lib_xassert: Assertions library

Publication Date: 2026/5/11
Document Number: XM-006382-UG v5.0.0

XMOS

IN THIS DOCUMENT

1 Introduction

This library provides a lightweight and flexible replacement for the standard C header `assert.h`.

The assertions in this library can be enabled/disabled and configured as to how much information they show. This configuration can be per *xassert unit* (i.e. for sets of files).

`lib_xassert` is intended to be used with the XCommon CMake , the *XMOS* application build and dependency management system.

2 API

To use this module, include `lib_xassert` in the application's `APP_DEPENDENT_MODULES` list and include the `xassert.h` header file.

2.1 Assertions

An assertion can be inserted into code with the `assert` macro e.g.:

```
assert(i < n);
```

Optionally a debug message can be added with the `msg` macro:

```
assert(i < n && msg("i must be less than the array bound"));
```

If assertions are enabled and the expression in the assertion is false than a trap will occur.

2.2 Unreachable

If the logic of a program dictates that certain code cannot be reached, the `unreachable` macro can be used e.g.:

```
switch (message) {
case 0:
  ...
case 1:
  ...
default:
  unreachable("message must be 0 or 1");
  break;
}
```

If assertions are enabled then this macro will cause a trap if executed.

2.3 Fail

A failure can be indicated with the `fail` macro e.g.:

```
if (reg_value != 0xA5)
  fail("device not connected properly")
```

A fail will always cause a trap if executed. A failure differs from unreachable in that an unreachable macro should never execute in a correct program whereas a fail could happen in catastrophic circumstances even if the program is correct.

3 Timing

lib_xassert provides a mechanism to check that certain code executes within a certain time.

Two mechanisms are provided for this, one for checking a block of code completes in a given time (in increments of timer ticks, 1000/REF_TIMER_MHZ ns, typically 10 ns) and another for checking that a loop executes at a given frequency (in Hz).

Timing assertions can be enabled/disabled via the `XASSERT_ENABLE_TIMING_ASSERTIONS` define. They respect other settings such as `XASSERT_ENABLE_DEBUG` and `XASSERT_ENABLE_LINE_NUMBERS` as documented in *Controlling assertions*.

> **Note**
>
> Adding timing asserts adds some overhead to the code.

> **Warning**
>
> Timing assertions maintain global mutable state and are not thread-safe or multi-core safe. They must only be used from a single thread/core. Using these APIs concurrently from multiple threads/cores can produce incorrect results.

> **Warning**
>
> Timing assertion state is defined in the header as translation-unit-local static data. Each source file gets its own timing buffer. A `xassert_timing_start()` in one source file will not match a `xassert_timing_end()` in another source file, and timing state memory is duplicated across source files.

The maximum number of timed items is set by `XASSERT_MAX_TIMING_BLOCKS`. If a code-base exceeds this number of timed items, then the oldest timed item will be dropped when a new one is added. This drop is silent by default; a warning is printed only when `XASSERT_ENABLE_DEBUG` is enabled.

3.1 Timing blocks

Code blocks can be timed with the `xassert_timing_start()` and `xassert_timing_end()` functions. For example:

```
xassert_timing_start("test", 1000);
// code to be timed
xassert_timing_end("test");
```

The first argument to these functions is a string *tag* used to match start/end calls. To avoid collisions, tags should be unique for each concurrently active timed block that you want to track independently.

For convenience, code blocks can also be timed with the `XASSERT_TIMED_BLOCK` macro. This wraps the passed code in calls to `xassert_timing_start()` and `xassert_timing_end()`, for example:

```
XASSERT_TIMED_BLOCK("test", 1000,
    // code to be timed
)
```

This will check that the code in the block executes within 1000 timer ticks, typically 1000 * 10 ns = 10 us. If the code takes longer than this (and assertions are enabled) then a trap will occur.

3.2 Timing loops

In some instances it is desirable to be able to check that a loop executes at a given frequency. That can be done with the `xassert_loop_freq()` function. This function can be inserted at any point in a loop, for example:

```
while (1) {
    // do some work
    xassert_loop_freq("test loop", 1000);
}
```

This will check that the loop executes at 1000 Hz. If a loop iteration executes slower than this (and assertions are enabled) then a trap will occur. A small tolerance is allowed via `XASSERT_TIMING_DELTA_ERROR` (defaults to 2 timer ticks). The first argument is a string *tag* used to identify the loop being timed.

It is not unusual for a loop, on occasion, to execute some additional code outside of its normal operation, for example, a loop processing audio may occasionally change sample rate, which may require configuring external hardware etc that may take some time but does not affect the normal operation of the program. It is not desirable for such code to cause a timing assertion failure. To allow for this, the `xassert_loop_exception()` function can be used to indicate that a loop iteration is executing code that should be excluded from timing assertions. For example:

```
while (1) {
    // do some work
    if (need_to_change_sample_rate) {
        xassert_loop_exception("test loop");
        // code to change sample rate
    }
    xassert_loop_freq("test loop", 1000);
}
```

4 Controlling assertions

Assertions can be enabled/disabled via command line options to your application build. The following defines can be set by using the `-D` option to the compiler. For example, the following in your application *CMakeLists.txt* will enable line numbers in assertion messages:

```
set(APP_COMPILER_FLAGS -DXASSERT_ENABLE_LINE_NUMBERS=1)
```

The following defines can be set:

XASSERT_ENABLE_ASSERTIONS
: This define can be used to turn assertions on or off (defaults to 1).

XASSERT_ENABLE_TIMING_ASSERTIONS
: This define can be used to turn timing assertions on or off (defaults to 1).

XASSERT_ENABLE_DEBUG
: This define will cause assertions to print out the failing expression before trapping (defaults to 0). Note that this option could significantly increase the code size of your application.

XASSERT_ENABLE_LINE_NUMBERS
: This define will cause assertions to print the file and line number of the assertion before trapping. Note that this option could significantly increase the code size of your application.

XASSERT_MAX_TIMING_BLOCKS
: This define sets the maximum number of timed items (blocks and loops) that can be tracked at any one time. If a code-base exceeds this number of timed items, then the oldest timed item will be dropped when a new one is added. The default value is 8

If `XASSERT_UNIT` is defined when `xassert.h` is included then all the assertions in that file belong to that unit. Assertions can then be controlled per debug unit. The mechanism is similar to that used in `module_logging`.

XASSERT_ENABLE_ASSERTIONS_[xassert unit]
: Enable asserts for a particular debug unit. If set to 1, this overrides the default set by `XASSERT_ENABLE_ASSERTIONS` for that debug unit.

XASSERT_DISABLE_ASSERTIONS_[xassert unit]
: Disable asserts for a particular debug unit. If set to 1, this overrides the default set by `XASSERT_ENABLE_ASSERTIONS` for that debug unit.

XASSERT_ENABLE_TIMING_ASSERTIONS_[xassert unit]
: Enable timing asserts for a particular debug unit. If set to 1, this overrides the default set by `XASSERT_ENABLE_TIMING_ASSERTIONS` for that debug unit.

XASSERT_DISABLE_TIMING_ASSERTIONS_[xassert unit]
: Disable timing asserts for a particular debug unit. If set to 1, this overrides the default set by `XASSERT_ENABLE_TIMING_ASSERTIONS` for that debug unit.

XASSERT_ENABLE_DEBUG_[xassert unit]
: Enable debug messages for a particular debug unit. If set to 1, this overrides the default set by `XASSERT_ENABLE_DEBUG` for that debug unit.

XASSERT_DISABLE_DEBUG_[xassert unit]
: Disable debug messages for a particular debug unit. If set to 1, this overrides the default set by `XASSERT_ENABLE_DEBUG` for that debug unit.