Version - Do Not Copy

Machine Translated by Google

La forma recomendada es usar On Interrupt, ya que es más eficiente y más fácil de configurar.

Durante algunas secciones del código, es deseable que no ocurra ninguna interrupción. Si este es el caso, utilice el comando IntOff para deshabilitar las interrupciones al comienzo de la sección e IntOn para volver a habilitarlas al final. Si se producen eventos de interrupción mientras las interrupciones están deshabilitadas, se procesarán tan pronto como se vuelvan a habilitar. Si el programa no utiliza interrupciones, GCBASIC eliminará automáticamente IntOn e IntOff.

IntOff

Sintaxis:

IntOff

Disponibilidad de comandos:
disponible en microcontroladores PIC con soporte de interrupción. Se eliminará automáticamente en fichas sin interrupciones.

Explicación:
IntOff se usa para deshabilitar las interrupciones en el microcontrolador. Debe usarse al comienzo del código que es sensible al tiempo y que no funcionaría correctamente si se pausara y reiniciara.

Es esencial que IntOn se use para volver a activar las interrupciones después de que el código sensible al tiempo haya terminado de ejecutarse. Si no, no se manejarán las interrupciones.

IntOff se eliminará del programa si no se utilizan interrupciones. Se recomienda colocar IntOff antes de todo el código sensible al tiempo, en caso de que se implementen interrupciones más adelante.

Sample Version - Do Not Copy

Machine Translated by Google

IntOn

Sintaxis:

IntOn

Disponibilidad de comandos:

Disponible en microcontroladores PIC con soporte de interrupción. Se eliminará automáticamente en fichas sin interrupciones.

Explicación:

IntOn se usa para habilitar interrupciones en el microcontrolador después de que IntOff las haya deshabilitado. Debe usarse al final del código que es sensible al tiempo.

IntOn se eliminará del programa si no se utilizan interrupciones.

Datos del teclado

Sintaxis:

var = datos del teclado

Explicación:

Esta función devolverá un valor correspondiente a la tecla que se presiona en el teclado. Tenga en cuenta que si se presionan dos o más teclas, solo se devolverá un valor.

Sample Version - Do Not Copy

Machine Translated by Google

var puede tener uno de los siguientes valores:

Valor	Nombre constante	Tecla presionada
0		0
1		1
2		2
3		3
4		4
5		5
6		6
7		7
8		8
9		9
10	CLAVE_A	A
11	CLAVE_B	B
12	CLAVE_C	C
13	CLAVE_D	D
14	CLAVE_ESTRELLA	Asterisco/Estrella (*)
15	CLAVE_HASH	Hachís (#)
255	CLAVE_NINGUNA	Ninguno

Sample Version - Do Not Copy

Ejemplo:

```
'Programa para mostrar el valor de la última tecla presionada en la pantalla LCD

#chip 18F4550, 20


'Configuración de conexión LCD
```

Machine Translated by Google

```
#definir LCD_IO 4

#define LCD_DB4 PUERTO.4

#define LCD_DB5 PUERTO.5

#define LCD_DB6 PUERTO.6

#define LCD_DB7 PUERTO.7

#define LCD_RS PUERTO.0

#define LCD_RW PUERTO.1

#define LCD_Habilitar PUERTO.2


'Configuración de conexión del teclado

#define KeypadPort PORTB


'Bucle principal

Hacer

'Obtener la clave

Temp = Datos del teclado


'Si se presiona una tecla, mostrarla

Si Temp <> KEY_NONE Entonces

CLS

Temperatura de impresión

Espera 100ms

Terminara si

Bucle
```

Sample Version - Do Not Copy

Machine Translated by Google

TecladoRaw

Sintaxis:

largevar = KeypadRaw

Explicación:

Esta función devolverá un valor de 16 bits, en el que cada bit corresponde a una tecla del teclado. Si se presiona la tecla, su bit contendrá un 1, y si se suelta, su bit contendrá un 0.

En esta tabla se muestra la clave a la que corresponde cada bit:

Poco	Posición clave (fila, columna)	Símbolo de clave común
15	1,1	1
14	1,2	2
13	1,3	3
12	1,4	A
11	2,1	4
10	2,2	5
9	2,3	6
8	2,4	B
7	3,1	7
6	3,2	8
5	3,3	9
4	3,4	C
3	4,1	*
2	4,2	0
1	4,3	#
0	4,4	D

Sample Version - Do Not Copy

Machine Translated by Google

Ejemplo:

```
'Programa para mostrar el estado del teclado mediante LEDs

#chip 16F877A, 20


'Configuración de conexión del teclado

#define KeypadPort PORTB


'LED

#define LED1 PUERTO

#define PUERTO LED2

Dirección LED1 Salida

Dir LED2 Salida


'Declarar una variable de 16 bits para el valor clave

Dim KeyStatus como palabra


'Bucle principal

Hacer

'Obtener la clave

KeyStatus = KeypadRaw

'Mostrar

LED1 = KeyStatus_H 'Byte alto

LED2 = KeyStatus 'Byte bajo

Bucle
```

Sample Version - Do Not Copy

Machine Translated by Google

LCDCreateChar

Sintaxis:

LCDCreateChar char, chardata()

Explicación:

El comando LCDCreateChar se utiliza para enviar un carácter personalizado a la pantalla LCD.

Cada carácter en la pantalla LCD se compone de una matriz de píxeles de 8 filas por 5 columnas (5x8). Los datos que se enviarán a la pantalla LCD se componen de una matriz de 8 elementos, donde cada elemento corresponde a una fila. Dentro de cada elemento, los 5 bits más bajos componen los datos de la fila correspondiente. Cuando se establece un bit, se dibujará un punto en la ubicación correspondiente; cuando se borre, no aparecerá ningún punto.

Se puede utilizar una matriz de más de 8 elementos, pero solo se leerán los 8 primeros.

char es el valor ASCII del carácter a crear. Los códigos ASCII del 0 al 7 generalmente se usan para almacenar caracteres personalizados.

chardata() es una matriz que contiene los datos del carácter.

Ejemplo:

```
'Este programa dibuja un personaje de cara sonriente

'Configuración general de hardware
```

Sample Version - Do Not Copy

Machine Translated by Google

```
#chip 16F877A, 20


 'Configuración de conexión LCD

#definir LCD_IO 8

#define LCD_DATA_PORT PUERTO

#define LCD_RS PUERTO.0

#define LCD_RW PUERTO.1

#define LCD_Habilitar PUERTO.2


 'Crea una matriz para almacenar el carácter hasta que se copie

Atenuación TempArray(8)


 'Establecer la matriz para contener el carácter

 'Se ha utilizado binario para mejorar la legibilidad del código, pero no es esencial

TempArray(1) = b'00011011'

TempArray(2) = b'00011011'

TempArray(3) = b'00000000'

TempArray(4) = b'00000100'

TempArray(5) = b'00000000'

TempArray(6) = b'00010001'

MatrizTemp(7) = b'00010001'

TempArray(8) = b'00001110'


 'Copia el caracter de la matriz a la pantalla LCD

LCDCreateChar 0, TempArray()
```

Sample Version - Do Not Copy

Machine Translated by Google

```
'Dibuja el personaje personalizado

LCDWriteChar 0
```

LCDHex

Sintaxis:

Valor LCDHex

Explicación:

El comando LCDHex mostrará el valor especificado en la pantalla LCD, en la posición actual del cursor. Convertirá el byte que se le ha dado a formato hexadecimal.

Ejemplo:

```
'Un programa para contar de 0 a FF en una pantalla LCD.


'Configuración general de hardware

#chip 16F877A, 20


'Configuración de conexión LCD

#definir LCD_IO 8

#define LCD_DATA_PORT PUERTO

#define LCD_RS PUERTO.0

#define LCD_RW PUERTO.1
```

Sample Version - Do Not Co

Machine Translated by Google

```
#define LCD_Habilitar PUERTO.2


Para contador = 0 a 255

CLS

Contador LCDHex

Espere 250ms

Próximo
```

LCDWriteChar

Sintaxis:

LCDWriteChar char

Explicación:

El comando LCDWriteChar mostrará el carácter especificado en la pantalla LCD, en la posición actual del cursor.

char es el valor ASCII del carácter a mostrar. En la mayoría de las pantallas LCD, los caracteres del 0 al 7 están definidos por el usuario y se pueden configurar con el comando LCDCreateChar.

Ejemplo:

```
'Este programa dibuja un personaje de cara sonriente


'Configuración general de hardware
```

Sample Version - Do Not Copy

Machine Translated by Google

```
#chip 16F877A, 20


 'Configuración de conexión LCD

#definir LCD_IO 8

#define LCD_DATA_PORT PUERTO

#define LCD_RS PUERTO.0

#define LCD_RW PUERTO.1

#define LCD_Habilitar PUERTO.2


 'Crea una matriz para almacenar el carácter hasta que se copie

Atenuación TempArray(8)


 'Establecer la matriz para contener el carácter

TempArray(1) = b'00011011'

TempArray(2) = b'00011011'

TempArray(3) = b'00000000'

TempArray(4) = b'00000100'

TempArray(5) = b'00000000'

TempArray(6) = b'00010001'

MatrizTemp(7) = b'00010001'

TempArray(8) = b'00001110'


 'Copia el caracter de la matriz a la pantalla LCD

LCDCreateChar 0, TempArray()
```

Sample Version - Do Not Copy

Machine Translated by Google

```
'Dibuja el personaje personalizado

LCDWriteChar 0
```

Línea

Sintaxis:
Línea (LíneaX1, LíneaY1, LíneaX2, LíneaY2, Color de línea opcional = 1)

Explicación:
Dibuja una línea en una pantalla LCD gráfica desde la ubicación X1, Y1 hasta la ubicación X2, Y2.
Debe ser horizontal o vertical. Diagonal no funcionará correctamente.

Localizar

Sintaxis:

Ubicar línea, columna

Explicación:

El comando Localizar se usa para mover el cursor en la pantalla LCD a la ubicación dada.

Ejemplo:

Sample Version - Do Not Copy

Machine Translated by Google

```
'Un programa Hello World para GCBASIC.
'Utiliza Localizar para mostrar "Mundo" en la segunda línea

'Configuración general de hardware
#chip 16F877A, 20

'Configuración de conexión LCD
#definir LCD_IO 8
#define LCD_DATA_PORT PUERTO
#define LCD_RS PUERTO.0
#define LCD_RW PUERTO.1
#define LCD_Habilitar PUERTO.2

'Rutina principal
Imprimir "Hola"
Localizar 1, 5
Imprimir "Mundo"
```

en interrupción

Sintaxis:

En evento de interrupción Controlador de llamadas

En evento de interrupción Ignorar

Disponibilidad de comandos:

Sample Version - Do Not Copy

Machine Translated by Google

Disponible en microcontroladores PIC con soporte de interrupción.

Explicación:

On Interrupt agregará código para llamar al controlador de subrutina cada vez que ocurra el evento de interrupción. Cuando se especifica Call, GCBASIC habilitará la interrupción y llamará al controlador de interrupciones cuando ocurra. Cuando se especifica Ignorar, GCBASIC deshabilitará el controlador de interrupciones y evitará que se llame cuando ocurra el evento. Si el evento ocurre mientras el controlador está deshabilitado, se llamará al controlador tan pronto como se vuelva a habilitar. La única forma de evitar que esto suceda es borrar manualmente el bit indicador de la interrupción.

Hay muchos eventos de interrupción posibles que pueden ocurrir, y los eventos varían mucho de un chip a otro. GCBASIC mostrará un error si un chip dado no puede admitir el evento especificado.

En algunos casos, On Interrupt no podrá establecer o borrar el indicador de interrupción y/o los bits de habilitación. Si este es el caso, GCBASIC mostrará una advertencia. Deberá consultar la hoja de datos del chip y usar el comando Set para configurar/borrar manualmente el indicador y habilitar los bits, tanto al inicio del programa como dentro de la subrutina del controlador de interrupciones.

Si se usa On Interrupt para manejar un evento, entonces no se llamará a la subrutina Interrupt para ese evento. Sin embargo, aún se llamará para cualquier evento no tratado por On Interrupt.

Eventos:

GCBASIC admite los eventos que se muestran en la siguiente tabla. Algunos eventos solo se implementan en unos pocos chips especializados. Los eventos en negrita solo son compatibles con algunos PIC.

Tenga en cuenta que GCBASIC no es totalmente compatible con todo el hardware que puede generar interrupciones; es posible que se requiera algo de trabajo con varias variables del sistema para controlar los periféricos no compatibles.

Sample Version - Do Not Copy

Machine Translated by Google

Nombre del evento	Descripción
Listo para ADC	El conversor analógico/digital ha finalizado una conversión.
CANActividad	La actividad del bus CAN está teniendo lugar
Mensaje incorrecto de CAN	Se ha recibido un mensaje CAN erróneo
CANError	Se ha producido algún error de CAN
CANHighMarca de agua	Marca de agua alta de CAN alcanzada
CANRx0Listo	Nuevo mensaje presente en el búfer 0
CANRx1Listo	Nuevo mensaje presente en el búfer 1
CANRx2Listo	Nuevo mensaje presente en el búfer 2
CANRxListo	Nuevo mensaje presente
CANTx0Listo	Se ha enviado el búfer 0
CANTx1Listo	Se ha enviado el búfer 1
CANTx2Listo	Se ha enviado el búfer 2
CANTxReady	El envío se ha completado
PCC1	El módulo CCP1 ha capturado un evento
PCC2	El módulo CCP2 ha capturado un evento
CCP3	El módulo CCP3 ha capturado un evento
CCP4	El módulo CCP4 ha capturado un evento
PCC5	El módulo CCP5 ha capturado un evento
Comp0Change	La salida del comparador 0 ha cambiado
Comp1Change	La salida del comparador 1 ha cambiado
Comp2Change	La salida del comparador 2 ha cambiado
Cripto	El módulo KEELOQ ha generado una interrupción
EEPROMListo	Una escritura de EEPROM ha terminado
ethernet	El módulo Ethernet ha generado una interrupción. Esto debe ser tratado en el controlador.
ExtInt0	El pin de interrupción externa 0 ha sido editado
ExtInt1	El pin de interrupción externa 1 ha sido editado
ExtInt2	El pin de interrupción externa 2 ha sido editado
ExtInt3	El pin de interrupción externa 3 ha sido editado
GPIOCambiar	Los pines del puerto GPIO han cambiado
LCDListo	La pantalla LCD está a punto de dibujar un segmento.
LPWU	El Low Power Wake Up ha sido editado

Sample Version - Do Not Copy

Machine Translated by Google

falla del oscilador	El oscilador externo ha fallado y el PIC se está ejecutando desde un oscilador interno.
PMP Listo	Ha finalizado la lectura o escritura de un puerto maestro paralelo
PORTACambiar	Los pines en el puerto A han cambiado
PORTABCambio	Los pines en el puerto A y/o B han cambiado
PUERTOCambiar	Los pines en el puerto B han cambiado
PSP Listo	Ha finalizado la lectura o escritura de un puerto esclavo paralelo
PWMTimeBase	La base de tiempo de PWM coincide con el registro de período de base de tiempo de PWM (PTPER)
Listo para SPP	Ha finalizado una lectura o escritura de SPP
SSP1Colisión	SSP1 ha detectado una colisión de autobús
SSP1Listo	El módulo SSP/SSP1/MSSP1 ha terminado de enviar o recibir
SSP2Colisión	SSP2 ha detectado una colisión de autobús
Listo para SSP2	El módulo SSP2/MSSP2 ha terminado de enviar o recibir
Temporizador0Desbordamiento	El temporizador 0 se ha desbordado
Desbordamiento del temporizador1	El temporizador 1 se ha desbordado
Desbordamiento del temporizador2	El temporizador 2 se ha desbordado
Timer3Desbordamiento	El temporizador 3 se ha desbordado
Timer4Overflow	El temporizador 4 se ha desbordado
Temporizador5CAP1	Una entrada en el pin CAP1 ha provocado que el valor del temporizador 5 se capture en CAP1BUF
Temporizador5CAP2	Una entrada en el pin CAP2 ha provocado que el valor del temporizador 5 se capture en CAP2BUF
Temporizador5CAP3	Una entrada en el pin CAP3 ha provocado que el valor del temporizador 5 se capture en CAP3BUF
Temporizador5Partido	Timer5 coincide con el registro PR5
UsartRX1Listo	UART/USART 1 ha recibido datos
UsartRX2Ready	UART/USART 2 ha recibido datos
UsartTX1Listo	UART/USART 1 está listo para enviar datos
UsartTX2Ready	UART/USART 2 está listo para enviar datos
USB	El módulo USB ha generado una interrupción. Esto debe ser tratado en el controlador.
Falla de voltaje	El voltaje de entrada ha bajado demasiado

Sample Version - Do Not Copy

Machine Translated by Google

Ejemplo:

```
'Este programa incrementa un contador cada vez que se desborda el Timer1

#chip 16F877A, 20



'Configuración de conexión LCD

#definir LCD_IO 4

#define LCD_DB4 PUERTO.4

#define LCD_DB5 PUERTO.5

#define LCD_DB6 PUERTO.6

#define LCD_DB7 PUERTO.7

#define LCD_RS PUERTO.0

#define LCD_RW PUERTO.1

#define LCD_Habilitar PUERTO.2



InitTimer1 Osc, PS1_1/8

Temporizador de inicio 1

Contravalor = 0



Espera 100ms

Imprimir "Prueba Int"



On Interrupt Timer1Overflow Call IncCounter



Hacer

CLS
```

Sample Version - Do Not Copy

Machine Translated by Google

Imprimir contravalor

Espera 100ms

Bucle

Contador de subinc.

Contravalor ++

Finalizar sub

Para obtener más ayuda, consulte:

El artículo InitTimer0 contiene un ejemplo del uso de Timer 0 y On Interrupt para generar una señal de modulación de ancho de pulso para controlar un motor.

Ojeada

Sintaxis:

OutputVariable = Peek (ubicación)

Explicación:

La función Peek se usa para leer información de la memoria RAM en el chip del microcontrolador.

ubicación es una palabra variable que da la dirección para leer. El rango exacto de valores válidos varía de un chip a otro.

Sample Version - Do Not Copy

Machine Translated by Google

Este comando normalmente no debe usarse, ya que dificultará mucho la transferencia del código a otro chip.

Ejemplo:

```
'Este programa leerá y verificará un valor de PORTA

'Solo funcionará en algunos PIC

Temperatura = PEEK(5)

SI Temp.2 ENCENDIDO ENTONCES SET verde ENCENDIDO

SI Temp.2 APAGADO ENTONCES SET rojo ENCENDIDO
```

Dar un toque

Sintaxis:

Poke (ubicación, valor)

Explicación:

El comando Poke se usa para escribir información en la memoria RAM del microcontrolador.

ubicación es una palabra variable que da la dirección a escribir. El rango exacto de valores válidos varía de un chip a otro.

El valor es el dato para escribir en la ubicación.

Sample Version - Do Not Copy

Machine Translated by Google

Este comando normalmente no debe usarse, ya que dificultará mucho la transferencia del código a otro chip.

Ejemplo:

```
'Este programa establecerá todos los pines PORTB altos

EMPUJAR (6, 255)
```

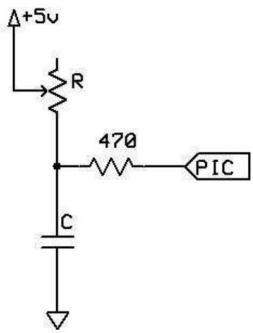
Maceta

Sintaxis:

```
Pasador de potenciómetro, salida
```

Explicación:

Pot permite medir una resistencia analógica con un puerto digital, con la adición de un pequeño condensador. Este es el circuito requerido:



El comando funciona utilizando el pin del microcontrolador para descargar el capacitor, luego mide el tiempo que tarda el capacitor en cargarse nuevamente a través de la resistencia.

Sample Version - Do Not Copy

Machine Translated by Google

El valor del capacitor debe ajustarse según el tamaño de la resistencia variable.
El tiempo de carga debe ser de aproximadamente 2,5 ms cuando la resistencia está en su valor máximo. Para un potenciómetro o LDR típico de 50 k, se requiere un capacitor de 50 nf.

Este comando debe usarse con cuidado. Cada vez que se inserta, se utilizan 20 palabras de memoria de programa en el chip, que como guía aproximada es más de 15 veces el tamaño del comando Set.

pin es el puerto conectado al circuito. La dirección del pin será tratada por el comando Pot.

salida es el nombre de la variable que recibirá el valor.

Ejemplo:

```
'Este programa emitirá un pitido cada vez que se detecte una sombra

'Se utiliza un potenciómetro para ajustar el umbral

#chip 16F628A, 4

#config INTOSC_OSC_NOCLKOUT

#define AJUSTAR PUERTO.0

#define PUERTO LDR.1

#define SoundOut PUERTO.2

Dir Salida de sonido


Hacer

Pot ADJUST, Umbral

Pot LDR, nivel de luz

Si Nivel de luz > Umbral Entonces

Tono 1000, 100

Terminara si

Bucle
```

Sample Version - Do Not Copy

Machine Translated by Google

Imprimir

Sintaxis:

cadena de impresión

Imprimir byte

Imprimir palabra

Imprimir entero

Explicación:

El comando Imprimir mostrará el contenido de una variable en la pantalla LCD. Puede mostrar variables de cadena, palabra o byte.

Ejemplo:

```
'Un programa de medidor de luz.

'Configuración general de hardware
#chip 16F877A, 20
#define sensor de luz AN0

'Configuración de conexión LCD
#definir LCD_IO 8
#define LCD_DATA_PORT PUERTO
#define LCD_RS PUERTO.0
#define LCD_RW PUERTO.1
```

Sample Version - Do Not Copy

Machine Translated by Google

```
#define LCD_Habilitar PUERTO.2


CLS

Imprimir "Medidor de luz:

Localizar 1,2

Imprimir "Una demostración de GCBASIC"

Espera 2 segundos


Hacer

CLS

Imprimir "Nivel de luz:"

Imprimir ReadAD (Sensor de luz)

Espere 250ms

Bucle
```

ProgramaBorrar

Sintaxis:

ProgramErase(ubicación)

Disponibilidad de comandos:

Disponible en todos los chips PIC con capacidad de autoescritura.

Explicación:

Sample Version - Do Not Copy

Machine Translated by Google

ProgramErase borra información de la memoria del programa en chips que admiten esta función. El mayor valor posible para la ubicación depende de la cantidad de memoria de programa en el PIC, que se proporciona en la hoja de datos.

Este comando debe llamarse antes de escribir en un bloque de memoria. Es lento en comparación con otros comandos GCBASIC. Tenga en cuenta que borra la memoria en bloques de 32 bytes; consulte la hoja de datos PIC correspondiente para obtener más información.

Este es un comando avanzado que solo deben usar los desarrolladores avanzados. Se debe tener cuidado con este comando, ya que puede borrar fácilmente el programa que se está ejecutando en el PIC.

ProgramaLeer

Sintaxis:

ProgramRead (ubicación, tienda)

Disponibilidad de comandos:

Disponible en todos los chips PIC con capacidad de autoescritura.

Explicación:

ProgramRead lee información de la memoria del programa en chips que admiten esta función. location y store son variables de palabra, lo que significa que pueden almacenar valores superiores a 255.

El mayor valor posible para la ubicación depende de la cantidad de memoria de programa en el PIC, que se proporciona en la hoja de datos. store tiene 14 bits de ancho y, por lo tanto, puede almacenar valores hasta 16383.

Este es un comando avanzado que solo deben usar los desarrolladores avanzados.

Sample Version - Do not Copy

Machine Translated by Google

ProgramaEscribir

Sintaxis:

ProgramWrite (ubicación, valor)

Disponibilidad de comandos:

Disponible en todos los chips PIC con capacidad de autoescritura.

Explicación:

ProgramWrite escribe información en la memoria del programa en chips que admiten esta función. ubicación y valor son variables de palabra.

El mayor valor posible para la ubicación depende de la cantidad de memoria de programa en el PIC, que se proporciona en la hoja de datos. El valor tiene 14 bits de ancho y, por lo tanto, puede almacenar valores hasta 16383.

Este es un comando avanzado que solo deben usar los desarrolladores avanzados. ProgramErase debe usarse para borrar un bloque de memoria ANTES de llamar a ProgramWrite.

Sample Version - Do Not Copy

PS2ReadByte

Sintaxis:

salida = PS2ReadByte

Explicación:

Machine Translated by Google

PS2ReadByte leerá un byte del bus PS/2. Devolverá el byte, o 0 si el dispositivo PS/2 no devolvió datos.

El bus PS/2 normalmente se mantendrá en el estado de inhibición. PS2ReadByte desinhibirá el bus durante 25 ms. Si se recibe una respuesta, se leerá. Luego, el bus se colocará nuevamente en el estado de inhibición.

Ejemplo:

Para ver un ejemplo, consulte la función INKEY en el archivo ps2.h.

PS2SetKBLeds

Sintaxis:

PS2SetKBLeds (LedStatus)

Explicación:

Esta rutina encenderá o apagará los LED de estado en un teclado. LedStatus es una variable, de los cuales los 3 bits inferiores corresponden a los 3 LED. El bit 0 es para el bloqueo de desplazamiento, el bit 1 controla el bloqueo numérico y el bit 2 controla el bloqueo de mayúsculas.

Tenga en cuenta que esta rutina no altera las variables de estado dentro de la rutina INKEY, por lo que incluso si el LED de bloqueo de mayúsculas está encendido, el bloqueo de mayúsculas permanecerá desactivado.

Sample Version - Do Not Copy

Ejemplo:

```
'Un programa LED giratorio para un teclado

'Parpadeará Num Lock, luego Caps Lock, luego Scroll Lock.

'Configuración de hardware

#chip 16F88, 8
```

Machine Translated by Google

```
#define PS2Clock PORTB.2

#define PS2Data PUERTO.3

#define PS2_DELAY 10ms



'Bucle principal

Hacer

'Activar solo Num Lock (bit 1)

PS2SetKBLeds b'00000010'

Espere 250ms

'Activar solo Bloq Mayús (bit 2)

PS2SetKBLeds b'00000100'

Espere 250ms

'Activar solo Scroll Lock (bit 0)

PS2SetKBLeds b'00000001'

Espere 250ms



Bucle
```

PS2WriteByte

Sintaxis:

PS2WriteByte de datos

Explicación:

Sample Version - Do Not Copy

Machine Translated by Google

PS2WriteByte enviará un byte a un dispositivo PS/2. Una vez que se ha escrito el byte, el bus PS/2 se colocará en el estado de inhibición.

Ejemplo:

Para ver un ejemplo, consulte la función PS2SetKBLeds en el archivo ps2.h.

Pset

Sintaxis:
PSet(GLCDX, GLCDY, GLCDEstado)

Explicación:
establece o borra un píxel en la ubicación X, Y especificada. Un uno para GLCDState establece el píxel y un cero borra el píxel.

Pulso de salida

Sintaxis:

Pin PulseOut, unidades de tiempo

Explicación:

Sample Version - Do Not Copy

Machine Translated by Google

El comando PulseOut establecerá el pin especificado en alto, esperará la cantidad de tiempo especificada y luego volverá a establecer el pin en bajo. El pin se especifica de la misma manera que para el comando Establecer y el tiempo es el mismo que para el comando Esperar.

Ejemplo:

```
'Este programa hace parpadear un LED en GPIO.0 usando PulseOut

#chip 12F629, 4

#config INTRC_OSC_NOCLKOUT


Dir GPIO.0 Salida

Hacer

PulseOut GPIO.0, 1 seg 'Enciende el LED durante 1 seg

Esperar 1 segundo 'Esperar 1 segundo con el LED apagado

Bucle
```

Sample Version - Do Not Copy

Poner

Sintaxis:

Poner línea, columna, carácter

Explicación:

El comando Put escribe el código de carácter ASCII dado en la ubicación de la pantalla LCD.

Machine Translated by Google

Ejemplo:

```
'Una estrella móvil para GCBASIC

'Configuraciones misceláneas

#define SCROLL_DELAY 250ms


'Configuración general de hardware

#chip 16F877A, 20


'Configuración de conexión LCD

#definir LCD_IO 8

#define LCD_DATA_PORT PUERTO

#define LCD_RS PUERTO.0

#define LCD_RW PUERTO.1

#define LCD_Habilitar PUERTO.2


'Rutina principal

Para StarPos = 0 a 16

Si StarPos = 0 Entonces

Pon 0, 16, 32

Pon 0, 0, 42

Demás

Poner 0, StarPos - 1, 32

Poner 0, StarPos, 42

Terminara si
```

Sample Version - Do Not Copy

Machine Translated by Google

```
Espera SCROLL_DELAY

Próximo
```

PWM Apagado

Sintaxis:

```
PWM Apagado
```

Disponibilidad de comandos:

Solo disponible en microcontroladores PIC con módulo Capture/Compare/PWM (CCP).

Explicación:

Este comando deshabilitará la salida del módulo PWM en el chip PIC.

Ejemplo:

```
'Este programa habilitará una señal PWM de 76 Khz, con un ciclo de trabajo

'del 80%. Emitirá la señal durante 10 segundos y luego se detendrá.

#define PWM_Freq 76 'Establece la frecuencia en KHz

#define PWM_Duty 80 'Establece el ciclo de trabajo al 80 %

PWMOn 'Activar el PWM

ESPERE 10 s 'Espere 10 segundos

PWMOff 'Apaga el PWM
```

Sample Version - Do Not Copy

Machine Translated by Google

PWMOn

Sintaxis:

PWMOn

Disponibilidad de comandos:

Solo disponible en microcontroladores PIC con módulo Capture/Compare/PWM (CCP).

Explicación:

Este comando habilitará la salida del módulo PWM en el chip PIC.

Ejemplo:

```
'Este programa habilitará una señal PWM de 76 Khz, con un ciclo de trabajo
'del 80%. Emitirá la señal durante 10 segundos y luego se detendrá.
#define PWM_Freq 76 'Establece la frecuencia en KHz
#define PWM_Duty 80 'Establece el ciclo de trabajo al 80 %
PWMOn 'Activar el PWM
ESPERE 10 s 'Espere 10 segundos
PWMOff 'Apaga el PWM
```

Sample Version - Do Not Copy

Machine Translated by Google

Salida PWM

Sintaxis:

Canal PWMOut, ciclo de trabajo, ciclos

Explicación:

Este comando usa una rutina PWM de software incluida en GCBASIC para producir una señal PWM en el puerto seleccionado del chip. Esta rutina NO requiere un módulo PWM en el chip.

channel establece el canal en el que se generará el PWM. Esto debe haber sido definido previamente configurando las constantes SoftPWM1, SoftPWM2, etc. El número máximo de canales disponibles es 4.

ciclo de trabajo especifica el ciclo de trabajo de PWM y va de 0 a 255. 255 corresponde al 100 %, 127 al 50 %, 63 al 25 %, etc.

ciclos se utiliza para establecer la cantidad de pulsos PWM a suministrar. Esto es útil para situaciones en las que se requiere un pulso de una duración específica. La fórmula para calcular el tiempo necesario para un ciclo es:

TCYCLE = (28 + 10C)TOSC + 255PWM_Delay,

donde C es el número de canales utilizados y TOSC es el tiempo que se tarda en ejecutar 1 instrucción en el chip (0,2 us en un chip de 20 MHz, 1 us en un chip de 4 Mhz). PWM_Delay es un período de tiempo especificado mediante la constante PWM_Delay.

Ejemplo:

```
'Este programa controla el brillo de un LED en PORTB.0 'utilizando la rutina PWM del software y un potenciómetro.


'Seleccionar modelo de chip

#chip 16F877A, 20

'Establecer puerto PWM
```

Sample Version - Do Not Copy

Machine Translated by Google

```
#define PWM_Out1 PORTB.0

'Establecer direcciones de puerto

dir SoftPWM1 fuera 'PWM

dir PORTA.0 en 'Potenciómetro

'Rutina principal

hacer

PWMOut 1, ReadAD(AN0), 100 '100 ciclos es puramente

'valor arbitrario aquí.

Bucle
```

Aleatorio

Sintaxis:

var = Aleatorio

Explicación:

La función Random generará un número pseudoaleatorio entre 0 y 255 inclusive.

Los números generados por Random seguirán la misma secuencia cada vez, hasta que se utilice Randomize.

Ejemplo:

```
'Establecer modelo de chip

#chip diminuto2313, 1

'Usar aleatorio, con el valor en PORTD como semilla
```

Sample Version - Do Not Copy

Machine Translated by Google

```
Aleatorizar PUERTO

'Generar números aleatorios y salida en PORTB

Hacer

PORTB = Aleatorio

Espera 1 s

Bucle
```

Aleatorizar

Sintaxis:

Aleatorizar

Aleatorizar semilla

Explicación:

Randomize se usa para sembrar el generador de números pseudoaleatorios, de modo que produzca una secuencia diferente de números cada vez que se use.

Si no se especifica ninguna semilla, la constante RANDOMIZE_SEED se utilizará como semilla. Si se especifica semilla, se usará para sembrar el generador.

Es importante que la semilla sea diferente cada vez que se usa Randomize. Si la semilla es siempre la misma, entonces la secuencia de números siempre será la misma. Es mejor utilizar un temporizador de ejecución, un puerto de entrada o el convertidor de analógico a digital como fuente de la semilla, ya que normalmente proporcionarán un valor diferente cada vez que se ejecute el programa.

Ejemplo:

```
'Establecer modelo de chip
```

Compile Version - Do Not Copy

Machine Translated by Google

```
#chip diminuto2313, 1

'Usar aleatorio, con el valor en PORTD como semilla

Aleatorizar PUERTO

'Generar números aleatorios y salida en PORTB

Hacer

    PORTB = Aleatorio

    Espera 1 s

Bucle
```

Leer AD

Sintaxis:

```
var = ReadAD (puerto)
```

Disponibilidad de comandos:

Disponible en todos los chips PIC con un módulo convertidor de analógico a digital incorporado.

Explicación:

ReadAD es una función que se puede usar para leer el convertidor analógico a digital incorporado que incluyen muchos chips de microcontroladores. puerto debe ser AN0, AN1, AN2, etc., hasta el número de entradas analógicas disponibles en el chip que está en uso. Consulte la hoja de datos del chip del microcontrolador para encontrar la cantidad de puertos disponibles. (Nota: es perfectamente aceptable usar ADCx en PIC).

Otra función, ReadAD10, es casi idéntica a ReadAD. La única diferencia es que devuelve un valor completo de 10 bits en una variable de palabra.

Sample Version - Do Not Copy

Machine Translated by Google

Ejemplo:

```
#chip 16F819, 8

#config osc = int


'Establecer la dirección del pin de entrada

Dir PORTA.0 Entrada


'Bucle para tomar lecturas hasta que la EEPROM esté llena

Para dirección actual = 0 a 255


    'Haz una lectura y regístrala

    EPWrite dirección actual, ReadAD(AN0)


    'Espere 10 minutos antes de obtener otra lectura

    Espera 10 minutos


Próximo
```

Sample Version - Do Not Copy

Leer tabla

Sintaxis:

ReadTable TableName, elemento, salida

Machine Translated by Google

Explicación:

El comando ReadTable se usa para leer información de las tablas de búsqueda.
TableName es el nombre de la tabla que se va a leer, Item es la línea de la tabla que se va a leer y Output es la variable en la que se escribe el valor recuperado.

El elemento es 1 para la primera línea de la tabla, 2 para la segunda y así sucesivamente. El elemento 0 da el tamaño de la tabla. Se debe tener cuidado para asegurarse de que no se indique al programa que lea más allá del final de la tabla, o se observarán efectos extraños.

El tipo de Salida debe coincidir con el tipo de datos almacenados en la tabla. Por ejemplo, si la tabla contiene valores de Word, la salida debe ser una variable de Word.
Si el tipo no coincide, GCBASIC intentará convertir el valor.

Ejemplo:

```
'Configuración de chips

#chip 16F88, 20



'Configuración de hardware

#define PUERTO LED.0

Dirección LED de salida



'Rutina principal

ReadTable TimesTwelve, 4, Temp

Establecer LED apagado

Si Temp = 48 Entonces encienda el LED

'Tabla de búsqueda llamada "TimesTwelve"

Tiempos De MesaDoce

12
```

Sample Version - Do Not Copy

Machine Translated by Google

24

36

48

60

72

84

96

108

120

132

144

Mesa final

Repetir

Sintaxis:

```
Repetir veces
...
código de programa
...
Finalizar repetición
```

Explicación:

El comando Repetir es ideal para situaciones en las que un fragmento de código debe ejecutarse un número determinado de veces. Utiliza menos memoria y se ejecuta más rápido que For

Sample Version - Do Not Copy

Machine Translated by Google

comando, y debe usarse donde no sea necesario contar cuántas veces se ha ejecutado el código.

Ejemplo:

```
'Este código hará parpadear una luz verde 6 veces.

#chip 16F88, 20

#define PUERTO LED.0
dir LED fuera

Repetir 6
    LED PulseOut, 1 s
    Espera 1 s
Finalizar repetición
```

Sample Version - Do Not Copy

Girar

Sintaxis:

Rotar variable {Izquierda | Derecha} [Simple]

Explicación:

El comando Rotar rotará la variable un bit en una dirección específica. El bit desplazado se colocará en el bit de acarreo del registro de estado (ESTADO.C).
STATUS.C actúa como un noveno bit de la variable que se está rotando.

Machine Translated by Google

Cuando una variable se gira a la derecha, el bit en la ubicación STATUS.C se coloca en el MSB de la variable que se gira, y el LSB de la variable se coloca en la ubicación STATUS.C.

Cuando se gira a la izquierda ocurre lo contrario. El MSB de la variable se cambia al bit STATUS.C y el LSB de la variable contendrá lo que estaba previamente en la ubicación del bit STATUS.C.

Esta tabla muestra el funcionamiento del comando Girar a la izquierda

Dominio	variable	ESTADO.C
Valores al inicio:	b'01110011'	0
Girar a la izquierda	b'11100110'	0
Girar a la izquierda de nuevo	b'11001100'	1
Girar a la izquierda por tercera vez	b'10011001'	1

Como puede notar, el bit STATUS.C agregó un 0 a la rotación. Entonces esto tomará 9 turnos para volver al valor original.

Opción SIMPLE

Muchas veces desea girar la variable como si el bit STATUS.C no estuviera allí, por lo que el MSB de la variable llena el LSB de la variable en Girar a la izquierda o el LSB llena el MSB en Girar a la derecha. Ahí es donde entra la opción SIMPLE. Agrega un paso oculto que cambia el bit STATUS.C dos veces para que el bit se mueva de un extremo de la variable al otro.

Dominio	variable	ESTADO.C
Valores al inicio:	b'01110011'	0
Girar a la izquierda	b'11100110'	0
Girar a la izquierda de nuevo	b'11001101'	1
Girar a la izquierda por tercera vez	b'10011011'	1

Sample Version - Do Not Copy

Machine Translated by Google

Ejemplo:

```
'Este programa usará Rotar para mostrar un LED perseguidor.
Se deben conectar 8 LED a PORTB, uno en cada pin.

#chip 16F819, 8 #config osc
= int

'Establecer la dirección del puerto

Dir PUERTO Salida

'Establece el estado inicial del puerto (bits 0 y 4 activados)

PUERTOB = b'00010001'

'Perseguir

Hacer
    Girar PORTB a la derecha Simple
    Espere 250ms
Bucle
```

Seleccionar

Sintaxis:

```
Seleccionar caso var

Valor del caso1
    codigo1
Valor del caso2
    codigo2
caso más
    codigo3
```

Sample Version - Do Not Copy

Machine Translated by Google

```
Finalizar Seleccionar
```

Explicación:

La estructura de control Select Case se usa para seleccionar y ejecutar una sección particular de código, según el valor de var. Si var es igual a valor1, entonces se ejecutará el código1. Una vez que se haya ejecutado el código 1, el chip saltará al comando Finalizar selección y continuará ejecutando el programa. Si ninguna de las otras condiciones es verdadera, se ejecutará el código de la sección Case Else.

Case Else es opcional y el programa funcionará correctamente sin él.

Si solo hay una línea de código después del caso, el código se verá mejor si la línea se coloca después del caso. Esto se muestra a continuación en el ejemplo, para los casos 3, 4 y 5.

Es importante tener en cuenta que solo se ejecutará una sección de código cuando se use Seleccionar mayúsculas y minúsculas.

Ejemplo:

```
'Programa para leer un valor de un potenciómetro y mostrar un

'diferente palabra basada en el resultado



#chip 18F4550, 20

'Configuración de conexión LCD

#definir LCD_IO 4

#define LCD_DB4 PUERTO.4

#define LCD_DB5 PUERTO.5

#define LCD_DB6 PUERTO.6

#define LCD_DB7 PUERTO.7

#define LCD_RS PUERTO.0
```

Sample Version - Do Not Copy

Machine Translated by Google

```
#define LCD_RW PUERTO.1

#define LCD_Habilitar PUERTO.2

DIR PORTA.0 EN

Hacer

    Temp = ReadAD(AN0) / 20

    CLS

    Seleccionar temperatura de la caja

        Caso 0

            Imprimir "¡Ninguno!"

        Caso 1

            Imprimir "Uno"

        Caso 2

            Imprimir "Dos"

        Caso 3: Imprimir "Tres"

        Caso 4: Imprimir "Cuatro"

        Caso 5: Imprimir "Cinco"

        caso más

            Imprimir "¡Mucho!"

    Finalizar Seleccionar

    Espere 250ms

Bucle
```

Sample Version - Do Not Copy

Machine Translated by Google

SerImprimir

Sintaxis:

Canal SerPrint, valor

Explicación:

SerPrint se utiliza para enviar un valor a través de la conexión en serie. El valor puede ser una cadena, una palabra o un byte: SerPrint es muy similar a Print. El canal es la conexión en serie para enviar datos.

SerPrint no enviará ningún carácter de nueva línea. Si el chip está enviando a una terminal, estos comandos deben seguir cada SerPrint:

Canal SerSend, 13

canal SerSend, 10

Ejemplo:

```
'Este programa mostrará los valores recibidos a través de la serie
'conexión. Si se recibe "pot", el valor del sensor analógico
'será enviado.

'Configuración de chips
#chip 18F2525, 8
#config Osc = Int

'Configuración de pantalla LCD
```

Sample Version - Do Not Copy

Machine Translated by Google

```
#definir LCD_IO 4

#define LCD_RS PORTC.7

#define LCD_RW PUERTO.6

#define LCD_Habilitar PUERTO.5

#define LCD_DB4 PORTC.4

#define LCD_DB5 PORTC.3

#define LCD_DB6 PUERTO.2

#define LCD_DB7 PUERTO.1


'Configuración de serie

#define SerInPort PORTB.6

#define SerOutPort PORTB.7


#define SendAHigh Desactivar SerOutPort

#define SendALow Activar SerOutPort

#define RecAHigh SerInPort Desactivado

#define RecALow SerInPort activado


'Potenciómetro

#define POT_PORT PORTA.0

#define POT_AN AN0


'Establecer direcciones de pines

Dir SerOutPort Salida

Dir SerInPort In
```

Sample Version - Do Not Copy

Machine Translated by Google

```
Dir POT_PORT Entrada


'Crear variables de búfer para almacenar mensajes recibidos
Búfer atenuado como cadena
Dim OldBuffer como cadena
Tamaño de búfer = 0

'Configurar conexión serie
InitSer 1, r9600, 1 + WaitForStart, 8, 1, ninguno, invertir

'Mostrar mensajes de prueba
Imprimir "Probador en serie"
Espera 1 s
SerPrint 1, "Prueba GCBASIC RS232"
SerSend 1, 13
SerSend 1, 10
Espera 1 s

'Bucle principal
Hacer
    'Obtener un byte de la terminal
    SerReceive 1, temperatura

    'Si se presionó la tecla Intro, tratar con el contenido del búfer
    Si temperatura = 13 Entonces
```

Sample Version - Do Not Copy

Machine Translated by Google

```
    Búfer(0) = Tamaño de Búfer



     'Intenta ejecutar comandos en el búfer
    Si no es ExecCommand (búfer), entonces
         'Mostrar mensaje en la línea inferior, último mensaje en la parte superior.
        CLS
        Imprimir búfer antiguo
        Localizar 1, 0
        Búfer de impresión
         'Guardar el mensaje para la próxima vez
        OldBuffer = Búfer
    Terminara si


    Tamaño de búfer = 0
Terminara si
'Código de retroceso, borra el último carácter en el búfer
Si temperatura = 8 Entonces
    Si BufferSize > 0 Entonces BufferSize -= 1
Terminara si
'Recibido código ASCII entre 32 y 127, agregar al búfer
Si Temp >= 32 y Temp <= 127 Entonces
    Tamaño del búfer += 1
    Búfer (Tamaño del búfer) = Temperatura
```

Sample Version - Do Not Copy

Machine Translated by Google

```
    Terminara si
Bucle

 'Toma una lectura del sensor y la envía a la terminal
Lectura del sensor de envío secundario
    SerPrint 1, "Lectura del sensor:"
    SerPrint 1, ReadAD10 (POT_AN)
    SerSend 1, 13
    SerSend 1, 10
Finalizar sub

 'Comprobará el búfer para un comando
'Si se encuentra el comando, ejecútelo y devuelva verdadero
'Si no, devuelve falso
Función ExecCommand (CmdIn como cadena)
    ComandoEjecutivo = Falso
     'Si el comando recibido es "pot", muestra el valor del potenciómetro
    Si CmdIn = "olla" Entonces
        EnviarSensorLectura
        ComandoEjecutivo = Verdadero
    Terminara si
función final
```

Sample Version - Do Not Copy

Machine Translated by Google

SerRecibir

Sintaxis:

SerReceive canal, salida

Explicación:

Este comando leerá un byte del canal RS232 dado por canal de acuerdo con las reglas establecidas usando InitSer, y almacenará el byte recibido en la salida variable.

Ejemplo:

```
'Este programa leerá un byte de PORTB.2 y encenderá el LED si
'el byte es más de 50. Esto se puede usar con SerSend
'programa de ejemplo.

#chip 16F88, 8
#config Osc = Int

#define RecAHigh PORTB.2 ENCENDIDO
#define RecALlow PORTB.2 APAGADO
#define PUERTO LED.0

Dir PORTB.0 Salida
Dir PORTB.2 Entrada
```

Sample Version - Do Not Copy

Machine Translated by Google

```
InitSer 1, r9600, 1 + WaitForStart, 8, 1, ninguno, normal
Hacer
    SerReceive 1, temperatura
    Si la temperatura <= 50, entonces apague el LED
    Si la temperatura es > 50, entonces encienda el LED
Bucle
```

SerEnviar

Sintaxis:

Canal SerSend, datos

Explicación:

Este comando enviará un byte dado por datos usando el canal RS232 denominado canal de acuerdo con las reglas establecidas usando InitSer.

Ejemplo:

```
'Este programa enviará un byte usando PORTB.2, cuyo valor
'depende de si se presiona un botón. Esto se puede usar con el 'ejemplo para SerReceive.

#chip 16F819, 8
#config Osc = Int
```

Sample Version - Do Not Copy

Machine Translated by Google

```
#define SendAHigh Activar PORTB.2

#define SendALlow Establecer PORTB.2 APAGADO

#define Botón PORTA.0

Botón de entrada de directorio

Dir PORTB.2 Salida

InitSer 1, r9600, 1+WaitForStart, 8, 1, ninguno, normal

Hacer

    Si Botón = Encendido Entonces Temp = 100

    Si Botón = Apagado Entonces Temp = 0

    SerSend 1, temporal

   Espera 100ms

Bucle
```

Colocar

Sintaxis:

```
Establecer variable.bit {Activado | Apagado}
```

Explicación:

El propósito del comando Set es activar y desactivar bits individuales. El comando Establecer es más útil para controlar los puertos de salida, pero también se puede usar para establecer variables.

A menudo, cuando se controlan los puertos de salida, Set se usa junto con constantes. Esto facilita la adaptación posterior del programa para un nuevo circuito.

Sample Version - Do Not Copy

Machine Translated by Google

Ejemplo:

```
'Programa de ejemplo de parpadeo LED para GCBASIC

'Controla un LED en PORTB bit 0.



'Establece el modelo de chip y las opciones de configuración

#chip 16F84A, 20



'Establecer una constante para representar el puerto de salida

#define PUERTO LED.0



'Establece la dirección del pin

Dirección LED de salida



'Rutina principal

Hacer

    Establecer LED encendido

    Espera 1 segundo

    Establecer LED APAGADO

    Espera 1 segundo

Bucle
```

Sample Version - Do Not Copy

Machine Translated by Google

tono corto

Sintaxis:

Frecuencia de tono corto, duración

Explicación:

Este comando producirá el tono especificado durante la duración especificada.
La frecuencia se mide en unidades de 10 Hz y la duración en unidades de 1 ms.

Tenga en cuenta que este comando puede no producir la frecuencia exacta especificada.
Si bien es lo suficientemente preciso para pitidos de error y pequeñas piezas de
música monofónica, no debe usarse para nada que requiera una frecuencia muy precisa.

Ejemplo:

```
'Programa de muestra para producir un tono en PORTB bit 1, basado en el
'lectura de un LDR en AN0 (generalmente PORTA bit 0).

#chip 16F88, 20
#define SoundOut PUERTO.1

Dir PORTA.0 Entrada

Hacer
    Lectura de tono cortoAD(AN0), 100
Bucle
```

Sample Version - Do Not Copy

Machine Translated by Google

Modo SPI

Sintaxis:

Modo SPIMode

Disponibilidad de comandos:

Disponible en microcontroladores PIC con módulo Synchronous Serial Port (SSP).

Explicación:

SPIMode establece el modo del módulo SPI dentro del chip PIC. Estos son los posibles Modos SPI:

Nombre del modo	Descripción
maestrolento	Modo maestro, el reloj SPI es 1/64 de la frecuencia del PIC.
Maestro	Modo maestro, el reloj SPI es 1/16 de la frecuencia del PIC.
MasterFast	Modo maestro, el reloj SPI es 1/4 de la frecuencia del PIC.
Esclavo	Modo esclavo
Esclavo	Modo esclavo, con el pin de selección de esclavo habilitado.

Sample Version - Do Not Copy

Machine Translated by Google

Transferencia SPI

Sintaxis:

```
Transferencia SPI tx, rx
```

Disponibilidad de comandos:

Disponible en microcontroladores PIC con módulo Synchronous Serial Port (SSP).

Explicación:

Este comando envía y recibe simultáneamente un byte de datos utilizando el protocolo SPI. Se comporta de manera diferente dependiendo de si el PIC se ha configurado para actuar como maestro o esclavo.

Al operar como maestro, SPITransfer iniciará una transferencia. Los datos en tx se enviarán al esclavo, mientras que el byte que se almacena en el búfer en el esclavo se leerá en rx.

En modo esclavo, el comando SPITransfer pausará el programa hasta que el maestro inicie una transferencia. En este punto, enviará los datos en tx mientras lee la transmisión del maestro en la variable rx.

Ejemplo:

Hay dos programas de ejemplo para este comando: uno para ejecutar en el PIC esclavo y otro en el maestro. Se toma una lectura de un sensor en el esclavo y se envía al maestro, que muestra los datos en su pantalla LCD.

Programa de esclavos:

Sample Version - Do Not Copy

Machine Translated by Google

```
'Seleccionar modelo de chip y configuración
#chip 16F88, 20
#config MCLR_OFF

'Establecer direcciones de pines SPI
dir PORTB.2 fuera
dir PORTB.1 en
dir PORTB.4 en
'Establece la dirección del pin analógico
dir PORTA.0 en

'Establece el modo SPI en esclavo
Esclavo de modo SPI

'Permitir que otro PIC inicialice LCD
Espera 1 segundo

'Bucle principal: toma una lectura y luego espera para enviarla.
hacer
'Tenga en cuenta que rx es 0; esto se debe a que no se recibirán datos.
Lectura de transferencia SPIAD(AN0), 0
bucle
```

Programa maestro:

```
'Configuración general de hardware
```

Sample Version - Do Not Copy

Machine Translated by Google

```
#chip 16F877A, 20


'Configuración de conexión LCD
#definir LCD_IO 8
#define LCD_DATA_PORT PUERTO
#define LCD_RS PUERTO.0
#define LCD_RW PUERTO.1
#define LCD_Habilitar PUERTO.2


'Establecer direcciones de pines SPI
dir PORTC.5 fuera
dir PORTC.4 en
dir PORTC.3 fuera


'Establece el modo SPI en maestro, con reloj rápido
SPIMode MasterFast


'Bucle principal
hacer
'Lee un byte del esclavo
'No hay datos para enviar, entonces tx es 0
Transferencia SPI 0, temperatura


'Mostrar datos
si temperatura > 0 entonces
```

Sample Version - Do Not Copy

Machine Translated by Google

```
  CLS
  Imprimir "Luz:"
  LCDInt Temp
  temperatura = 0
terminara si

'Espere para dar tiempo a que la pantalla LCD muestre el valor dado
espera 100ms
bucle
```

Temporizador de inicio

Sintaxis:

```
Temporizador de inicio Temporizador No
```

Disponibilidad de comandos:

Disponible en todos los microcontroladores PIC con módulos de temporizador integrados.

Explicación:

StartTimer se utiliza para iniciar el temporizador especificado. Tenga en cuenta que no se puede utilizar para iniciar el temporizador 0 en un PIC; este temporizador siempre se ejecuta.

Sample Version - Do Not Copy

Machine Translated by Google

StopTimer

Sintaxis:

StopTimer TimerNo

Disponibilidad de comandos:

Disponible en todos los microcontroladores PIC con módulos de temporizador integrados.

Explicación:

StopTimer se utiliza para detener el temporizador especificado. Tenga en cuenta que no se puede usar para detener el temporizador 0 en un PIC; este temporizador siempre se ejecuta.

Ejemplo:

Consulte el artículo InitTimer1 para ver un ejemplo.

Tono

Sintaxis:

Frecuencia de tono, duración

Explicación:

Este comando producirá el tono especificado durante la duración especificada.
La frecuencia se mide en Hz y la duración en unidades de 10 ms.

Sample Version - Do Not Copy

Machine Translated by Google

Tenga en cuenta que este comando puede no producir la frecuencia exacta especificada.
Si bien es lo suficientemente preciso para pitidos de error y pequeñas piezas de música monofónica, no debe usarse para nada que requiera una frecuencia muy precisa.

Ejemplo:

```
'Programa de muestra para producir una nota A constante (440 Hz)
'en PORTB bit 1.
#chip 16F877A, 20
#define SoundOut PUERTO.1
Hacer
    Tono 440, 1000
Bucle
```

Esperar

Sintaxis:

Retraso de longitud fija: unidades de tiempo de espera

Retraso condicional: Espera {Mientras | Hasta} condición

Explicación:

El comando Esperar hará que el programa espere una cantidad específica de tiempo (como 1 segundo), o mientras/hasta que una condición sea verdadera.

Cuando se utiliza el retraso de longitud fija, hay una variedad de unidades disponibles:

Unidad	Longitud de la unidad	Rango de retraso

Sample Version - Do Not Copy

Machine Translated by Google

a nosotros	1 microsegundo	1 us - 65535 us
10us	10 microsegundos	10 us - 2,55 ms
EM	1 milisegundo	1ms - 65535ms
10ms	10 milisegundos	10 ms - 2,55 s
s	1 segundo	1 s - 255 s
metro	1 minuto	1 minuto - 255 minutos
h	1 hora	1 hora - 255 horas

En una etapa, las variables GCBASIC no podían contener más de 255. Las unidades de 10us y 10ms se agregaron como una forma de evitar este límite. Ahora no existe tal límite (esperar 1000 ms funcionará, por ejemplo), por lo que estos no son realmente necesarios.
Sin embargo, puede verlos en algunos ejemplos o programas más antiguos, y las unidades de 10us son a veces el retraso más corto que funcionará con precisión.

Ejemplo:

```
'Este código esperará hasta que se presione un botón, luego parpadeará

'una luz cada medio segundo y producir un tono de 440 Hz.



#chip 16F819, 8

#config osc = int



#define BOTÓN PORTB.0

#define PUERTO DE ALTAVOZ.1

#define PUERTO DE LUZ.2

Dir BOTÓN Entrada

Dir ALTAVOZ Salida

Dir LUZ Fuera
```

Sample Version - Do Not Copy

Machine Translated by Google

```
'Supone que el botón se enciende cuando se presiona
Espere hasta que BOTÓN = 1
Esperar hasta BOTÓN = 0

Hacer
    'Enciende la luz
    Encender LUZ
    Espere 500ms
    Desactivar LUZ

    'Producir el tono
    '440 Hz = 880 cambios = tono activado durante 1,14 ms
    Repetir 440
        ALTAVOZ PulseOut, 1140 us
        Espera 114 10us' Espera 114 x 10us (1.14ms)
    Finalizar repetición
Bucle
```

Sample Version - Do Not Copy

Machine Translated by Google

Sample Version - Do Not Copy

Machine Translated by Google

6.0 Proyectos de muestra Aquí

hay algunos proyectos de muestra para ayudarlo a comenzar con Great Cow Basic CHIPINO. Están diseñados para funcionar con el Demo-Shield de CHIPINO, pero puede construir los circuitos por separado si sabe cómo hacerlo.



Escudo de demostración CHIPINO

Sample Version Do Not Copy

Proyecto 1 - Cruce de trenes Hacer

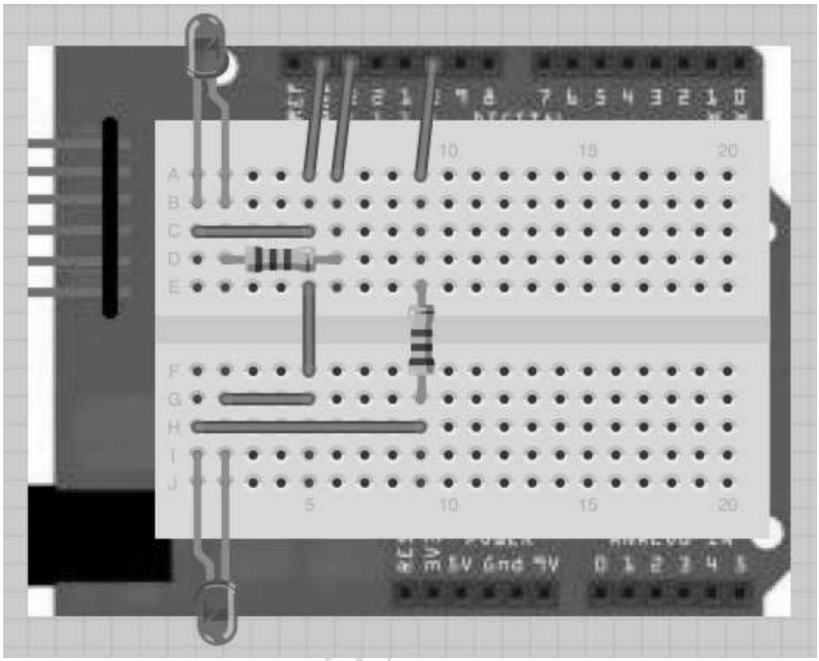
parpadear un LED es un excelente lugar para comenzar, pero démosle un uso práctico. ¿Alguna vez se detuvo en un cruce de trenes y vio las luces rojas parpadear de un lado a otro para indicar que se acerca un tren? Esto se puede lograr muy fácilmente con dos LED. Este proyecto también se puede incorporar en un modelo de señal de cruce de ferrocarril de tren para hacerlo un poco más realista.

Hardware

Hay dos LED para controlar. Los LED se activan mediante una señal alta en el pin digital que los activa. Un bajo los apaga. Se coloca una resistencia en serie con los LED para limitar la corriente. Una resistencia de 1k ohm funciona bien.

El LED izquierdo está conectado al pin digital número 13 en el CHIPINO. El LED derecho está conectado al pin 10. Se usa una resistencia en serie para limitar la corriente y se puede usar cualquier valor de 220 ohmios a 1k ohmios. Ambos LED están rojos en el escudo de demostración de CHIPINO.

Machine Translated by Google



Dibujado con Fritzing.org

Figura 6-1: Proyecto de cruce de trenes

Gratuite Version

Software

'Train.gcb'
Un programa para hacer parpadear dos LED
rojos como un 'cruce de tren en los pines
digitales 10 y 13' (LED rojos en Demo-Shield)

```
'Chip modelo
#chip 16F886, 4 #incluir
<chipino.h>                    'Define la configuración de CHIPINO

'Rutina principal
Comenzar:
'Encienda un LED, el otro apagado establezca D10
 encendido 'LED izquierdo encendido establezca
 D13 apagado 'LED derecho apagado espere 1
 segundo
```

Machine Translated by Google

```
'Ahora cambia los LED
  Establecer D10 APAGADO          'LED izquierdo apagado
  Establezca D13 en 'LED derecho en
  espera 1 segundo

'Volver al inicio del programa ir a Inicio
```

Cómo funciona La

primera sección es solo un encabezado de comentario ,para describir el proyecto. Cada línea es una línea de comentario ya que comienza con un apóstrofo.

```
'Train.gcb' Un
programa para hacer parpadear dos LED rojos como un
'cruce de tren en los pines digitales 10 y 13' (LED rojos en
Demo-Shield)
```

Se selecciona el PIC16F886 en el CHIPINO y luego un archivo de cabecera con todos los nicknames D0-D13 y A0-A5.

```
'Chip modelo
#chip 16F886, 4 #incluir
<chipino.h>                       'Define la configuración de CHIPINO
```

El bucle principal controla los dos LED separados mediante el control de los pines digitales a los que están conectados. Si se desactiva el pin D13 y se enciende el pin D10, se enciende el LED izquierdo. Un comando de espera de 1 segundo crea la tasa de parpadeo. La segunda sección hace lo contrario y enciende el LED derecho en el pin D13 y apaga el LED izquierdo en el pin D10. Se añade otra espera de 1 segundo.

```
'Rutina principal
Comenzar:
  'Encienda un LED, el otro apagado establezca D10
  encendido 'LED izquierdo encendido establezca
  D13 apagado 'LED derecho apagado espere 1
  segundo

'Ahora cambia los LED
```

Sample Version - Do Not Copy

Machine Translated by Google

```
Establecer D10 APAGADO          'LED izquierdo apagado
Establezca D13 en 'LED derecho en
espera 1 segundo
```

El programa permanece en un bucle continuo con la línea de comando goto Start creando una iluminación alterna de los LED para crear la pantalla de cruce de tren.

```
'Volver al inicio del programa ir a Inicio
```

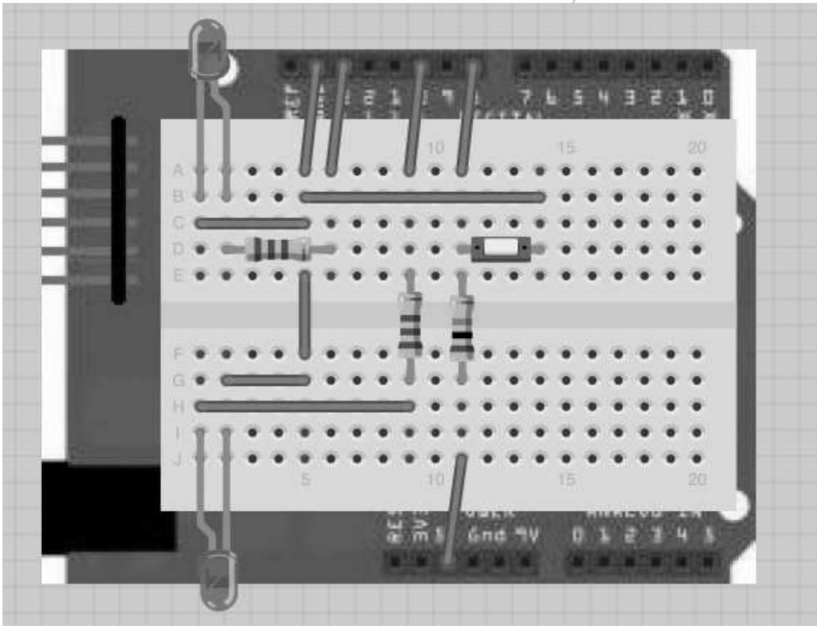
Sample Version - Do Not Copy

Machine Translated by Google

Proyecto 2 - Detección de un interruptor

En muchos proyectos, necesitará algún tipo de interfaz humana para controlar la operación. Un interruptor de botón momentáneo es una forma muy común de hacerlo. Puede iniciar y detener la operación o puede acelerar o ralentizar lo que controla el microcontrolador. Para hacer eso, el software necesita reconocer que se presionó un interruptor. Este proyecto muestra un método simple para detectar un interruptor de botón momentáneo.

El software tendrá que monitorear el interruptor continuamente como parte del bucle principal de código y luego responder. En este proyecto, el software encenderá un LED hasta que se presione el interruptor, momento en el que se apagará el LED y se encenderá un segundo LED. Mientras se presione el interruptor, el primer LED permanecerá apagado y el segundo LED permanecerá encendido. Tan pronto como se suelte el interruptor, el primer LED volverá a encenderse y el segundo LED se apagará. El proyecto completo se muestra en la Figura 6-2.



Dibujado con Fritzing.org

Figura 6-2: Control de interruptores

Machine Translated by Google

Hardware El

hardware usa las mismas dos conexiones LED rojas que el proyecto de cruce de trenes en el Capítulo 3. La adición del interruptor se muestra en la Figura 6-2. El interruptor está cableado como un interruptor del lado bajo, lo que significa que el circuito tiene una resistencia pull-up de 5 voltios, por lo que la entrada del micro es alta cuando no se presiona el interruptor y baja cuando se presiona el interruptor. Esto se conoce como interruptor lateral bajo. Si las partes estuvieran invertidas y el interruptor estuviera conectado a 5 voltios y la resistencia a tierra, entonces sería un interruptor del lado alto.

El software cambiará el pin 8 a una entrada y luego probará el pin 8 para ver si cambia a bajo, lo que indica que se presionó el interruptor. Los LED están conectados al pin 10 y al pin 13 a través de resistencias de 1k.

Software

```
'switches.gcb'
Supervisa dos interruptores y enciende el LED cuando se presiona.
'Otro LED encendido cuando está
inactivo 'Configuración para Demo-
Shield 'El interruptor D8 controla el LED D10

'Configuración de
chip #chip 16F886,4
#incluye <chipino.h> DIR D8
en 'Establece el pin del interruptor D8 en la entrada DIR D10
out 'Establece el pin LED D10 en la salida DIR D13 out
'Establece el pin LED D13 en la salida

Comenzar:
Si D8=0 Entonces 'Si se presiona el interruptor izquierdo,
  configure D10 encendido 'D10 LED encendido configure
  D13 apagado 'D13 LED apagado
De lo
  contrario, configure D13 encendido 'D13 LED
  encendido configure D10 apagado 'D10 LED apagado
Terminara si
Ir a Inicio
```

Cómo funciona La

primera sección es solo un encabezado de comentario para describir el proyecto. Cada línea es una línea de comentario ya que comienza con un apóstrofo.

```
'switches.gcb
'Supervisa dos interruptores y enciende el LED cuando se presiona.
```

Sample Version - Do Not Copy

Machine Translated by Google

'Otro LED encendido cuando está inactivo

'Configuración para Demo-Shield
El interruptor 'D8 controla el LED D10

Se selecciona el PIC16F886 en el CHIPINO y luego un archivo de cabecera con todos los nicknames D0-D13 y A0-A5.

'Chip modelo
#chip 16F886, 4 #incluir
<chipino.h> 'Define la configuración de CHIPINO

Se define la dirección del puerto para cada pin. Los LED no necesitan esto ya que el compilador GCB establece automáticamente los pines en las salidas, pero lo agregué aquí para mostrar cómo configurar un pin de entrada y salida.

DIR D8 en 'Establece el pin del interruptor D8 en la entrada
DIR D10 out 'Establece el pin LED D10 en la salida
DIR D13 out 'Establece el pin LED D13 en la salida

Ahora entramos en el bucle principal. Se utiliza un comando If Else Endif.
La primera sección prueba si el pin D8 está bajo, lo que indica que se presionó el interruptor.
Si es bajo, el LED D10 se enciende y el D13 se apaga. Si el pin D8 no está bajo, entonces se realiza la sección Else y se enciende el LED D13 y se apaga D10. End if indica el final del comando If Else.

Comenzar:
Si D8=0 Entonces 'Si se presiona el interruptor izquierdo,
configure D10 encendido 'D10 LED encendido configure
D13 apagado 'D13 LED apagado
De lo
contrario, configure D13 encendido 'D13 LED
encendido configure D10 apagado 'D10 LED apagado
Terminara si

Una línea de comando Goto Start regresa a la etiqueta Start para hacer que el programa monitoree continuamente la entrada del interruptor.

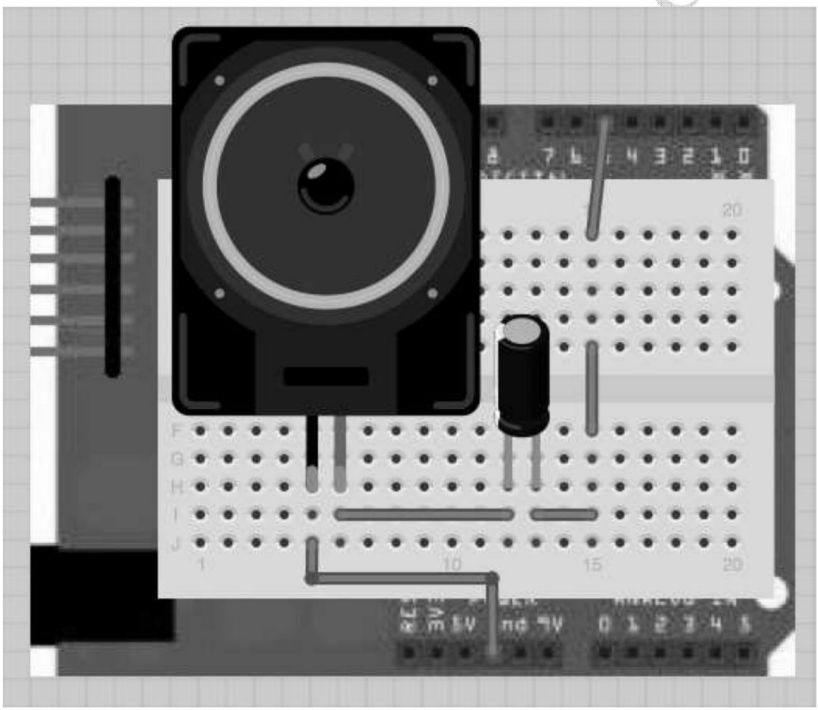
Ir a Inicio

Sample Version - Do Not Copy

Machine Translated by Google

Proyecto 3 – Creación de sonido

Crear sonido a través de un altavoz puede ser muy útil para muchos proyectos. Puede crear un sonido cuando se presiona un interruptor o hacer sonar una alarma cuando baja la temperatura. En este proyecto, te mostraré cómo hacer un tono a través de un altavoz usando un pin digital. Seleccioné un pin PWM porque el escudo de demostración está cableado de esa manera, pero esto se puede ejecutar en cualquier pin digital. El comando Tono en Great Cow Basic se usará para hacer que este sea un programa muy corto de escribir. Este proyecto creará el sonido de un objeto que cae similar al que podría escuchar en un videojuego antiguo.



Dibujado con Fritzing.org

Figura 6-3: Creación de un tono

Machine Translated by Google

Hardware El

pin CHIPINO D5 producirá una onda cuadrada porque es un pin digital que controla el altavoz. La onda cuadrada se puede convertir en una señal semirredondeada para que funcione mejor con el parlante colocando un capacitor de 10 uf en serie entre el pin 5 y el cable positivo del parlante. Luego se conecta a tierra el otro lado del altavoz.

Un altavoz de 8 ohmios funciona bien, pero un enfoque más común es un altavoz piezoeléctrico. Un altavoz piezoeléctrico de Jameco.com con el número de pieza DBX05-PN funcionará bien.

La configuración se muestra en la Figura 6-3. El lado derecho del capacitor es el lado positivo y se conecta al pin digital D5. El altavoz positivo también está en el lado derecho y se conecta al lado negativo del capacitor. Luego, el altavoz se conecta a tierra al pin de tierra en el cabezal de alimentación CHIPINO.

Software

```
'tone.gcb
'Este programa produce un sonido de caída 'en
el pin digital 5 conectado al altavoz piezoeléctrico
en Demo-Shield.

#chip 16F886, 4
#incluye <chipino.h> DIM
SND como palabra 'Crear una palabra variable

#define SoundOut D5
Comenzar:
Para SND = 2000 a 200 paso -5
    Tono SND, 1' Tono con 10 mseg de retraso

Siguiente ir a empezar
```

Cómo funciona La

primera sección es solo un encabezado de comentario para describir el proyecto. Cada línea es una línea de comentario ya que comienza con un apóstrofo.

```
'tone.gcb
'Este programa produce un sonido de caída 'en
el pin digital 5 conectado al altavoz piezoeléctrico
en Demo-Shield.
```

Se selecciona el PIC16F886 en el CHIPINO y luego un archivo de cabecera con todos los nicknames D0-D13 y A0-A5.

Sample Version - Do Not Copy

Machine Translated by Google

```
#chip 16F886, 4
#incluir <chipino.h>
```

El comando de tono utilizará un valor superior a 255, por lo que debemos definir una variable de palabra que llamamos SND (abreviatura de sonido). Las variables de palabra contendrán un número de hasta 65.535. GCB crea automáticamente variables de byte cuando usa una variable, pero para un tamaño mayor que un byte, debe definirlo usando la línea de comando DIM.

```
DIM SND como palabra 'Crear una variable de palabra
```

Ahora entramos en el bucle principal. Se utiliza un comando For Next para crear el bucle de producción de sonido. Comienza configurando la variable SND en 2000 y luego reduce en 5 cada ciclo hasta que SND sea igual a 200. Luego, se usa el comando Tone para producir el tono en el altavoz. Produce una frecuencia en el altavoz igual al valor de la variable SND. Comienza con un tono de 2000 hercios y baja lentamente a 200 hercios. La velocidad se establece según la duración de cada tono y luego se controla mediante el valor en la línea de comando Tono. En este caso, está configurado para producir un tono durante 10 milisegundos cada uno.

```
Comenzar:
Para SND = 2000 a 200 paso -5
    Tono SND, 1' Tono con 10 mseg de retraso
Próximo
```

El bucle For Next se repite automáticamente, pero agregamos un Goto Start en la parte inferior en caso de que el software se pierda y tenga que volver al principio.

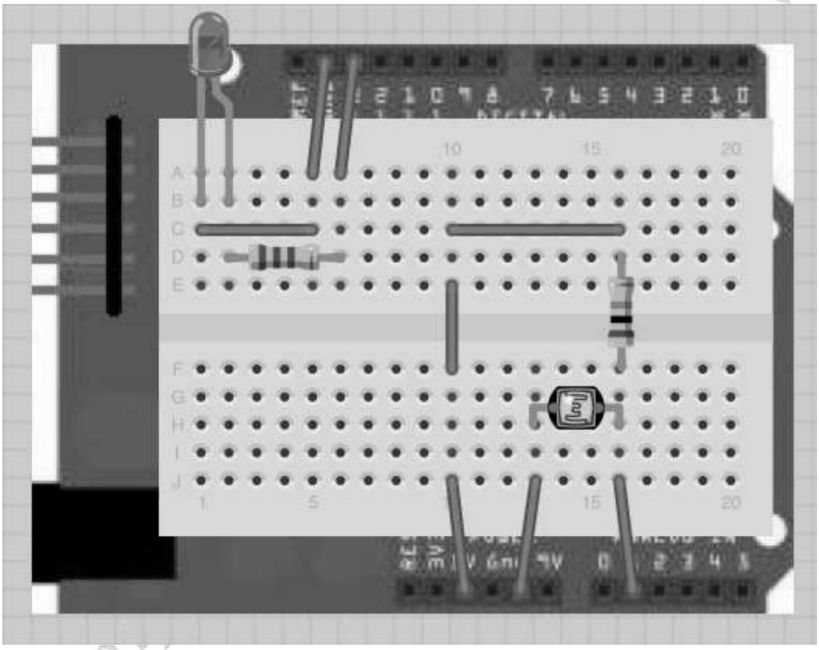
Ir a Inicio

Sample Version - Do Not Copy

Machine Translated by Google

Proyecto 4 - Detección de luz

Podemos usar una fotorresistencia para detectar la luz usando un pin analógico. Una fotorresistencia cambia de resistencia a medida que se expone a la luz. Al colocar una resistencia pull-up conectada a 5v en serie con la fotorresistencia, el cambio de luz creará un voltaje variable. El voltaje variable puede entonces ser monitoreado por un pin analógico en el CHIPINO. En este caso, solo encenderemos un LED cuando esté oscuro y lo apagaremos cuando haya luz.



Dibujado con Fritzing.org

Figura 6-4: Creación de un tono

Hardware

Un lado del sensor de luz está conectado a tierra. El otro se conecta a la resistencia y también al pin analógico 1. La resistencia pull-up es de 10k conectada a 5v.

Es posible que deba ajustar la resistencia de su sensor de luz, pero en la mayoría de los casos, una de 10k funcionará bien. En la oscuridad, la fotorresistencia tiene una resistencia alta y en la luz tiene una resistencia baja. Por lo tanto, en la oscuridad, el voltaje será alto y en la luz, el voltaje será bajo.

Machine Translated by Google

El LED controlado por el sensor se conecta al pin 10 del puerto digital. Una resistencia en serie de 220 ohmios conectada al ánodo y el cátodo conectado a tierra completan el circuito.

Software

```
'light.gcb'
Este programa monitorea un fotorresistor conectado
  al pin AN1. Si está en la oscuridad, encenderá el
LED en D13. Si está en la luz, entonces el LED está
apagado.
'Escrito para CHIPINO Demo-Shield.

'Configuración de
chip #chip 16F886, 4
#incluir <chipino.h> Dir A1
In 'Hacer entrada analógica de 1 pin Dir D13 out
'Hacer salida de pin D15

Comenzar:

Hacer

'Luz LED si está en la oscuridad
Si ReadAD(A1) < 150, configure D13
  en 'El sensor está oscuro; de lo
  contrario, configure D13 en 'Luz
  brillante en el extremo del sensor si
```

Bucle

Cómo funciona La

primera sección es solo un encabezado de comentario para describir el proyecto. Cada línea es una línea de comentario ya que comienza con un apóstrofo.

```
'light.gcb'
Este programa monitorea un fotorresistor conectado
  al pin AN1. Si está en la oscuridad, encenderá el
LED en D13. Si está en la luz, entonces el LED está
apagado.
'Escrito para CHIPINO Demo-Shield.
```

Se selecciona el PIC16F886 en el CHIPINO y luego un archivo de cabecera con todos los nicknames D0-D13 y A0-A5.

```
#chip 16F886, 4
#incluir <chipino.h>
```

Sample Version - Do Not Copy

Machine Translated by Google

El pin analógico A1 se establece en una entrada para leer el voltaje del sensor de luz.
La dirección del puerto para el pin D13 también se define como una salida. Los pines digitales no necesitan esto ya que el compilador GCB establece automáticamente los pines en las salidas, pero lo agregué aquí para mostrar cómo configurar el pin de salida si desea definirlo.

```
Dir A1 In 'Hacer entrada analógica de 1 pin
Dir D13 out 'Hacer salida de pin D15
```

Ahora entramos en el bucle principal. En este caso, es un comando Do Loop real en lugar del Lable Goto Lable que he usado anteriormente. El Do Loop ejecutará todo lo que esté entre Do y Loop continuamente. Agregué una etiqueta Inicio: solo para indicar dónde comienza el ciclo principal, pero realmente no tiene ningún propósito para el programa.

El comando ReadAD se usa para leer el voltaje en el pin A1 del CHIPINO. Utilizará el convertidor analógico a digital (ADC) interno integrado en los pines analógicos del CHIPINO para convertir el voltaje en A1 a un valor de 0 a 255.
Luego podemos comparar eso con un valor, en este caso 150, para determinar si el LED debe encenderse o no.

Si el valor convertido por el comando ReadAD es inferior a 150 (oscuro), entonces se enciende el LED D13. Si el valor es superior a 150 (luz), el LED D13 se apaga.

```
Comenzar:
Hacer
'Luz LED si está en la oscuridad
Si ReadAD(A1) < 150, configure D13
  en 'El sensor está oscuro; de lo
  contrario, configure D13 en 'Luz
  brillante en el extremo del sensor si

Bucle
```

Sample Version - Do Not Copy

Machine Translated by Google

Sample Version - Do Not Copy

Machine Translated by Google

Apéndice A: características de la GUI de PICkit 2

El programador PICkit 2 de Microchip es un programador de hardware de código abierto.
El cable de programación USB CHIPINO básico de Great Cow es en realidad un programador PICkit 2 reducido a un cable. El cable de programación PICkit 2 / USB puede hacer más que programar el módulo CHIPINO. También puede recibir señales para un programa de terminal llamado UART Tool y mostrar señales en modo digital o en un modo gráfico de analizador lógico con Logic Tool. Este apéndice cubrirá esas características.

Puede iniciar la GUI de PICkit 2 desde la barra de herramientas del IDE de GCB haciendo clic en el icono de PICkit2.



Nota: Al usar las características de la GUI de PICkit 2, no puede programar el CHIPINO usando la aplicación de programador PICkit 2 integrada asociada con el icono de programa y compilación de la flecha azul. Deberá apagar la GUI de PICkit 2, luego reprogramar el CHIPINO y luego reiniciar la GUI de PICkit 2.

Herramienta

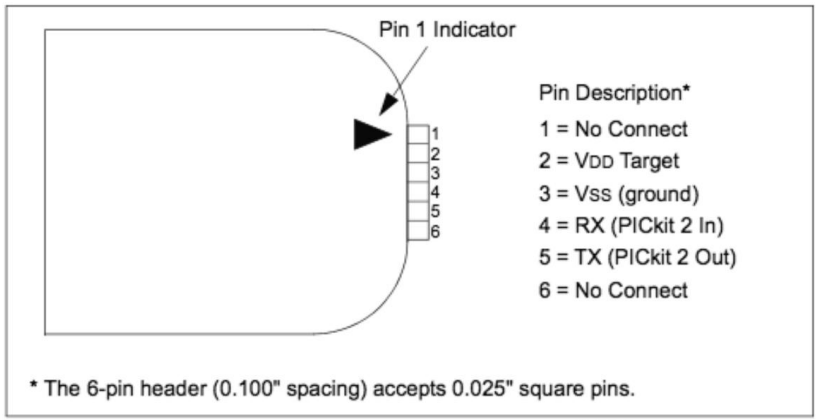
UART La GUI del programador PICkit 2 incluye la función de herramienta UART. Esta característica permite que el cable CHIPINO USB Programmer se utilice como una interfaz de terminal UART serial para comunicarse con un microcontrolador PIC. Los usos potenciales incluyen:

- Mostrar la salida de texto de depuración del microcontrolador
- Registrar los datos del microcontrolador en un archivo de texto
- Desarrollar y depurar una interfaz UART del microcontrolador
- Interactuar con el microcontrolador y enviarle comandos durante el desarrollo

La herramienta admite comunicaciones en serie asíncronas de dúplex completo de 150 a 38400 baudios, incluidas velocidades de baudios no estándar personalizadas. El PICkit 2 se conecta a través del conector CHIPINO ICSP directamente al pin RB7 (PICkit 2 RX) y al pin RB6 (PICkit 2 TX) del microcontrolador en niveles lógicos. No se necesita transceptor.

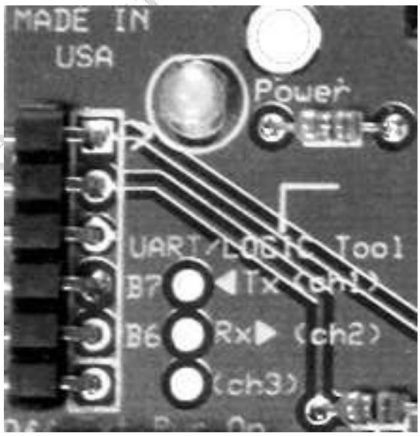
Sample Version - Do Not Copy

Machine Translated by Google



Conexión de la herramienta UART

Los pines CHIPINO B6 y B7 no están conectados a los encabezados. Estos pines tienen orificios ubicados cerca de la conexión de programación. Puede soldar un encabezado de tres pines allí para facilitar la conexión. Cualquier pin a través del cual desee enviar datos se puede conectar al pin B7 para enviar datos al software de terminal PICkit 2. Cualquier pin que desee recibir datos del terminal PICkit 2 se puede conectar al pin B6.



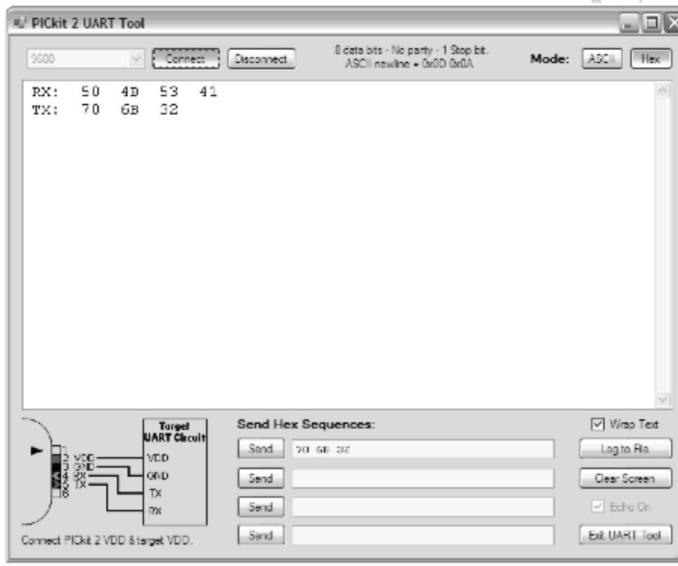
Sin embargo, una forma más fácil es enrutar esas señales a través del software a los pines B7 y B6. De esta manera, tiene acceso completo a los pines de encabezado CHIPINO para señales y usa B7 y B6 para la comunicación.

Machine Translated by Google

Ventana de herramientas UART

Para abrir la ventana de la herramienta PICkit 2 UART, seleccione Herramientas>Herramienta UART. Se abrirá la ventana del programador PICkit 2 y se cerrará la ventana de la aplicación del programador PICkit 2, ya que el programador no se puede utilizar mientras la herramienta UART está activa y viceversa.

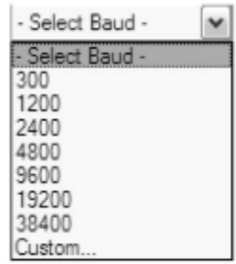
La herramienta UART tiene dos modos: ASCII y HEX. El modo actual se selecciona con los botones en la parte superior derecha de la pantalla. El botón correspondiente al modo activo se mostrará presionado. La selección de modo afecta la forma en que se muestran e ingresan los datos en serie en la ventana, y los modos se explican en las siguientes dos secciones.



Configuración de la velocidad en baudios y conexión La

velocidad en baudios se puede seleccionar desde el cuadro combinado en la esquina superior izquierda de la ventana de la herramienta UART. Se enumeran varias tasas de baudios comunes y, al seleccionar Personalizado, se puede usar cualquier tasa de baudios de 150 a 38400 en incrementos de 1 baudio.

Machine Translated by Google



Modo ASCII

Cuando se selecciona el modo ASCII, los bytes en serie recibidos del CHIPINO se muestran como caracteres ASCII en la pantalla de la terminal de la ventana principal. Todos los bytes se muestran consecutivamente. Para mostrar una nueva línea, el CHIPINO debe transmitir los valores de carácter 0x0D (retorno de carro) y 0x0A (avance de línea) en secuencia. Si se envía uno de estos valores, o se envían en orden inverso, los caracteres aparecerán como un carácter no imprimible (cuadro) en la pantalla.

Los bytes se pueden transmitir de tres maneras:

1. Haga clic en la pantalla del terminal para seleccionarlo. Cualquier carácter escrito en el teclado de la PC se transmitirá inmediatamente fuera de PICkit 2.
2. Haga clic con el botón derecho en la pantalla del terminal y seleccione Pegar en el menú emergente para pegar cualquier texto copiado o cortado anterior. También se pueden usar <Ctrl> + <v>. Cualquier dato pegado se transmitirá inmediatamente fuera de PICkit 2.
3. Utilice las cadenas de macros en la parte inferior de la ventana.

Las cadenas de macros permiten introducir hasta cuatro cadenas de caracteres. Cada cadena puede tener hasta 60 caracteres. Cuando se hace clic en Enviar, la cadena completa se transmitirá en secuencia desde el PICkit 2. Las cadenas de cadenas de macros se guardan incluso cuando la herramienta UART no está en uso.

Cuando está marcada, la casilla de verificación Agregar CR+LF sobre los cuadros de macro de cadena transmitirá automáticamente los caracteres de retorno de carro (0x0D) y avance de línea (0x0A) al final de una cadena. Cuando no está marcada, no se transmite nada después de los caracteres de la cadena.

No se mostrarán caracteres transmitidos si la casilla de verificación Echo On no está marcada. Marque esta casilla para mostrar los caracteres transmitidos junto con los caracteres recibidos en la pantalla del terminal.

El texto en la pantalla del terminal se puede seleccionar (resaltar) con el cursor y copiar en el portapapeles haciendo clic derecho y seleccionando Copiar o presionando <Ctrl> + <c>.

Sample Version 4 - Do Not Copy

Machine Translated by Google

Modo

hexadecimal El modo hexadecimal de la herramienta UART muestra los valores hexadecimales de los bytes recibidos del CHIPINO en la pantalla del terminal. Una línea de bytes recibida por la herramienta UART está precedida por el texto "RX:". Una línea de bytes transmitida por la herramienta UART está precedida por el texto "TX:".

En el modo hexadecimal, los bytes solo se pueden transmitir de una manera: escribiendo una secuencia de uno o más valores hexadecimales en uno de los cuatro cuadros Enviar secuencias hexadecimales:. Para enviar una secuencia, haga clic en Enviar junto a ella. Una secuencia hexadecimal puede contener de 1 a 48 bytes.

La casilla de verificación Echo On no tiene ninguna función en el modo hexadecimal.

Los datos en la pantalla del terminal se pueden seleccionar (resaltar) con el cursor y copiar en el portapapeles haciendo clic derecho y seleccionando Copiar o presionando <Ctrl> + <c>.

Ajustar texto

Esta casilla de verificación afecta a ambos modos, ASCII y HEX. Cuando está marcada, una línea mostrada se ajustará automáticamente en el borde derecho de la pantalla del terminal. Si no está marcada, la línea recibida no se ajustará y aparecerá una barra de desplazamiento horizontal en la pantalla del terminal para permitir que se vea la línea completa.

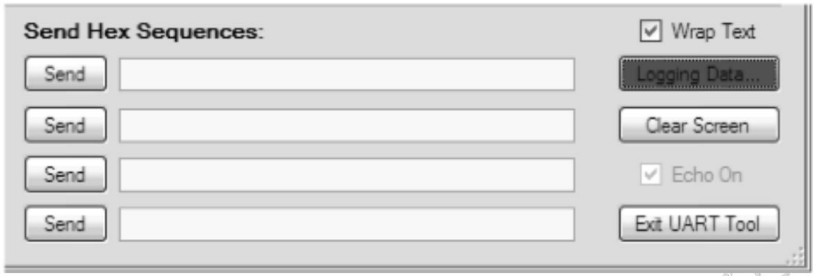
La pantalla del terminal mantendrá las últimas 200 líneas de datos recibidos en el búfer.
La barra de desplazamiento vertical se puede utilizar para ver datos o texto anteriores.

Registrar en

archivo El botón Registrar en archivo permite guardar todo el texto y los datos en un archivo de texto tal como aparece en la pantalla. Haga clic en Registrar en archivo para abrir un cuadro de diálogo Guardar archivo. Especifique la ubicación y el nombre del archivo y haga clic en Guardar. Mientras la pantalla del terminal se guarda en el archivo de registro, el botón Registrar en archivo se volverá verde y mostrará el texto "Registro de datos".

Sample Version - Do Not Copy

Machine Translated by Google



Para detener el registro de datos de la pantalla y cerrar el archivo de registro, haga clic en Registro de datos.

Pantalla clara

Haga clic en el botón Borrar pantalla para borrar todo el texto o datos de la ventana de visualización del terminal. Si los datos se registran en un archivo, borrar la pantalla no afectará el archivo de registro.

Salir de la herramienta UART

Haga clic en Salir de la herramienta UART para cerrar la ventana de la herramienta UART y volver a la ventana principal de la aplicación PICkit 2 Programmer. Los datos y la configuración de la pantalla del terminal de la herramienta UART se conservarán durante cualquier operación de programación hasta que se vuelva a ingresar a la herramienta UART.

Si se cierra la aplicación PICkit 2, la configuración de la herramienta UART, incluidas las cadenas de macros y las secuencias hexadecimales, se guardará y restaurará para la próxima vez que se utilice la herramienta UART. El texto y los datos del búfer de visualización del terminal no se guardan cuando se cierra la aplicación.

Herramienta lógica

La herramienta lógica PICkit 2 permite utilizar los pines del conector ICSP PICkit 2 para estimular y sondear señales digitales en el CHIPINO, y como un simple analizador lógico de 3 canales. La herramienta lógica se abre seleccionando Herramientas > Herramienta lógica en la ventana principal de la aplicación PICkit 2.

Sample Version - Do Not Copy

Machine Translated by Google

La herramienta lógica tiene dos modos de funcionamiento:

1) Modo de E/S lógica

Este modo es útil para activar entradas al CHIPINO y puede monitorear señales digitales para mostrar su estado. En esencia, proporciona una alternativa para conectar botones y LED a pines o señales mientras se depuran o desarrollan funciones de E/S.

2) Modo de analizador lógico

El modo Analizador puede mostrar formas de onda de hasta 3 señales digitales y disparar en eventos específicos, como un flanco ascendente en una señal cuando otra señal está en un nivel lógico alto. Esto puede ser muy útil para depurar buses de comunicación en serie como UART, SPI e I2C. También es muy aplicable para monitorear el comportamiento de la E/S general del microcontrolador.

Sample Version - Do Not Copy

Modo de E/S lógica

El modo “E/S lógica” de la herramienta lógica PICkit 2 es el modo predeterminado cuando se abre por primera vez la herramienta lógica. Permite estímulos simples y monitoreo de señales digitales.
El modo Herramienta lógica se establece mediante los dos botones en la esquina superior derecha de la ventana Herramienta lógica.

El conector de programación CHIPINO de 6 pines tiene 4 pines de señal que se pueden usar para inyectar una señal digital en un circuito o mostrar el estado de una señal digital de un circuito. Los dos pines restantes están dedicados a las conexiones Vdd y Ground.

Los 6 pines pueden funcionar de la siguiente manera en el modo de E/S lógica:

Función de pasador

Solo salida digital 1
2 VDD
3 TIERRA
4 (RB7) Salida digital o Entrada digital
5 (RB6) Salida digital o Entrada digital
6 Salida Digital AUX o Entrada Digital

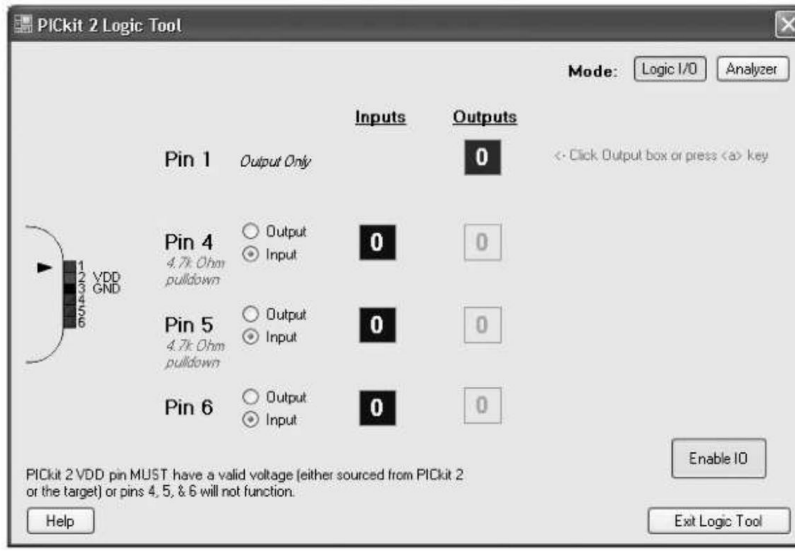
Configuración de la herramienta lógica E/S lógica

Para usar el modo de E/S lógica, el botón de modo de E/S lógica en la parte superior derecha de la ventana lógica debe estar presionado. Los cuatro pines utilizados para las señales digitales de E/S lógica (pines 1, 4, 5 y 6) permanecerán triestablecidos (inactivos) hasta que se presione el botón Enable IO. Una vez que el IO está habilitado, se vuelve activo y se puede configurar. Si no

Machine Translated by Google

se detecta un voltaje válido en el pin VDD al hacer clic en Habilitar IO, aparecerá un cuadro de diálogo para alertar al usuario y los pines permanecerán deshabilitados.

Cuando los pines están habilitados, las direcciones de los pines y los estados de salida se pueden configurar.



Simple Ver

Configuración de la dirección de

los pines Los pines 4, 5 y 6 se pueden configurar como salidas (salida de una señal digital) o entradas (monitoreo de un estado de señal digital conectado al pin). El pin 1 solo está disponible como salida.

Haga clic en los botones de opción junto al número de pin para establecer el pin como salida o entrada. Cuando el pin es una entrada, el estado de la señal conectada se muestra en el cuadro azul "Entradas".

Nota: El pin 4 y el pin 5 tienen una resistencia pulldown de 4,7 k ohmios interna en el PICkit 2. Esta resistencia es necesaria para las funciones del depurador PICkit 2, pero tenga en cuenta que esta resistencia desplegable afectará cualquier señal digital a la que esté conectada. Generalmente, esto es solo un problema cuando se usa el Pin 4 o el Pin 5 como entrada.

Machine Translated by Google

Cuando se selecciona un pin como salida, el pin impulsará el nivel lógico que se muestra en el cuadro de lectura "Salidas". Alterne el estado de salida haciendo clic en el cuadro Estado de salida. Alternativamente, se puede usar una tecla de método abreviado de teclado para cada pin para alternar la salida. Las teclas de acceso directo son:

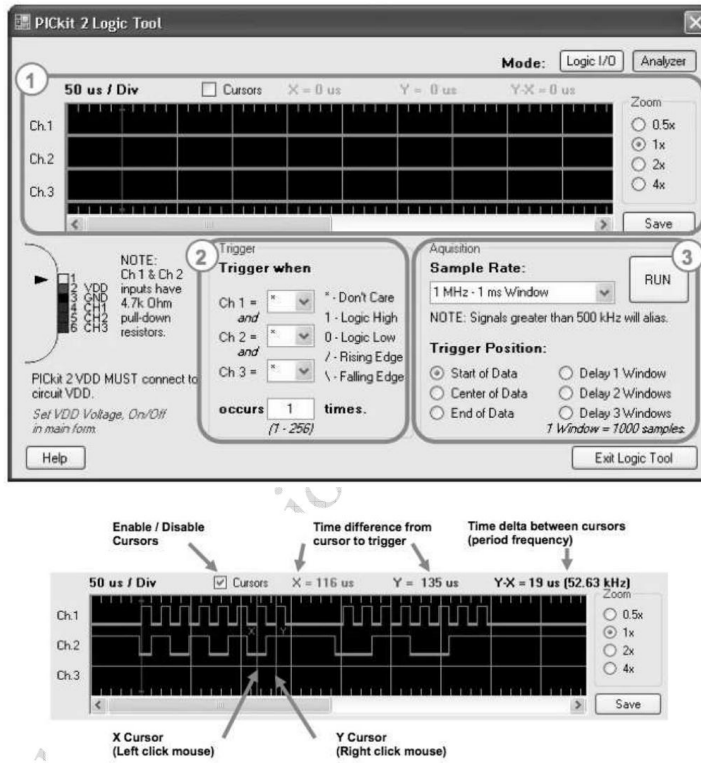
Clave de atajo de PIN
1
4
5
6 <F>

Sample Version - Do Not Copy

Machine Translated by Google

Modo de analizador lógico El

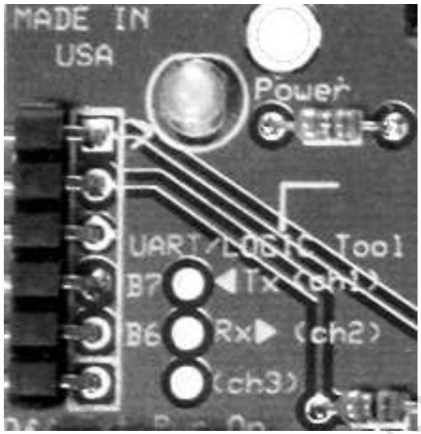
modo de analizador de la herramienta lógica PICkit 2 permite usar la GUI de PICkit 2 como un analizador lógico simple de tres canales para capturar, ver y medir las formas de onda digitales de hasta tres señales.



Conexión del analizador Los

pines 4, 5 y 6 del conector de programación CHIPINO se utilizan como entradas para los tres canales lógicos. Hay orificios en la placa en los que puede soldar un cabezal para conectarse al analizador lógico o puede enrutar señales a los pines B7 y B6 para mostrar la señal.

Machine Translated by Google



Función de pasador
1 MCLR
2 VDD
3 TIERRA
4 (B7) Canal 1
5 (B6) Canal 2
6 AUX Canal 3

Por ejemplo, para monitorear un bus SPI, los pines del canal del analizador podrían conectarse para monitorear las tres señales del bus principal de la siguiente manera:

Pin del analizador lógico	Señal de bus SPI
Canal 1	SCK (reloj maestro de bus)
Canal 2	SDO (salida maestra de bus)
Canal 3	SDI (entrada maestra de bus)

Nota: El pin 4 y el pin 5 tienen una resistencia pulldown interna de 4,7 k ohmios. Esta resistencia es necesaria para las funciones del depurador, pero tenga en cuenta que esta resistencia desplegable afectará cualquier señal digital a la que estén conectados estos pines.

La ventana del analizador lógico La
ventana del analizador de herramientas lógicas se divide en 3 secciones. Éstas incluyen:

1) Pantalla: para ver y medir formas de onda capturadas.
2) Activador: para establecer las condiciones de activación para una captura

Sample Version - Do Not Copy

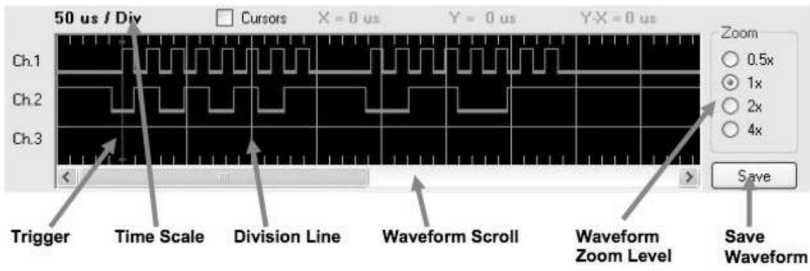
Machine Translated by Google

3) Adquisición: para configurar la frecuencia de muestreo de la forma de onda y la relación de la forma de onda con la muestra de disparo.

Pantalla del analizador La

sección de visualización de la ventana del analizador permite ver, ampliar, medir y guardar la forma de onda como un archivo de mapa de bits.

A continuación se muestra una captura de forma de onda del bus SPI de una transmisión de 2 bytes y se detallan los elementos de la sección de la ventana de visualización.



Disparador

El disparador es un evento predefinido en las señales monitoreadas que provoca una captura de la forma de onda de la señal. La activación se analiza en detalle en la sección 3.2.2 La sección Activación del analizador.

En la pantalla de forma de onda, el punto donde ocurrió el disparo se indica mediante una línea roja vertical. El disparador se configuró para que ocurriera en el primer flanco ascendente del Canal 1, en la señal de reloj SPI SCLK anterior.

Escala de tiempo

Arriba a la izquierda de la pantalla de forma de onda se encuentra la escala de tiempo. Este es el tiempo que representa cada línea de división en la forma de onda. Cada división es de 50 microsegundos de tiempo en la imagen.

Línea de división

Una línea de división es una línea vertical gris a lo largo de la pantalla de forma de onda, que se puede utilizar para dar una referencia de tiempo a la forma de onda mostrada. Las marcas más pequeñas en la parte superior e inferior de la pantalla subdividen cada división de tiempo en 5 unidades más pequeñas.

Desplazamiento de la

forma de onda La forma de onda capturada es más larga de lo que se puede mostrar de una sola vez en la pantalla, por lo que la barra de desplazamiento horizontal permite que la pantalla se desplace para ver la forma de onda completa.

Sample Version

Machine Translated by Google

Nivel de zoom de forma de

onda El zoom de forma de onda permite seleccionar 4 niveles de zoom. Normalmente, con un zoom de 1x, cada muestra de una forma de onda se muestra como un píxel. Una forma de onda tiene 1024 píxeles, de los cuales 500 se pueden mostrar en la ventana. Al seleccionar el nivel de zoom "0.5x", la forma de onda se comprime para que se muestren 2 muestras por píxel, lo que permite ver la forma de onda completa a la vez, pero con pérdida de detalles.

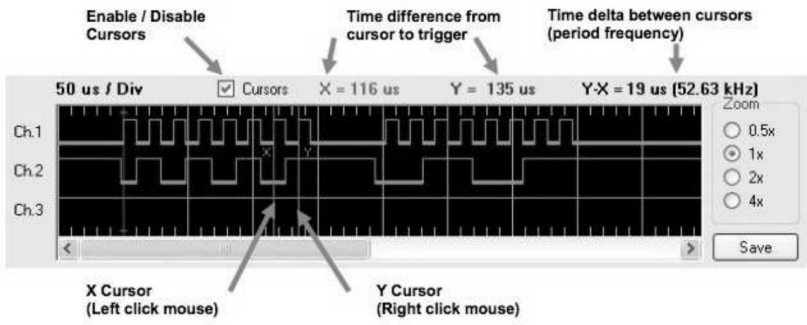
Los niveles de zoom 2x y 4x muestran la forma de onda con 2 píxeles por muestra y 4 píxeles por muestra, respectivamente. Esto permite ver más fácilmente los detalles de tiempo relativo entre las formas de onda.

Guardar forma de

onda Haga clic en el botón Guardar para guardar la pantalla de forma de onda actual en un archivo de mapa de bits. La escala de tiempo se agregará en la parte inferior de la pantalla. Si los cursores están activos, los cursores y sus tiempos también se guardarán con la pantalla.

Cursores de pantalla del analizador

Los cursores de pantalla son útiles para realizar mediciones de tiempo y frecuencia en la forma de onda mostrada. Haga clic para marcar la casilla de verificación "Cursores" y habilite el cursores



Coloque el cursor X haciendo clic izquierdo en la pantalla de forma de onda.
Coloque el cursor Y haciendo clic derecho en la pantalla de forma de onda.

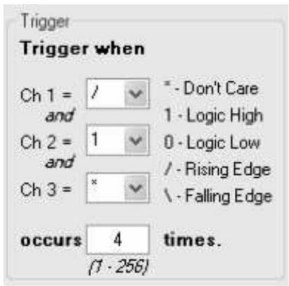
Puede ser útil usar Zoom para la ubicación exacta de los cursores. Cuando se hace zoom, los cursores se "ancharán" ya que tienen el ancho de una muestra y el ancho de la muestra crece al aumentar el zoom.

Encima de la pantalla de forma de onda, se muestra la diferencia de tiempo entre el disparador y cada cursor, junto con la diferencia entre los disparadores. El período de tiempo entre el cursor también se muestra como la frecuencia relacionada.

Machine Translated by Google

Disparador del analizador

El "disparador" es un conjunto de eventos definido por el usuario en las señales monitoreadas que provoca la captura de una forma de onda. A cada canal se le puede asignar uno de los siguientes eventos de activación:



'*' (No importa): el canal del analizador se ignora con fines de activación
'1' (Logic High): el canal debe estar en un estado lógico alto para activar
'0' (Logic Low): el canal debe estar en un estado lógico bajo para activar
'/' (Borde ascendente): el canal debe pasar de un estado bajo a un estado alto para disparar
'\' (Rising Edge): el canal debe pasar de un estado alto a un estado bajo para activarse

Es mejor configurar solo un activador a la vez, pero si configura eventos de activación en todos los canales, deben ocurrir todos a la vez para que el activador active la captura de datos.

Por ejemplo:
estos ajustes se activarán en el borde ascendente del canal 1.

Ch 1 = / (flanco ascendente)
Ch 2 = * (ignorar)
Ch 3 = * (ignorar)

Estos ajustes se disparan en el flanco ascendente del Canal 1 y el Canal 2 a un nivel alto.

Ch 1 = / (flanco ascendente)
Canal 2 = 1 (lógica alta)
Ch 3 = * (ignorar)

También es posible especificar cuántas veces debe ocurrir la condición de disparo antes de que se inicie la captura de forma de onda (hasta 256 veces como máximo).
Por ejemplo, supongamos que queremos capturar el byte 16 de una secuencia de transmisión SPI larga. Si disparamos en el primer borde del reloj del primer byte, probablemente no podríamos ver el byte 16, ya que el analizador se detendría.

Sample Version - Do Not Copy

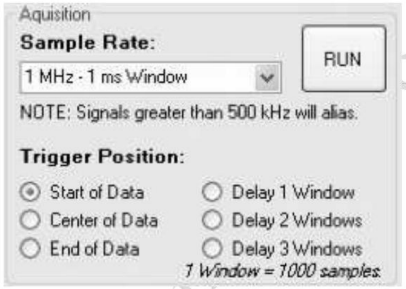
Machine Translated by Google

muestreo antes de que ocurriera. Sin embargo, podemos configurar el analizador para que pase por alto los primeros 15 bytes, configurando el conteo de activación en 15 bytes. De esta manera, comenzará a contar los bordes del reloj en el primer byte, pero no activará la captura de datos hasta el byte 16. es transmitida.

8 relojes * 1 = 121 veces

Adquisición del analizador La

sección "Adquisición" de la ventana del analizador se utiliza para establecer la frecuencia de muestreo de la forma de onda, la posición del disparador en relación con la forma de onda capturada y para iniciar o "ejecutar" el analizador.



Frecuencia de

muestreo La frecuencia de muestreo es la frecuencia con la que se examinan los canales del analizador. Cada captura de forma de onda tiene solo 1024 muestras, por lo que si queremos ver un período de tiempo más largo en la pantalla de forma de onda, tenemos que muestrear con menos frecuencia.
La compensación es que a frecuencias de muestreo más altas, podemos ver más detalles y señales más rápidas, pero solo en una pequeña ventana de tiempo. A frecuencias de muestreo más bajas, podemos ver una ventana más larga de tiempo transcurrido pero con menos detalles y es posible que se pierdan pulsos rápidos. En general, la frecuencia de muestreo debe establecerse al menos 10 veces la frecuencia más alta o 5 veces el ancho de pulso más rápido para obtener una representación decente de la forma de onda.

Cualquier forma de onda que tenga una frecuencia superior a la mitad de la frecuencia de muestreo puede crear un alias. Aliasing significa que se pierden los bordes de la forma de onda y, por lo tanto, la forma de onda puede parecer más lenta de lo que realmente es. Por supuesto, la frecuencia de muestreo siempre se puede establecer más lenta que estos límites si todo lo que se desea es tener una idea general de lo que sucede en el circuito sin muchos detalles.

Sample Version Not Copy

Machine Translated by Google

TARIFAS DE MUESTRA COMPATIBLES

Sample Rate	Time Between Samples	Waveform Limitations	
		Captured Waveform Length[1] (1024 samples)	Maximum Frequency (before aliasing)
1 MHz	1 us	1 ms	500 kHz
500 kHz	2 us	2 ms	250 kHz
250 kHz	4 us	4.1 ms	125 kHz
100 kHz	10 us	10.2 ms	50 kHz
50 kHz	20 us	20.5 ms	25 kHz
25 kHz	40 us	41 ms	12.5 kHz
10 kHz	100 us	102.4 ms	5 kHz
5 kHz	200 us	204.8 ms	2.5 kHz

Note 1: Waveform length is rounded to the nearest 0.1 decimal place.

Posición del disparador
Cambiar la posición del disparador permite una mayor flexibilidad sobre cómo los datos capturados se relacionan con el evento del disparador. Por ejemplo, podríamos estar más interesados en lo que sucedió antes del desencadenante que después.
Hay 6 posiciones de disparo seleccionables:

Inicio de datos

Esta es la posición de disparo utilizada en todas las formas de onda de la figura anterior.
Todos los datos de forma de onda, excepto una división, se capturan después de que se produce el disparo. Esto se usa mejor cuando todos los datos de forma de onda de interés ocurren después del disparo.

Centro de datos Se

utiliza mejor cuando todos los datos de forma de onda de interés ocurren alrededor del disparador.
El evento de disparo está en el medio de la pantalla de forma de onda.

Fin de los datos

Todos los datos de la forma de onda, excepto un poco más de una división, se capturan antes de que se produzca el disparo. Esto se utiliza mejor cuando todos los datos de forma de onda de interés ocurren antes del disparo.

Retardo 1 ventana
Retraso 2 Ventanas
Retraso 3 Ventanas

En estos casos, la posición de disparo se considera "Inicio de datos", pero la captura de forma de onda se retrasa 1000 muestras (casi una visualización de forma de onda) después del disparo.
Esto permite al usuario capturar eventos que ocurren más allá del ancho de la pantalla después de que ocurre un disparo, sin reducir la frecuencia de muestreo.

En otras palabras, cuando se selecciona "Ventana de retardo 1", el analizador esperará 1000 muestras después de que ocurra el evento de activación antes de comenzar a registrar datos de forma de onda.
Cuando se selecciona "Delay 2 Windows", esperará 2000 muestras, etc.

Sample Version - Do Not C

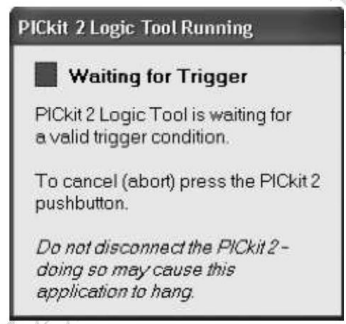
Machine Translated by Google

Cada pantalla de forma de onda tiene 1024 muestras, por lo que el incremento de retardo de 1000 muestras da una pequeña superposición entre capturas de retardo sucesivas. Asumiendo que los datos de interés después del disparador son repetibles y consistentes, esto permite unir una forma de onda total de hasta 4 veces el ancho de la ventana de frecuencia de muestreo.

Por ejemplo, sería posible recopilar 4 ms de datos de forma de onda después de un evento de activación a una frecuencia de muestreo de 1 MHz.

Ejecución del analizador Una vez que

las condiciones de activación, la frecuencia de muestreo y la posición de activación se hayan configurado como desee, haga clic en el botón EJECUTAR para comenzar a recopilar datos de forma de onda y buscar eventos de activación.



Cuando el analizador está funcionando, mostrará el cuadro de diálogo anterior y se encenderá el LED "Ocupado" en la unidad PICkit 2.

Una vez que se cumpla la condición de activación, el LED "Ocupado" se apagará, el cuadro de diálogo "Esperando activación" se cerrará y la pantalla de forma de onda del analizador se actualizará con los datos recién capturados.

Sample Version - Not Copy