
# UnoLib documentation

## timer.pas

version 10/26/2025

The timer.pas module contains low-level routines for counting clock cycles and pausing the program. Some of the routines were originally preprocessor functions (macros) in the Arduino core library, but because FPC doesn't support such routines, they were translated as regular procedures or functions. Consequently, the binary code produced by FPC is slightly larger than the Arduino code.

| **Routines** |
|---|
| `function` **`ClockCyclesToMicroseconds`**`(aCnt: UInt32): UInt32;`<br><br>Converts the number of clocks to microseconds.<br><br>*Parameters*<br>*aCnt* – number of clock cycles.<br><br>Note: as macro in Arduino sources. |
| `function` **`MicrosecondsToClockCycles`**`(aCnt: UInt32): UInt32;`<br><br>Converts the microseconds to the number of clocks.<br><br>*Parameters*<br>*aCnt* – the number of microseconds.<br><br>Note: as macro in Arduino sources. |
| `function` **`Micros`**`: UInt32;`<br><br>Returns the number of microseconds since the board has booted. |
| `function` **`Millis`**`: UInt32;`<br><br>Returns the number of milliseconds since the board has booted. |
| `procedure` **`Delay`**`(ms: UInt32);`<br><br>Pauses the program for the amount of time (in milliseconds) specified as parameter.<br><br>*Parameters*<br>*ms* - the number of milliseconds to pause. |
| `procedure` **`Delay2`**`(const ms: Uint16);`<br><br>Pauses the program for the amount of time (in milliseconds) specified as parameter. The procedure is limited to 65535 ms (about 65 seconds). It is intended for shorter latencies and is |

more optimal in terms of code size and execution speed.

*Parameters*
*ms* - the number of milliseconds to pause.

Note: based on counting of clock cycles.

```
procedure DelayMicroseconds(const us: UInt16);
```

Pauses the program for the amount of time (in microseconds) specified as parameter.

*Parameters*
*us* - the number of microseconds to pause.

Note: based on counting of clock cycles.