Lock handling library

IN THIS DOCUMENT

This module provides access to hardware and software locks for use in concurrent C programs. In general it is not safe to use these to marshall within XC due to the assumptions XC makes about safe concurrent data access.

Two types of locks are provided. Hardware locks are fast and power efficient but there are a limited number per tile. Software locks are slower but you can use an unlimited number of them.

1 Hardware lock API

`hwlock_t`

This type represents a hardware lock.

`hwlock_t hwlock_alloc(void)`

Allocate a hardware lock.

This function will allocate a new hardware lock from the pool of hardware locks available on the xCORE. The hardware has a limited number of hardware locks (for example, current L and S series devices have 4 locks per tile).

This function returns:

the allocated lock if allocation is successful or the value `HWLOCK_NOT_ALLOCATED` if not.

`void hwlock_free(hwlock_t lock)`

Free a hardware lock.

This function frees a given hardware lock and returns it to the hardware pool to be reallocated elsewhere.

This function has the following parameters:

`lock` — the hardware lock to be freed. If this is an invalid lock id or not an currently allocated lock then the function will trap.

`void hwlock_acquire(hwlock_t lock)`

Acquire a hardware lock.

This function acquires a lock for the current logical core. If another core holds the lock the function will pause until the lock is released.

This function has the following parameters:

`lock` the hardware lock to acquire

`void hwlock_release(hwlock_t lock)`

Release a hardware lock.

This function releases a lock from the current logical core. The lock should have been previously claimed by hwlock_acquire().

This function has the following parameters:

`lock` the hardware lock to release

2 Software lock API

`swlock_t`

Type that represents a software lock.

`SWLOCK_INITIAL_VALUE`

This define should be used to initialize a software lock e.g.

```
swlock_t my_lock = SWLOCK_INITIAL_VALUE;
```

If you intialize this way there is no need to call swlock_init().

`void swlock_init(swlock_t &lock)`

Initialize a software lock.

This function will initialize a software lock for use. Note that unlike hardware locks, there is no need to allocate or free a software lock from a limited pool.

`int swlock_try_acquire(swlock_t &lock)`

Try and acquire a software lock.

This function tries to acquire a lock for the current logical core. If another core holds the lock then the function will fail and return.

This function has the following parameters:

`lock` the software lock to acquire.

This function returns:

a value that is equal to `SWLOCK_NOT_ACQUIRED` if the attempt fails. Any other value indicates that the acquisition has succeeded.

`void swlock_acquire(swlock_t &lock)`

Acquire a software lock.

This function acquires a lock for the current logical core. If another core holds the lock then the function will wait until it becomes available.

This function has the following parameters:

`lock` the software lock to acquire.

```
void swlock_release(swlock_t &lock)
```

Release a software lock.

This function releases a previously acquired software lock for other cores to use.

This function has the following parameters:

`lock` the software lock to release.