NetMedia

Basic Express
Application Note

Programming Timer1 as a Stopwatch

What is Timer1?

BasicX systems have a built-in timer called Timer1. This timer can be used for several functions, one of which is measuring time intervals, which we'll address in this application note.

The timer is capable of operating at 5 discrete tick frequencies ranging from approximately 7.20 kHz to 7.37 MHz, which means you can use Timer1 to measure time intervals at resolutions ranging from about 136 ns to 139 µs.

Why use Timer1? One reason is that sometimes you can't use the built-in real time clock for measuring time intervals. This might happen if you use system calls PulseIn or RCtime to measure time intervals longer than about 2 ms, which can potentially interfere with the real time clock.

Another reason to use Timer1 is to measure time intervals at finer resolutions than that provided by the real time clock, which has a tick rate of 512 Hz. Timer1 tick rates are at least an order of magnitude higher.

Warning -- Timer1 should only be used where it does not conflict with other system resources, such as InputCapture and OutputCapture, both of which depend on Timer1. (Note that the real time clock is completely independent of Timer1, and can be used in parallel with it.)

The following registers are used for getting access to 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

See also the following PDF files for more details on Timer1, which is actually called Timer/Counter1. These files are provided in the BasicX installation and document the Atmel chip used as the BasicX processor.

```
BX-01: File AT90S4414_8515.pdf, page 31.

BX-24, BX-35: File AT90S_8535.pdf, page 32.
```

Using Timer1

Initialization

The timer should be initialized before use. The first step is to clear the Timer1 control register TCCR1A. This disconnects Timer1 from output pins OC1A and OC1B, and disables PWM operation of the timer:

```
Register.TCCR1A = 0
```

The next step is to clear the 2 bytes of the Timer1 counter. The high byte must be written first, followed by the low byte:

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

The last initialization step is to clear the overflow flag TOV1, which is a bit in register TIFR. Note that writing a logic 1 to the bit causes it to be cleared:

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

Register.TIFR = TOV1
```

Starting Timer1

Once Timer1 has been initialized, the timer can be started by writing one of 5 enumerated values to one of the timer's control registers (TCCR1B). The allowable values are shown below:

TCCR1B Value	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

In this example, we'll us the lowest frequency of 7.2 kHz:

```
Register.TCCR1B = 5
```

Reading Timer1

Once the timer has started, you can read it at any time by reading the two counter registers TCNT1L and TCNT1H. To read the counter, you must read the low byte first, followed by the high byte (note that this is the reverse of writing to the counter, where the high byte must be written first):

```
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).

2.00.A