UnoLib documentation

timer.pas

version 11/10/2025

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;` Converts the number of clocks to microseconds. *Parameters* *aCnt* – number of clock cycles. Note: as macro in Arduino sources.
`function` **`MicrosecondsToClockCycles`**`(aCnt: UInt32): UInt32;` Converts the microseconds to the number of clocks. *Parameters* *aCnt* – the number of microseconds. Note: as macro in Arduino sources.
`function` **`Micros`**`: UInt32;` Returns the number of microseconds since the board has booted.
`function` **`Millis`**`: UInt32;` Returns the number of milliseconds since the board has booted.
`procedure` **`Delay`**`(ms: UInt32);` Pauses the program for the amount of time (in milliseconds) specified as parameter. *Parameters* *ms* - the number of milliseconds to pause.
`procedure` **`Delay2`**`(const ms: Uint16);` 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.

Example

This example demonstrates blinking the built-in LED (Pin 13) without using the blocking *Delay* function. This technique allows the microcontroller to perform other tasks while simultaneously managing the timing for the LED. It includes three core units: *defs*, *timer*, and *digital*.

The program enters the infinite loop (while True do) that handles the timing. First, it reads the number of milliseconds that have passed since the Arduino board started (*Millis*). Next, it executes a comparison to check if the time elapsed since the last update is greater than or equal to the desired interval. If time is up the program resets the timer by storing the current time as the new starting point, toggles the LED state (if it was LOW, it becomes HIGH, and vice versa) by updating the physical state by *DigitalWrite*(LED_BUILTIN, *LedState*).

```
program TestBlinkWithoutDelay;

{
  created 2005 by David A. Mellis
  modified 8 Feb 2010 by Paul Stoffregen
  modified 11 Nov 2013 by Scott Fitzgerald
  modified 9 Jan 2017 by Arturo Guadalupi
  AVRPascal version Oct 2024 by Henk Heemstra
  modified 13 Oct 2024 by Andrzej Karwowski
}

{$IF NOT (DEFINED(atmega328p) or DEFINED(arduinouno) or
DEFINED(arduinonano) or DEFINED(fpc_mcu_atmega328p) or
DEFINED(fpc_mcu_arduinouno) or DEFINED(fpc_mcu_arduinonano))}
 {$Fatal Invalid controller type, expected: atmega328p, arduinouno, or
arduinonano}
{$ENDIF}

{$mode objfpc}

uses
  defs, timer, digital;

const
```

```
  LED_BUILTIN = 13; //internal LED
  INTERVAL = 1000;

var
  LedState: UInt8 = LOW;
  PreviousMillis: UInt32 = 0;
  CurrentMillis: UInt32;

begin
  PinMode(LED_BUILTIN, OUTPUT);

  while True do
  begin
    CurrentMillis := Millis;
    if (CurrentMillis - PreviousMillis >= INTERVAL) then
    begin
      PreviousMillis := CurrentMillis;
      LedState := not LedState;
      DigitalWrite(LED_BUILTIN, LedState);
    end;
  end;
end.
```