NAME

util_delay − <util/delay.h>: Convenience functions for busy-wait delay loops

Macros

#define **F_CPU** 1000000UL

Functions

void **_delay_ms** (double __ms)
void **_delay_us** (double __us)

Detailed Description

```
#define F_CPU 1000000UL  // 1 MHz
//#define F_CPU 14.7456E6
#include <util/delay.h>
```

Note:

As an alternative method, it is possible to pass the F_CPU macro down to the compiler from the Makefile. Obviously, in that case, no `#define` statement should be used.

The functions in this header file are wrappers around the basic busy-wait functions from <**util/delay_basic.h**>. They are meant as convenience functions where actual time values can be specified rather than a number of cycles to wait for. The idea behind is that compile-time constant expressions will be eliminated by compiler optimization so floating-point expressions can be used to calculate the number of delay cycles needed based on the CPU frequency passed by the macro F_CPU.

Note:

In order for these functions to work as intended, compiler optimizations *must* be enabled, and the delay time *must* be an expression that is a known constant at compile-time. If these requirements are not met, the resulting delay will be much longer (and basically unpredictable), and applications that otherwise do not use floating-point calculations will experience severe code bloat by the floating-point library routines linked into the application.

The functions available allow the specification of microsecond, and millisecond delays directly, using the application-supplied macro F_CPU as the CPU clock frequency (in Hertz).

Macro Definition Documentation

#define F_CPU 1000000UL

CPU frequency in Hz. The macro F_CPU specifies the CPU frequency to be considered by the delay macros. This macro is normally supplied by the environment (e.g. from within a project header, or the project’s Makefile). The value 1 MHz here is only provided as a ’vanilla’ fallback if no such user-provided definition could be found.

In terms of the delay functions, the CPU frequency can be given as a floating-point constant (e.g. 3.6864E6 for 3.6864 MHz). However, the macros in <**util/setbaud.h**> require it to be an integer value.

Function Documentation

void _delay_ms (double __ms)

Perform a delay of `__ms` milliseconds, using **_delay_loop_2()**.

The macro F_CPU is supposed to be defined to a constant defining the CPU clock frequency (in Hertz).

The maximal possible delay is 262.14 ms / F_CPU in MHz.

When the user request delay which exceed the maximum possible one, **_delay_ms()** provides a decreased resolution functionality. In this mode **_delay_ms()** will work with a resolution of 1/10 ms, providing delays up to 6.5535 seconds (independent from CPU frequency). The user will not be informed about decreased resolution.

If the avr-gcc toolchain has __builtin_avr_delay_cycles() support, maximal possible delay is 4294967.295 ms/ F_CPU in MHz. For values greater than the maximal possible delay, overflows results in no delay i.e., 0ms.

Conversion of `__ms` into clock cycles may not always result in integer. By default, the clock cycles rounded up to next integer. This ensures that the user gets at least `__ms` microseconds of delay.

Alternatively, by defining the macro `__DELAY_ROUND_DOWN__`, or `__DELAY_ROUND_CLOSEST__`, before including this header file, the algorithm can be made to round down, or round to closest integer, respectively.

Note:

The implementation of **_delay_ms()** based on __builtin_avr_delay_cycles() is not backward compatible with older implementations. In order to get functionality backward compatible with previous versions, the macro `'__DELAY_BACKWARD_COMPATIBLE__'` must be defined before including this header file. Also, the backward compatible algorithm will be chosen if the code is compiled in a *freestanding environment* (GCC option `-ffreestanding`), as the math functions required for rounding are not available to the compiler then.

void _delay_us (double __us)

Perform a delay of `__us` microseconds, using **_delay_loop_1()**.

The macro F_CPU is supposed to be defined to a constant defining the CPU clock frequency (in Hertz).

The maximal possible delay is 768 us / F_CPU in MHz.

If the user requests a delay greater than the maximal possible one, **_delay_us()** will automatically call **_delay_ms()** instead. The user will not be informed about this case.

If the avr-gcc toolchain has __builtin_avr_delay_cycles() support, maximal possible delay is 4294967.295 us/ F_CPU in MHz. For values greater than the maximal possible delay, overflow results in no delay i.e., 0us.

Conversion of `__us` into clock cycles may not always result in integer. By default, the clock cycles rounded up to next integer. This ensures that the user gets at least `__us` microseconds of delay.

Alternatively, by defining the macro `__DELAY_ROUND_DOWN__`, or `__DELAY_ROUND_CLOSEST__`, before including this header file, the algorithm can be made to round down, or round to closest integer, respectively.

Note:

The implementation of **_delay_ms()** based on __builtin_avr_delay_cycles() is not backward compatible with older implementations. In order to get functionality backward compatible with previous versions, the macro `__DELAY_BACKWARD_COMPATIBLE__` must be defined before including this header file. Also, the backward compatible algorithm will be chosen if the code is compiled in a *freestanding environment* (GCC option `-ffreestanding`), as the math functions required for rounding are not available to the compiler then.

Author

Generated automatically by Doxygen for avr-libc from the source code.