Basic Express
Nota de aplicación

Programación del Timer1 como un cronómetro

¿Qué es el temporizador Timer1?

Los sistemas BasicX tienen un temporizador integrado denominado Timer1. Este temporizador puede utilizarse para múltiples funciones, una de las cuales es medir los intervalos de tiempo, que es el que vamos a analizar en esta nota de aplicación.

El temporizador puede operar con 5 **frecuencias de marcación discretas** que oscilan entre los 7,20 a los 7.37 MHz aproximadamente, lo que significa que puede utilizar el temporizador Timer1 para medir intervalos de tiempo que oscilen entre 136 ns y 139 µs.

¿Por qué utilizar un temporizador Timer1? En algunos casos no se puede utilizar el reloj de tiempo real integrado para medir los intervalos de tiempo. Esto es posible si utiliza llamadas del sistema *PulseIn* o *RCtime* para medir los intervalos de tiempo superiores a 2 ms, que potencialmente pueden interferir con el reloj de tiempo real.

Otra razón para utilizar el temporizador Timer1 es medir los intervalos de tiempo con resoluciones menores que la que ofrece el reloj de tiempo real, que tiene una tasa de marcación de 512 Hz. Las tasas de marcación de Timer1 son como mínimo 10 veces inferiores.

Advertencia – El temporizador Timer1 debería utilizarse sólo en los casos en los que no afecte al funcionamiento de otros recursos del sistema, como InputCapture y OutputCapture, que dependen del temporizador Timer1. (Tenga en cuenta que el reloj de tiempo real es completamente independiente del Timer1, y que puede utilizarse de forma paralela a él.)

Los siguientes registros se utilizan para acceder al temporizador Timer1:

Type	Name	Description
Byte	TCNT1L	Timer/Counter1 Low Byte
Byte	TCNT1H	Timer/Counter1 High Byte
Byte	TCCR1B	Timer/Counter1 Control Register B
Byte	TCCR1A	Timer/Counter1 Control Register A
Byte	TIFR	Timer/Counter Interrupt Flag register

Consulte también los siguientes ficheros PDF si desee obtener más información sobre el Timer1, que en realidad recibe el nombre de Timer/Counter1. Estos ficheros se proporcionan en la instalación del BasicX y el documento del chip Atmel que se utiliza como el procesador del BasicX.

BX-01: Fichero AT90S4414_8515.pdf, página 31.

BX-24, BX-35: Fichero AT90S_8535.pdf, página 32.

Modo de uso del temporizador Timer1

Iniciación

El temporizador debe iniciarse antes de empezar a utilizarlo. El primer paso es poner a cero el registro de control TCCR1A del temporizador Timer1. Así se desconecta el Timer1 de los pines de salida OC1A y OC1B, y desactiva la operación PWM del temporizador:

```
Register.TCCR1A = 0
```

El siguiente paso es borrar los 2 bytes del contador del Timer1. El byte alto debe escribirse en primer lugar, y a continuación el byte alto:

```
Register.TCNT1H = 0
Register.TCNT1L = 0
```

El último paso de la iniciación es borrar el **indicador de desbordamiento** TOV1, que corresponde con un bit en el registro TIFR. Tenga en cuenta que escribir un valor lógico 1 para el bit acabaría borrándolo:

```
Const TOV1 As Byte = bx10000000 ' BX-01
Const TOV1 As Byte = bx00000100 ' BX-24, BX-35

Register.TIFR = TOV1
```

Arranque del temporizador Timer1

Una vez que el Timer1 ha sido iniciado, se puede arrancar el temporizador escribiendo uno de los 5 valores enumerados para uno de los registros de control (TCCR1B) del temporizador. Los valores disponibles aparecen a continuación:

Valor TCCR1B	Resolución del timer (µs)	Frecuencia de pulsos (Hz)	Rango de tiempo máximo (s)
1	0.135 633 7	7 372 800	0.008 889
2	1.085 069	921 600	0.071 11
3	8.680 555	115 200	0.568 9
4	34.722 22	28 800	2.276
5	138.888 9	7 200	9.102

En este ejemplo, utilizaremos la frecuencia más baja de 7.2 kHz:

```
Register.TCCR1B = 5
```

Lectura del temporizador Timer1

Una vez que se ha arrancado el temporizador, puede leerlo en cualquier momento a través de los dos registros de tiempo TCNT1L y TCNT1H. Para leer el contador, debe leer el valor de byte bajo, y a continuación, debe leer el valor inferior del byte y después el valor superior (al contrario que el proceso de escritura del contador, en el que el valor superior debe escribirse antes):

```
Dim LowByte As Byte, HighByte As Byte, Count As Long

LowByte = Register.TCNT1L
HighByte = Register.TCNT1H
Count = CLng(HighByte) * 256 + CLng(LowByte)
```

¿What about timer overflow? In this example, we're using the lowest tick frequency of 7.2 kHz, which means the timer overflows after about 9.1 s. You can check the timer overflow flag to see whether an overflow has occurred:

```
Dim TimerHasOverflowed As Boolean

If ((Register.TIFR And TOV1) = TOV1) Then
    TimerHasOverflowed = True
Else
    TimerHasOverflowed = False
End If
```

Pausing and resuming Timer1

To stop the timer without affecting the value of the counter, you clear control register TCCR1B:

```
Register.TCCR1B = 0
```

To cause the timer to resume operation, you write one of the previously-mentioned enumerated values to the control register. In this case we'll have the timer resume at a tick frequency of 28.8 kHz:

```
Register.TCCR1B = 4
```

Example program

An example program can be found in file Timer1Example.bas. The program illustrates the use of Timer1 as a stopwatch. The program makes use of modules Timer1.bas (see the next section), as well as SerialPort.bas for doing serial I/O.

Module Timer1

Module Timer1 gives you a high-level interface to Timer1. The source code for the module has been provided so you can see low-level details at the register level. To use the module, you can include file Timer1.bas in your BXP project file.

The module provides the following calls:

Subprogram Name	Function

GetTimerCount	Reads the value of Timer1, in units of counts
GetTimerValue	Reads the value of Timer1, in units of floating point seconds
InitializeTimer	Initializes the timer and defines the tick frequency
PauseTimer	Stops Timer1 without affecting its current value
PutTimerCount	Writes the value of Timer1, in units of counts
PutTimerValue	Writes the value of Timer1, in units of floating point seconds
ResumeTimer	Restarts Timer1 without affecting its current value
StartTimer	Starts Timer1 after clearing it
TimerHasOverflowed	Determines whether Timer1 has overflowed

Warning -- be careful to avoid conficts with system calls InputCapture and OutputCapture, as well as the Com2 and Com3 serial ports, all of which depend on Timer1.

GetTimerCount procedure

Syntax

Call GetTimerCount(*Count*)

Arguments

Item	Type	Direction	Description
Count	Long	Output	Value of Timer1, in units of counts. Range is 0 to 65 535.

Description

GetTimerCount returns the 2-byte value of Timer1. The value is the equivalent of a 16-bit unsigned integer. The procedure has no effect on whether the timer is running or halted. The time scaling factor depends on the range setting in InitializeTimer.

GetTimerValue procedure

Syntax

Call GetTimerValue(*Value*)

Arguments

Item	Type	Direction	Description
Value	Single	Output	Value of timer. Units are in seconds. The range depends on the range setting.

Description

GetTimerValue returns the current value of the timer. Units are in floating point seconds. See InitializeTimer for the range, which depends on the range setting.

This procedure has no effect on whether the timer is running or halted.

InitializeTimer procedure

Syntax

Call InitializeTimer(*RangeSetting*)

Arguments

Item	Type	Direction	Description
RangeSetting	Byte	Input	Determines the tick frequency and time range. See below for allowable values.

Allowable values for *RangeSetting:*

RangeSetting	Timer Resolution (µs)	Tick Frequency (Hz)	Maximum Time Range (s)
1	0.135 633 7	7 372 800	0.008 889
2	1.085 069	921 600	0.071 11
3	8.680 555	115 200	0.568 9
4	34.722 22	28 800	2.276
5	138.888 9	7 200	9.102

Description

InitializeTimer clears and pauses the timer, defines the tick frequency and clears the overflow flag.

This procedure should be called before any other calls in this module. You can call InitializeTimer any number of times -- to change the tick frequency, for example.

PauseTimer procedure

Syntax

Call PauseTimer

Arguments

None.

Description

PauseTimer stops Timer1 without affecting its current value.

PutTimerCount procedure

Syntax

Call PutTimerCount(*Count*)

Arguments

Item	Type	Direction	Description
Count	Long	Input	Value of Timer1, in units of counts. Range is 0 to 65 535.

Description

PutTimerCount defines a 2-byte value for Timer1. The value is the equivalent of a 16-bit unsigned integer. The time scaling factor depends on the range setting in InitializeTimer.

This procedure also pauses the timer and clears the overflow flag.

PutTimerValue procedure

Syntax

Call PutTimerValue(*Value*)

Arguments

Item	Type	Direction	Description
Value	Single	Input	Value of timer. Units are in seconds. The allowable range depends on range setting.

Description

PutTimerValue writes the value of Timer1. See InitializeTimer for the allowable range, which depends on the range setting.

This procedure also pauses the timer and clears the overflow flag.

ResumeTimer procedure

Syntax

Call ResumeTimer

Arguments

None.

Description

ResumeTimer restarts Timer1 without affecting its current value.

StartTimer procedure

Syntax

Call StartTimer

Arguments

None.

Description

StartTime starts Timer1 after clearing it. The overflow flag is also cleared.

TimerHasOverflowed function

Syntax

F = TimerHasOverflowed

Arguments

Item	Type	Direction	Description
F	Boolean	Output	Whether Timer1 has overflowed.

Description

The TimerHasOverflowed function returns a boolean value that indicates whether Timer1 has overflowed since the last operation that cleared the overflow flag (StartTimer is an example of a procedure that clears the overflow flag).