ThreadX SMP

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

Revision 5.6.2

Contents

Chapter 1

Introduction to ThreadX SMP

ThreadX SMP brings Symmetric Multi-Processing (SMP) technology to embedded applications. ThreadX application threads (of varying priority) that are "READY" to run are dynamically allocated to available processor cores during scheduling. This results in true SMP processing, including automatic load balancing of application thread execution across all available processor cores.

Initialization

Initialization is performed by or initiated by core 0, which is the default running core after reset.

Interrupts

All cores are allowed to process interrupts. The mapping of interrupts to cores is under the direct control of the application. The ThreadX timer interrupt is by default assigned to core 0 for processing. Please see the code in *tx_timer_interrupt.S* for implementation of this assignment.

Synchronization

All ThreadX synchronization primitives such as semaphores, mutexes, and event flags work seamlessly regardless of which thread, ISR, or core they are called from. For example, if a thread running on core 0 owns a mutex, then a different thread attempting to obtain the same mutex while running in parallel on core 1 is blocked.

Communication

ThreadX queue communication primitives also work seamlessly regardless of which thread, ISR, or core they are called from.

Memory Management

All ThreadX memory management primitives such as byte pools and block pools work seamlessly regardless of which thread and core they are called from.

SMP Scheduling Differences

The biggest difference in behavior between ThreadX SMP and ThreadX for a single core system is that multiple threads, of multiple priority levels can be executing simultaneously.

ThreadX SMP Extensions

There are several extensions to ThreadX SMP that provide the application control over the parallel execution of threads across varying priority levels. However, before the extensions are described, it is worthwhile to note that the standard ThreadX *preemption-threshold* technology inherently provides control of parallel execution. When a thread sets *preemption-threshold* and is assigned a core, ThreadX SMP prevents any other ready threads below (and including) the *preemption-threshold* value from being executed. Hence, this provides a very useful mechanism to regulate thread execution in ThreadX SMP. In addition to *preemption-threshold*, the following ThreadX SMP extensions are provided for additional execution control:

tx_thread_smp_core_exclude
tx_thread_smp_core_exclude_get
tx_thread_smp_core_get
tx_timer_smp_core_exclude
tx_timer_smp_core_exclude_get

The ***tx_thread_smp_core_exclude*** service allows the application to exclude a thread from execution on one or more cores. By default, threads are allowed to execute on any core.

The ***tx_thread_smp_core_exclude_get*** service allows the application to retrieve the current core exclusion list for the specified thread.

The ***tx_thread_smp_core_get*** service allows the application to retrieve the current core the thread is executing on.

The ***tx_timer_smp_core_exclude*** service allows the application to exclude a timer from execution on one or more cores. By default, timers are allowed to execute on any core.

The ***tx_thread_smp_core_exclude_get*** service allows the application to retrieve the current core exclusion list for the specified timer.

All of the new ThreadX SMP API extensions are described fully in *Chapter 4*.

Chapter 2

ThreadX SMP Installation and Use

This chapter contains a description of various issues related to the installation and use of ThreadX SMP.

Product Distribution

ThreadX SMP is distributed in full source code   just like the standard ThreadX distribution. There are additional files associated with the SMP implementation, but in general these are not relevant to the application.

Please refer to the supplied ***readme_thread.txt***, ***readme_threadx_generic.txt*** , and ***readme_threadx_generic_smp.txt*** files for additional information about the distribution as well as the complete version history of ThreadX SMP.

Product Installation

In order to use follow the tool-specific build and installation instructions found in ***readme_threadx.txt***.

Configuring ThreadX SMP

There is a small number of ThreadX SMP configuration options, as described below. Defining these options when building the ThreadX library source will result in the desired configuration.

Define	**Meaning**
TX_THREAD_SMP_MAX_CORES	Defines the maximum number of cores.

Define	Meaning
TX_THREAD_SMP_CORE_MASK	Defines the bit map mask for CORE exclusion. For example, a 4 core environment has a value of 0xF for this define.
TX_THREAD_SMP_DYNAMIC_CORE_MAX	Defined, this enables the dynamic maximum number of cores that can be adjusted at run-time.
TX_THREAD_SMP_INTER_CORE_INTERRUPT	Defined, ThreadX SMP generates inter-core interrupts.
TX_THREAD_SMP_DEBUG_ENABLE	Defined, ThreadX SMP debug information is saved in a circular buffer.
TX_THREAD_SMP_WAKEUP_LOGIC	Defined, the application macro to wakeup core "i" is invoked. This should be defined prior to inclusion of ***tx_port.h***.
TX_THREAD_SMP_WAKEUP(i)	Defines an application macro to wakeup core "i". This should be defined prior to inclusion of ***tx_port.h***.
TX_THREAD_SMP_EQUAL_PRIORITY	Defined, ThreadX SMP only schedules equal priority threads in parallel. This should be defined prior to building the ThreadX library.
TX_THREAD_SMP_ONLY_CORE_0_DEFAULT	Defined, ThreadX SMP defaults all threads and timers to execute only on core 0 by default. The application may override this by calling the core exclude APIs. This should be defined prior to building the ThreadX library.

Chapter 3

ThreadX SMP Implementation

Most of ThreadX remains unchanged for ThreadX SMP. The structural changes center on several internal ThreadX data structures. In addition, several C and assembly files have changed as well as the addition of completely new C and assembly files. However, the vast majority of ThreadX features remains unchanged.

Data Structure Changes

ThreadX SMP necessitates changes to several internal ThreadX data structures. The change is effectively converting a single entry into an array, where the index in the array is the executing core. Below are the data structure changes for ThreadX SMP:

ThreadX	**ThreadX SMP**
_tx_thread_current_ptr	**_tx_thread_current_ptr**[TX_THREAD_SMP_MAX_CORES]
_tx_thread_execute_ptr	**_tx_thread_execute_ptr**[TX_THREAD_SMP_MAX_CORES]
_tx_thread_system_state	**_tx_thread_system_state**[TX_THREAD_SMP_MAX_CORES]
_tx_thread_system_stack_ptr	**_tx_thread_system_stack_ptr**[TX_THREAD_SMP_MAX_CORES]
_tx_timer_time_slice	**_tx_timer_time_slice**[TX_THREAD_SMP_MAX_CORES]

Each of the above arrays are indexed by core ID. For example, *_tx_thread_current_ptr[1]* contains the currently executing thread on core 1 and *_tx_thread_current_ptr[5]* contains the currently executing thread on core 5.

The *_tx_thread_current_ptr[]* array contains the thread pointer currently running on each core, while *_tx_thread_execute_ptr[]* array contains the next thread to run on each core. A *NULL* value indicates that no thread is running or ready to run. Typically, these values are the same, but do change during preemption and context switching.

The *_tx_thread_system_state[]* array contains the system state for each core. A value of 0 in this array indicates the core is either idle or executing a thread. A non-zero value typically indicates the core is executing within an ISR. The only exception is during initialization at this time each entry in the array contains a very large value.

The *_tx_thread_system_stack[]* array contains the system stack pointer for each core. Each core switches to its system stack upon return to the scheduler or at the beginning of interrupt processing. Note that the size and location of the system stack for each core is implementation dependent.

The *_tx_timer_time_slice[]* array contains the current time-slice for the thread running on each core. A value of 0 indicates that no time-slice is active.

Internal ThreadX SMP Protection

The internal TX_DISABLE and TX_RESTORE macros used for protection regions of ThreadX from interrupt processing are expanded to protect against multiprocessor access in ThreadX SMP. This protection is accomplished via a combination of a global test-set flag as well as interrupt control for the core holding the test-set flag.

The global ThreadX SMP variable containing the protection is ***_tx_thread_smp_protection***.

Standard ThreadX Files Changed for SMP

Although the vast majority of standard ThreadX files remains unchanged for ThreadX SMP, SMP support necessitated changes in the following files:

File	**Description**
tx_api.h	ThreadX for SMP (API, constants, etc.)
tx_port.h	Processor-specific information for SMP
tx_thread.h	Thread component definitions
tx_timer.h	Timer component definitions
tx_trace.h	TraceX trace support definitions
tx_initialize_kernel_enter.c	ThreadX entry processing
tx_initialize_kernel_setup.c	ThreadX pre-setup processing
tx_mutex_priority_change.c	Mutex priority change
tx_thread_create.c	Thread creation API
tx_thread_initialize.c	Thread component initialization
tx_thread_preemption_change.c	Thread preemption-threshold change API
tx_thread_priority_change.c	Thread priority change API
tx_thread_relinquish.c	Thread relinquish
tx_thread_resume.c	Thread resumption API
tx_thread_suspend.c	Thread suspension API
tx_thread_system_preempt_check.c	Checks for thread preemption
tx_thread_system_resume.c	Internal thread resume processing
tx_thread_system_suspend.c	Internal thread suspend processing
tx_thread_time_slice.c	Internal time-slice processing
tx_thread_time_slice_change.c	Time-slice change API
tx_timer_initialize.c	Timer component initialization
tx_timer_create.c	Timer create
tx_timer_thread_entry.c	Timer thread processing

Most of the assembly files (tx*.s) are specific to the ThreadX SMP for the given processor platform.

New ThreadX SMP Files

The following are new files for ThreadX SMP:

tx_thread_smp_core_get.[s, asm]	Get core ID
tx_thread_smp_core_preempt.[s, asm]	Interrupt specified core for preemption
tx_thread_smp_current_state_get.[s, asm]	Get current state
tx_thread_smp_current_state_set.c	Sets the current state for all cores
tx_thread_smp_current_thread_get.[s, asm]	Get currently running thread on current core
tx_thread_smp_debug_entry_insert.c	Insert a debug record in debug buffer
tx_thread_smp_high_level_initialize.c	High-level SMP initialization
tx_thread_smp_initialize_wait.[s, asm]	Waiting area for additional cores
tx_thread_smp_low_level_initialize.[s, asm]	Low-level SMP initialization
tx_thread_smp_protect.[s, asm]	Get SMP protection
tx_thread_smp_rebalance_execute_list.c	Rebalance the thread execution list for all cores
tx_thread_smp_core_exclude.c	Excludes one or more cores from executing thread
tx_thread_smp_core_exclude_get.c	Retrieves the exclusion list for a thread
tx_thread_smp_time_get.[s, asm]	Get time stamp for debug and trace
tx_thread_smp_unprotect.[s, asm]	Release SMP protection
tx_timer_smp_core_exclude.c	Excludes one or more cores from executing timer
tx_timer_smp_core_exclude_get.c	Retrieves the exclusion list for a timer

Chapter 4

ThreadX SMP Extensions

As mentioned previously, extensions have been made to ThreadX to provide the application control over when threads of different priority execute and what core they execute on. The following are the ThreadX SMP extensions for execution control:

tx_thread_smp_core_exclude
tx_thread_smp_core_exclude_get
tx_thread_smp_core_get
tx_timer_smp_core_exclude
tx_timer_smp_core_exclude_get

Each of these APIs is discussed in the remainder of this chapter.

tx_thread_smp_core_exclude

Exclude thread execution on a set of cores

Prototype

```
UINT  tx_thread_smp_core_exclude(TX_THREAD *thread_ptr,
                                  ULONG exclusion_map);
```

Description

This function excludes the specified thread from executing on the core(s) specified in the bit map called "*exclusion_map*." Each bit in "*exclusion_map*" represents a core (bit 0 represents core 0, etc.). If the bit is set, the corresponding core is excluded from executing the specified thread.

Parameters

thread_ptr	Pointer to thread to change the core exclusion.
exclusion_map	Bit map where a sit bit indicates that that core is excluded. Supplying a 0 value enables the thread to execute on any core (default).

Return Values

TX_SUCCESS	(0x00)	Successful core exclusion.
TX_THREAD_ERROR	(0x0E)	Invalid thread pointer.

Allowed From

Initialization, ISRs, threads, and timers

Example

```
/* Exclude core 0 for "Thread 0".  */
tx_thread_smp_core_exclude(&thread_0, 0x01);
```

See Also

tx_thread_smp_core_exclude_get, tx_thread_smp_core_get

tx_thread_smp_core_exclude_get

Gets the thread's current core exclusion

Prototype

```
UINT  tx_thread_smp_core_exclude_get(TX_THREAD *thread_ptr,
                                      ULONG *exclusion_map_ptr);
```

Description

This function returns the current core exclusion list.

Parameters

thread_ptr	Pointer to thread from which to retrieve the core exclusion.
exclusion_map_ptr	Destination for current core exclusion bit map.

Return Values

TX_SUCCESS	(0x00)	Successful retrieval of thread's core exclusion.
TX_THREAD_ERROR	(0x0E)	Invalid thread pointer.
TX_PTR_ERROR	(0x03)	Invalid exclusion destination pointer.

Allowed From

Initialization, ISRs, threads, and timers

Example

```
ULONG excluded_cores;

/* Retrieve the core exclusion for "Thread 0".  */
tx_thread_smp_core_exclude_get(&thread_0,
                                     &excluded_cores);
```

See Also

tx_thread_smp_core_exclude, tx_thread_smp_core_get

tx_thread_smp_core_get

Retrieve the currently executing core of caller

Prototype

```
UINT  tx_thread_smp_core_get(void);
```

Description

This function returns the core ID of the core executing this service.

Parameters

None

Return Values

`core_id`	ID of currently executing core, (0 through TX_THREAD_SMP_MAX_CORES-1)

Allowed From

Initialization, ISRs, threads, and timers

Example

```
UINT  core;

/* Pickup the currently executing core.  */
core =  tx_thread_smp_core_get();

/* At this point, "core" contains the executing core ID.  */
```

See Also

`tx_thread_smp_core_exclude, tx_thread_smp_core_exclude_get`

tx_timer_smp_core_exclude

Exclude timer execution on a set of cores

Prototype

```
UINT  tx_timer_smp_core_exclude(TX_TIMER *timer_ptr,
                                  ULONG exclusion_map);
```

Description

This function excludes the specified timer from executing on the core(s) specified in the bit map called "*exclusion_map*." Each bit in "*exclusion_map*" represents a core (bit 0 represents core 0, etc.). If the bit is set, the corresponding core is excluded from executing the specified timer.

Parameters

timer_ptr	Pointer to timer to change the core exclusion.
exclusion_map	Bit map where a sit bit indicates that that core is excluded. Supplying a 0 value enables the timer to execute on any core (default).

Return Values

TX_SUCCESS	(0x00)	Successful core exclusion.
TX_TIMER_ERROR	(0x0E)	Invalid timer pointer.

Allowed From

Initialization, ISRs, threads, and timers

Example

```
/* Exclude core 0 for "Timer 0".  */
tx_timer_smp_core_exclude(&timer_0, 0x01);
```

See Also

tx_timer_smp_core_exclude_get

tx_timer_smp_core_exclude_get

Gets the timer's current core exclusion

Prototype

```
UINT  tx_timer_smp_core_exclude_get(TX_TIMER *timer_ptr,
                                    ULONG *exclusion_map_ptr);
```

Description

This function returns the current core exclusion list.

Parameters

timer_ptr	Pointer to timer from which to retrieve the core exclusion.
exclusion_map_ptr	Destination for current core exclusion bit map.

Return Values

TX_SUCCESS	(0x00)	Successful retrieval of timer's core exclusion.
TX_TIMER_ERROR	(0x0E)	Invalid timer pointer.
TX_PTR_ERROR	(0x03)	Invalid exclusion destination pointer.

Allowed From

Initialization, ISRs, threads, and timers

Example

```
ULONG excluded_cores;

/* Retrieve the core exclusion for "Timer 0".  */
tx_timer_smp_core_exclude_get(&timer_0,
                              &excluded_cores);
```

See Also

tx_timer_smp_core_exclude

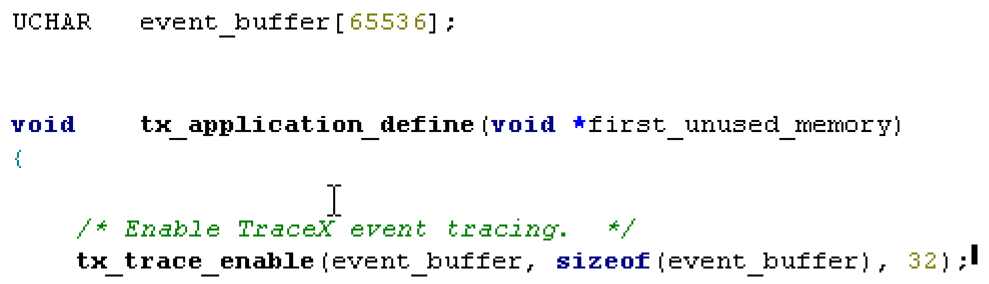
Chapter 5

ThreadX SMP Debug Capabilities

Under application control, ThreadX SMP can generate a multicore TraceX buffer. TraceX is a host-based tool that graphically displays system events on each core, giving multicore developers unmatched visibility into their applications.

Enabling TraceX Multicore

ThreadX SMP is easily enabled. First, build the entire ThreadX SMP library and application code with ***TX_ENABLE_EVENT_TRACE***. Doing this will enable trace capture throughout ThreadX execution. The next step is to define a trace buffer inside the application and enable tracing. This is often done inside of ***tx_application_define***, as the following code example in Figure 5.1 shows:



```
UCHAR   event_buffer[65536];


void    tx_application_define(void *first_unused_memory)
{

    /* Enable TraceX event tracing.  */
    tx_trace_enable(event_buffer, sizeof(event_buffer), 32);
```

Figure 5.1

Once enabled, all calls to ThreadX and internal system events (context switches, etc.) are logged in a circular trace buffer. As shown above, the size of the buffer is determined by the application. To view the trace buffer via TraceX, the trace buffer must first be exported from the target to the host. Please review the ***TraceX User Guide*** for more information on exporting the trace buffer.

Multicore Trace File Splitter Utility

In order to use a ThreadX SMP multicore trace buffer with TraceX, a DOS command line tool must be used. This tool splits the trace buffer into separate trace buffers   one for each core detected in the trace file   and launches TraceX for each of core-specific trace files. Invoking the ThreadX SMP multicore trace buffer utility is done as follows:

```
> tracesplit  c:\full_path\trace_file.trx  tracex_install_path<cr>
```

For example, in order to view the trace events in the file ***demo_threadx_smp.trx*** that resided in the directory ***c:\temp***, using the default installation path for ***TraceX 5.2.0***, the following command line should be used:

```
> tracesplit c:\temp\demo_threadx_smp.trx c:\Express_Logic\TraceX_5.2.0 <cr>
```

The result of this command are two new trace files, ***CORE-0_demo_threadx_smp.trx*** and ***CORE-1_demo_threadx_smp.trx***, each displayed in a separate instance of ***TraceX 5.2.0***.