Lightweight assertions library

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

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

Features

- Low memory usage
- Ability to enable or disable various features via compile time defines

Software version and dependencies

This document pertains to version 2.0.0 of this library. It is known to work on version 14.1.1 of the xTIMEcomposer tools suite, it may work on other versions.

The library does not have any dependencies (i.e. it does not rely on any other libraries).

1 API

To use the module you need to use `lib_xassert` in your application and include the `xassert.h` header.

1.1 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.

1.2 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.

1.3 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.

1.4 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 `Makefile` will enable line numbers in assertion messages:

```
XCC_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_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.

If DEBUG_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_[debug 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_ENABLE_DEBUG_[debug 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_ASSERTIONS_[debug 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_DISABLE_DEBUG_[debug 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 .

APPENDIX A - Known Issues

There are no known issues with this library.

APPENDIX B - Locks library change log

B.1 2.0.0

- Restructured library