ThreadX Run-time Stack Checking

Run-time Stack Checking is a standard feature of ThreadX that can be enabled by the user when building the ThreadX Library. The Run-Time Stack Checking feature identifies stack corruption and overflow situations automatically, and enables the user to take appropriate action through a user-defined error handler.

To enable run-time stack checking, simply build the ThreadX library with TX_ENABLE_STACK_CHECKING defined. With stack checking enabled, ThreadX examines the stack integrity of every thread being scheduled prior to execution, as well as upon suspension. The run-time stack checking checks the following conditions:

1. Stack starting memory area contains non-“0xEFEFEFEF” data pattern.
2. Stack ending memory area contains non-“0xEFEFEFEF” data pattern.
3. Current stack pointer is less than stack starting memory address.

If any of these conditions are detected, ThreadX immediately calls the internal ThreadX default stack error handler _tx_thread_stack_error_handler_. Alternatively, the application may register its own stack error handler by supplying a callback function to tx_thread_stack_error_notify. The following figure shows the thread control block for a typical thread.

---

**Figure: Typical thread stack area**
In this example, ThreadX checks the stack starting and ending addresses at 0xF200 and 0xFC00, respectively, to make certain the 0xEFEEEEFE pattern is intact in both locations. In addition, ThreadX checks to make certain the current stack pointer is within the range of the stack memory area. If any stack error conditions are detected, stack error handling is called immediately.

In addition to checking for overflow conditions, ThreadX run-time stack checking keeps track of the high water mark of stack usage. This address is subtracted from the ending address, which yields the amount of stack space used. The TX_THREAD structure member `tx_thread_stack_highest_ptr` contains the highest used address of the thread’s stack.

The user is cautioned to consider the scope of application testing, in an effort to determine whether stack overflow might occur. Run-time stack checking is only as good as the tests used to generate worst-case function call depth and local variable usage. If the tests generate the worst-case call depth for every function, the developer is assured that there is no chance of stack overflow. Then, stack checking can be de-activated in the production units. However, if the tests might not cover the worst case scenario, then an overflow can still occur after product deployment. If this case is deemed possible, the developer might opt to leave stack checking active in a production unit, with an appropriate error handler to reset the unit or otherwise notify the user.

To summarize, the principal advantage of ThreadX run-time stack checking is that overflow detection occurs closer to the point where the overflow occurred and it does not rely on a developer spotting the overflow condition in a debug session. It also provides the high water mark so that stack size tuning is possible.