libcodecengine Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Library Overview 3
Features 3
Files 3
Sample Programs 3
Reference Materials 3
2 Usage 4
Basic Procedure (Allocating and Releasing Memory for Libraries Using the Codec Engine) 4
Basic Procedure (Measuring the Processor Load for Codec Engine Debugging) 5
3 Reference Information 6
Codec Engine 6
4 Notes 7
Memory Passed to `sceCodecEngineOpenUnmapMemBlock()` 7
Memory Allocated with `sceCodecEngineAllocMemoryFromUnmapMemBlock()` 7
Using for the title submission 7
Impact of measurement on execution 7
How to Use Measurement Results 7

1 Library Overview

Features

The libcodecengine library provides operations related to the Codec Engine media processor. The APIs provided by libcodecengine are divided into APIs that can operate using retail units, Development Kits (DevKit) or Testing Kits (TestKit) and debug APIs that operate using DevKits or TestKits. The former are APIs that allocate and release memory for libraries using the Codec Engine, and the latter are APIs that measure the processor load of the Codec Engine.

The following libraries use the Codec Engine.

- libaudiodec
- libaudioenc
- scejpegdec
- scejpegenc
- Video Player Library
- NGS
- libvoice

Files

The files required to use libcodecengine are as follows.

Filename	Description
codecengine.h	Header file
libSceCodecEngine_stub.a	Stub library file
libSceCodecEnginePerf_stub.a	Stub library file (For debug APIs)
libSceCodecEnginePerf_stub_weak.a	weak import stub library file (For debug APIs)

Sample Programs

The following program is provided as a sample libcodecengine program for reference purposes.

samples/sample_code/audio_video/api_libcodecengine/libaudiodec/

This is a sample for obtaining the Codec Engine processor load when the api_libaudiodec/on_memory sample is being run.

Reference Materials

Refer to the following documents for libraries using Codec Engine.

- libaudiodec Overview
- libaudioenc Overview
- JPEG Decoder Overview
- JPEG Encoder Overview
- Video Player Library Overview
- NGS Overview
- libvoice Overview

2 Usage

Basic Procedure (Allocating and Releasing Memory for Libraries Using the Codec Engine)

(1) Allocate a cache-disabled and physical continuous memory that is enabled for reading and writing by the user

Use `sceKernelAllocMemBlock()` to allocate a cache-disabled and physical continuous memory that is enabled for reading and writing by the user. In the case allocating memory on LPDDR2 specify `SCE_KERNEL_MEMBLOCK_TYPE_USER_MAIN_PHYCONT_NC_RW` to the argument and in the case allocating memory on the CDRAM specify `SCE_KERNEL_MEMBLOCK_TYPE_USER_CDRAM_RW` to the argument.

(2) Remap as a cache-disabled and physical continuous memory that is enabled for reading and writing by the Codec Engine

Use `sceCodecEngineOpenUnmapMemBlock()` to unmap the cache-disabled and physical continuous memory that is enabled for reading and writing by the user, and then remap as a cache-disabled and physical continuous memory that is enabled for reading and writing by the Codec Engine but not by the user.

(3) Allocate memory with the remapped memory area as a heap

Use `sceCodecEngineAllocMemoryFromUnmapMemBlock()` to treat the cache-disabled and physical continuous memory that is enabled for reading and writing by the Codec Engine but not by the user as a heap and then allocate the memory.

(4) Use the allocated memory with the libraries using the Codec Engine

Pass the allocated memory to the libraries that use the Codec Engine, such as a video player library or libaudiodec, and use that memory. Multiple allocated memories can be passed to different libraries. See the library documents for information on using memory allocated with libcodecengine.

(5) Release the allocated memory

After completing use of a memory for a library using the Codec Engine, release the memory with `sceCodecEngineFreeMemoryFromUnmapMemBlock()`.

(6) Remap as a cache-disabled and physical continuous memory that is enabled for reading and writing by the user

Use `sceCodecEngineCloseUnmapMemBlock()` to unmap the cache-disabled and physical continuous memory that is enabled for reading and writing by the Codec Engine but not by the user, and then remap as a cache-disabled and physical continuous memory that is enabled for reading and writing by the user.

(7) Release the cache-disabled and physical continuous memory that is enabled for reading and writing by the user

Use `sceKernelFreeMemBlock()` to release the memory to a cache-disabled and physical continuous memory area that is enabled for reading and writing by the user.

List of APIs

The following are the APIs provided by libcodecengine. For details, refer to the "libcodecengine Reference" document.

API	Description
`sceCodecEngineOpenUnmapMemBlock()`	Remaps as memory that is enabled for reading and writing by the Codec Engine but not by the user.
`sceCodecEngineCloseUnmapMemBlock()`	Remaps as a memory that is enabled for reading and writing by the user.
`sceCodecEngineAllocMemoryFromUnmapMemBlock()`	Allocates memory from a memory area that is enabled for reading and writing by the Codec Engine but not by the user.
`sceCodecEngineFreeMemoryFromUnmapMemBlock()`	Releases memory to a memory area that is enabled for reading and writing by the Codec Engine but not by the user.

Basic Procedure (Measuring the Processor Load for Codec Engine Debugging)

(1) Load the module

libcodecengine debug APIs are provided as the PRX module. First, use `sceSysmoduleLoadModule(SCE_SYSMODULE_CODECENGINE_PERF)` to load the libcodecengine module for debugging.

(2) Measuring Performance

Start measuring Codec Engine performance by calling `sceCodecEnginePmonStart()` immediately before the segment to be measured. Measurement will continue until `sceCodecEnginePmonStop()` is called. When multiple segments are used to measure performance, the measurement results will be accumulated.

(3) Obtaining Measurement Results

By calling `sceCodecEnginePmonGetProcessorLoad()`, the Codec Engine processor load within the measurement segment can be obtained. Call `sceCodecEnginePmonReset()` to reset measurement results.

(4) Unload the module

Use `sceSysmoduleUnloadModule(SCE_SYSMODULE_CODECENGINE_PERF)` to unload the libcodecengine module for debugging.

List of Debug APIs

The following are the debug APIs provided by libcodecengine. For details, refer to the "libcodecengine Reference" document.

API	Description
`sceCodecEnginePmonStart()`	Starts measuring Codec Engine performance
`sceCodecEnginePmonStop()`	Stops measuring Codec Engine performance
`sceCodecEnginePmonGetProcessorLoad()`	Obtains Codec Engine processor load
`sceCodecEnginePmonReset()`	Resets Codec Engine performance measurement results

3 Reference Information

Codec Engine

The Codec Engine is a media processor composed of several cores. Only specific processes of specific libraries will be processed on the Codec Engine and not on the CPU. Refer to the Library Overview chapter for libraries using Codec Engine.

When executing multiple APIs using the Codec Engine, requests from the CPU to the Codec Engine are queued and executed in parallel as much as the design will allow. At this time, threads used to call these APIs are independently blocked until processing on the Codec Engine has terminated. Therefore, APIs executed on a single thread on the CPU side are executed sequentially, whereas the completion of APIs executed on multiple threads in parallel depend on the process load and the order of their completion will not necessarily be sequential. For specific APIs, processing is divided into several Codec Engine cores and executed in parallel even if a single API is called.

4 Notes

Memory Passed to `sceCodecEngineOpenUnmapMemBlock()`

Specify all memory areas allocated with `sceKernelAllocMemBlock()` to `sceCodecEngineOpenUnmapMemBlock()`. This is to unmap all regions specified as regions not enabled for reading or writing by the user. When only part of the allocated memory area is specified, `sceCodecEngineOpenUnmapMemBlock()` returns an error.

Memory Allocated with `sceCodecEngineAllocMemoryFromUnmapMemBlock()`

The memory allocated with `sceCodecEngineAllocMemoryFromUnmapMemBlock()` must be passed to the memory for the libraries using the Codec Engine. This is for managing within the libraries the fact that the specified memory area is in use by the Codec Engine. When part of the memory area remapped with `sceCodecEngineOpenUnmapMemBlock()` is specified without using `sceCodecEngineAllocMemoryFromUnmapMemBlock()`, the APIs provided by the various libraries return an error.

Using for the title submission

To ensure future compatibility, debug APIs cannot be used for the title submission. The debug APIs operate on a DevKit or TestKit, but an error occurs when executed on a retail unit.

Impact of measurement on execution

The APIs for measuring the processor load of Codec Engine may take some time for their measurements due to internal systems calls. Provide the appropriate length for the measurement segment so that the measurement load does not affect measurement results.

How to Use Measurement Results

The processor load to be measured using `sceCodecEnginePmonGetProcessorLoad()` can be obtained as a relative value to the states where all the processing resources on all Codec Engine cores are used. Units are measured in percentages. Using the measurement results, this example shows how to estimate the number of NGS playable voices for a specific NGS configuration.

If NGS voices can be played back continuously, the following is true.

(CPU process time + Codec Engine process time) < NGS system update interval

CPU process time is measured using libperf, and Codec Engine process time is measured using process time. If we assume that the CPU process time is constant and independent of the number of playable voices on the NGS system update thread, the number of playable voices can be estimated by observing the Codec Engine process time relative to the difference in the number of play voices.

Two points should be noted here: First, although the number of play voices and Codec Engine process time are positive correlated, they are not directly proportional to each other. This is due to the fact that Codec Engine processes are run on multiple cores. On the other hand, the Codec Engine process load measured by libcodecengine and the number of play voices are more or less directly proportional to one another.

Secondly, the number of play voices can not necessarily be increased until the processor load measured by libcodecengine reaches 100%.The process load value measured by libcodecengine is a relative value to the states where all the processing resources on all Codec Engine cores are used. Ideally, the process load can reach up to 100%, but not all run requests will be filled until they reach 100% of the load. This is due to the

dependency among multiple run units on the Codec Engine, which implies that not all processes can be run in parallel.

The number of playable voices on the NGS can be estimated with the above points in mind.

(1) Based on the current NGS configuration, increase the number of play voices to the desired number.

(2) Ensure that the sum of the CPU process time and the Codec Engine process time on the NGS system update thread is less than the NGS system update time interval. If conditions have been met, terminate the estimation. If conditions have not been met, ensure that file access or other CPU process times are not the cause. If the Codec Engine process time is longer, go to Step (3).

(3) Modify the Codec Engine process load so that it is less than 100% by decreasing the number of desired play voices, or by modifying the NGS configuration (modules, racks, parameters). If the Codec Engine process load is less than 100% from the beginning, decrease the load again. Then go back to Step (2).