IPC Users Guide/SharedRegion Module

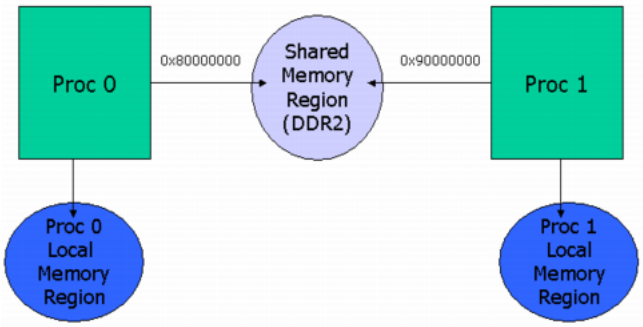
< IPC Users Guide

Contents

API Reference Links

The SharedRegion module is designed to be used in a multi-processor environment where there are memory regions that are shared and accessed across different processors.

In an environment with shared memory regions, a common problem is that these shared regions are memory mapped to different address spaces on different processors. This is shown in the following figure. The shared memory region "DDR2" is mapped into Proc0's local memory space at base address 0x80000000 and Proc1's local memory space at base address 0x90000000. Therefore, the pointers in "DDR2" need to be translated in order for them to be portable between Proc0 and Proc1. The local memory regions for Proc0 and Proc1 are not shared thus they do not need to be added to the SharedRegion module.



On systems where address translation is not required, translation is a noop, so performance is not affected.

The SharedRegion module itself does not use any shared memory, because all of its state is stored locally. The APIs use the system gate for thread protection.

This module creates a shared memory region lookup table. The lookup table contains the processor's view of every shared region in the system. In cases where a processor cannot view a certain shared memory region, that shared memory region should be left invalid for that processor. Each processor has its own lookup table.

Each processor's view of a particular shared memory region can be determined by the same region ID across all lookup tables. At runtime, this table, along with the shared region pointer, is used to do a quick address translation.

The lookup table contains the following information about each shared region:

- **base.** The base address of the region. This may be different on different processors, depending on their addressing schemes.
- **len.** The length of the region. This should be should be the same across all processors.
- **ownerProcId.** MultiProc ID of the processor that manages this region. If an owner is specified, the owner creates a HeapMemMP instance at runtime. The other cores open the same HeapMemMP instance.
- **isValid.** Boolean to specify whether the region is valid (accessible) or not on this processor.
- **cacheEnable.** Boolean to specify whether a cache is enabled for the region on the local processor.
- **cacheLineSize.** The cache line size for the region. It is *crucial* that the value specified here be the same on all processors.
- **createHeap.** Boolean to specify if a heap is created for the region.
- **name.** The name associated with the region.

The maximum number of entries in the lookup table is statically configurable using the SharedRegion.numEntries property. Entries can be added during static configuration or at runtime. When you add or remove an entry in one processor's table, you must update all of the remaining processors' tables to keep them consistent. The larger the maximum number of entries, the longer it will take to traverse the lookup table when searching for the index. Therefore, keep the lookup table small for better performance and footprint.

Because each processor stores information about the caching of a shared memory region in the SharedRegion lookup table, other modules can (and do) make use of this caching information to maintain coherency and alignment when using items stored in shared memory.

In order to use the SharedRegion module, the following must be true:

- The SharedRegion.numEntries property must be the same on all processors.
- The size of a SharedRegion pointer is 32-bits wide.
- The SharedRegion lookup table must contain at least 1 entry for address translation to occur.
- Shared memory regions must not overlap each other from a single processor's viewpoint.
- Regions are not allowed to overlap from a single processor's view.

- The SharedRegion with an index of 0 (zero) is used by IPC_start() to create resource management tables for internal use by other IPC modules. Thus SharedRegion "0" must be accessible by all processors. Your applications can also make use of SharedRegion "0", but must be aware of memory limitations.

> **NOTE**
>
> The SharedRegion module is not used on Concerto F28M35x devices. Instead, the IpcMgr module (in the ti.sdo.ipc.family.f28m35x package) is used to configure access to shared memory by Concerto devices. See Using IPC on Concerto Devices.

Adding Table Entries Statically

To create a shared region lookup table in the XDCtools configuration, first determine the shared memory regions you plan to use.

Next, specify the maximum number of entries in the lookup table with the SharedRegion.numEntries property. You can specify a value for the SharedRegion.cacheLineSize configuration property, which is the default cache line size if no size is specified for a region. You can also specify the value of the SharedRegion.translate property, which should only be set to false if all shared memory regions have the same base address on all processors. Setting the translate property to false improves performance because no address translation is performed. For example:

<syntaxhighlight lang='javascript'> var SharedRegion = xdc.useModule('ti.sdo.ipc.SharedRegion'); SharedRegion.cacheLineSize = 32; SharedRegion.numEntries = 4; SharedRegion.translate = true; </syntaxhighlight>

Then, use the SharedRegion.setEntryMeta() method in the configuration file to specify the parameters of the entry.

<syntaxhighlight lang='c'>var SHAREDMEM = 0x0C000000; var SHAREDMEMSIZE = 0x00200000;

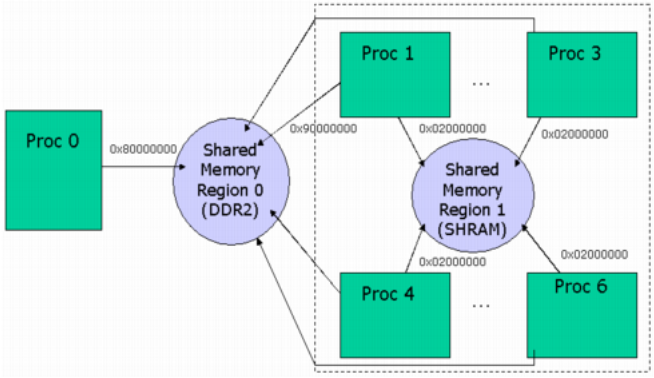
SharedRegion.setEntryMeta(0,

```
{ base: SHAREDMEM,
  len: SHAREDMEMSIZE,
  ownerProcId: 0,
  isValid: true,
  cacheEnable: true,
  cacheLineSize: 128,
  createHeap: true,
  name: "internal_shared_mem"
```

}); </syntaxhighlight>

If, during static configuration, you don't know the base address for every processor, you should set the "isValid" field for an entry for which you don't yet know the base address to "false". Storing this information will allow it to be completed at runtime.

The following figure shows the configuration of a SharedRegion table for the system in the following figure. This system has seven processors and two shared memory regions. Region 0 ("ext") is accessible by all processors. Region 1 ("local") is accessible only by processors 1 to 6.



If the "createHeap" field is set to true, a HeapMemMP instance is created within the SharedRegion.

The latest version of the SharedRegion module configuration documentation is available online (http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/ipc/latest/docs/cdoc/index.html#ti/sdo/ipc/SharedRegion.html).

Modifying Table Entries Dynamically

In the application's C code, a shared memory region can be modified in the SharedRegion table by calling `SharedRegion_setEntry()`.

Typically, applications configure SharedRegion table entries statically as described in the previous section, and only modify the table entries dynamically in applications where it is possible for shared memory region availability to change dynamically.

The call to `SharedRegion_setEntry()` must specify all the fields in the SharedRegion_Entry structure. The index specified must be the same across all processors for the same shared memory region. The index also must be smaller than the `maxNumEntries` property, otherwise an assert will be triggered.

<syntaxhighlight lang='c'> typedef struct SharedRegion_Entry {

```
Ptr base;
SizeT len;
UInt16 ownerProcId;
Bool isValid;
Bool cacheEnable;
SizeT cacheLineSize;
```

```
  Bool createHeap;
  String name;
```

} SharedRegion_Entry; </syntaxhighlight>

You can use `SharedRegion_getEntry()` to fill the fields in a SharedRegion_Entry structure. Then, you can modify fields in the structure and call `SharedRegion_setEntry()` to write the modified fields back to the SharedRegion table.

If you want to reuse an index location in the SharedRegion table, you can call `SharedRegion_clear()` on all processors to erase the existing entry at that index location.

The latest version of the SharedRegion module run-time API documentation is available online (http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/ipc/latest/docs/doxygen/html/_shared_region_8h.html).

Using Memory in a Shared Region

Note that the SharedRegion with an index of 0 (zero) is used by `IPC_start()` to create resource management tables for internal use by the GateMP, NameServer, and Notify modules. Thus SharedRegion "0" must be accessible by all processors.

This example allocates memory from a SharedRegion:

<syntaxhighlight lang='c'> buf = Memory_alloc(SharedRegion_getHeap(0), sizeof(Tester) * COUNT, 128, NULL); </syntaxhighlight>

Getting Information About a Shared Region

The shared region pointer (SRPtr) is a 32-bit portable pointer composed of an ID and offset. The most significant bits of a SRPtr are used for the ID. The ID corresponds to the index of the entry in the lookup table. The offset is the offset from the base of the shared memory region. The maximum number of table entries in the lookup table determines the number of bits to be used for the ID. An increase in the id means the range of the offset would decrease. The ID is limited to 16-bits.

Here is sample code for getting the SRPtr and then getting the real address pointer back.

<syntaxhighlight lang='c'> SharedRegion_SRPtr srptr; UInt16 id;

// Get the id of the address if id is not already known. id = SharedRegion_getId(addr);

// Get the shared region pointer for the address srptr = SharedRegion_getSRPtr(addr, id);

// Get the address back from the shared region pointer addr = SharedRegion_getPtr(srptr); </syntaxhighlight>

In addition, you can use the `SharedRegion_getIdByName()` function to pass the name of a SharedRegion and receive the ID number of the region.

You can use `SharedRegion_getHeap()` to get a handle to the heap associated with a region using the heap ID.

You can retrieve a specific shared region's cache configuration from the SharedRegion table by using `SharedRegion_isCacheEnabled()` and `SharedRegion_getCacheLineSize()`.

> **NOTE**
>
> In order to account for the case where the contents of the SharedRegion with an index of 0 cannot be known before the cores are booted, it is recommended to place the SharedRegion 0 owner procId in the MultiProc config array before all other cores that will be attaching to the owner of SharedRegion 0. In this way, it is guaranteed that the SharedRegion 0 owner is able to zero-init the memory before other cores start to accesses it during Ipc_attach. For example, if IPU1 is the SharedRegion 0 owner, the order would look like this in order to put IPU1 first in the array: ["IPU1","EVE1","DSP1"]

> **NOTE**
>
> The SharedRegion Module is only supported in SYS/BIOS environments. It is not provided on HLOS's. For HLOS environments, we suggest using native shared memory APIs when available, for example ION on Android. Other alternatives include CMEM (for Linux) and the QNX-specific SharedMemoryAlloctor, provided in IPC's qnx/ directory.

here.

Please post only comments related to the article **IPC Users Guide/SharedRegion Module** here.

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters
DLP & MEMS
High-Reliability
Interface
Logic
Power Management
Processors

- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity