libperf Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Library Overview

Characteristics

libperf is a library that uses Performance Monitor functions on the ARM Processor to analyze the program's performance. With libperf, part of the program can be specified explicitly for measurement, allowing the event counter and cycle counter values for that part to be obtained from the ARM Processor. These values can be used to analyze the program's performance.

As Performance Monitor functions, the ARM Processor has six 32-bit event counters and one 32-bit cycle counter.

Please note that Performance Monitor resources such as event selection, measurement start/end states, event counter and cycle counter values are saved and restored as part of the thread context.

Files

Files required for using libperf are as follows.

Filename	Description
libperf.h	Header file
libScePerf_stub.a	Stub library file
libScePerf_stub_weak.a	weak import stub library file

Sample Programs

Sample programs for libperf are as follows.

sample_code/developer_tools/api_libperf/basic/

This sample program shows the basic usage of libperf.

Reference Materials

Refer to the following documents for ARM Performance Monitor events.

ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition

"C9.10 Event numbers" describes event information defined in ARMv7 architecture.

Cortex-A9 Technical Reference Manual Revision: r3p0

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0388g/BEHDIGBF.html

"11.4.1 Cortex-A9 specific events" describes event information defined specifically in Cortex-A9.

Events up to Cortex-A9 Revision r3p0 are supported as Cortex-A9 specific events.

(The above reference destination has been confirmed as of January 6, 2015. Note that pages may have been subsequently moved or its contents modified.)

2 Using the Library

Basic Procedure

(1) Load module

libperf is provided as a PRX module. First, use `sceSysmoduleLoadModule(SCE_SYSMODULE_PERF)` to load the libperf module.

(2) Select Event

When using an event counter, call `scePerfArmPmonSelectEvent()` and select the Performance Monitor event of your choice.

(3) Measure Performance

Call `scePerfArmPmonStart()` immediately before the section that needs to be measured. Measurement will begin and continue until `scePerfArmPmonStop()` is called. An event called Software Increment is defined as a Performance Monitor function on the ARM Processor. By calling `scePerfArmPmonSoftwareIncrement()` during performance measurement, the event counter value set in the Software Increment event can be incremented.

(4) Obtain Measurement Results

By calling `scePerfArmPmonGetCounterValue()`, a specific event counter or cycle counter value can be obtained.

(5) Unload module

Use `sceSysmoduleUnloadModule(SCE_SYSMODULE_PERF)` to unload libperf.

List of Functions

libperf functions are listed below.

Function	Description
`scePerfArmPmonReset()`	Resets event counter and cycle counter
`scePerfArmPmonSelectEvent()`	Selects Performance Monitor event
`scePerfArmPmonStart()`	Starts measurement
`scePerfArmPmonStop()`	Stops measurement
`scePerfArmPmonGetCounterValue()`	Gets counter value
`scePerfArmPmonSetCounterValue()`	Sets counter value
`scePerfArmPmonSoftwareIncrement()`	Increments Software Increment event
`scePerfGetTimebaseValue()`	Gets timebase value
`scePerfGetTimebaseFrequency()`	Gets timebase frequency
`sceRazorCpuPushMarker()`	Pushes a marker (deprecated)
`sceRazorCpuPopMarker()`	Pops a marker
`sceRazorCpuStartCapture()`	Starts Razor CPU capture
`sceRazorCpuStopCapture()`	Stops Razor CPU capture
`sceRazorCpuSync()`	Custom synchronization point for Razor
`sceRazorCpuIsCapturing()`	Queries if host tool is doing a CPU capture
`sceRazorCpuPushMarkerWithHud()`	Pushes a marker with support for the Razor HUD
`sceRazorCpuStartUserMarkerTrace()`	Assigns buffer for user marker trace, and start tracing
`sceRazorCpuStopUserMarkerTrace()`	Stops tracing user markers.
`sceRazorCpuStartActivityMonitor()`	Assigns buffer for a CPU activity monitor trace, and start tracing.
`sceRazorCpuStopActivityMonitor()`	Stops tracing CPU activity monitor.
`sceRazorCpuGetUserMarkerTraceBuffer()`	Gets the HUD user marker trace buffer.
`sceRazorCpuGetActivityMonitorTraceBuffer()`	Gets the CPU activity monitor trace buffer.

3 Notes

Using for Title Submission

To ensure future compatibility, libperf cannot be used for title submission.

Impact of Measurement on Execution

Many of the functions provided in libperf may take some time to control the Performance Monitor, particularly when they are used for threads on other processes, due to internal systems calls. Please ensure that this time does not affect the performance measurement.

Resources Saved in Thread Context

When the thread is context switched out of the RUNNING state, then the cycle and performance counters are saved with the thread context. When the thread is rescheduled then the cycle and performance counters are restored.

When the thread is terminated then restarted, the usual thread context is initialized to the thread creation state. But Performance Monitor resources such as event selection and measurement start/end states are not initialized and maintain their previous condition, making these resources available for dormant threads.

4 Known Hardware Defects in the ARM Performance Monitor Event

ISB Is Counted in Performance Monitor Events 0x0C and 0x0D

The ISB is implemented as a branch in the Cortex-A9 microarchitecture.

This implies that events 0x0C (software change of PC) and 0x0D (immediate branch) are asserted when an ISB occurs, which is not compliant with the ARM Architecture.

Implications

The count of events 0x0C and 0x0D are not 100% precise when using the Performance Monitor counters, due to the ISB being counted in addition to the real software changes of PC (0x0C) and immediate branches (0x0D).

Workaround

You can count ISB instructions alone with event 0x90.

You can subtract this ISB count from the results you obtained in events 0x0C and 0x0D, to obtain the precise count of software change of PC (0x0C) and immediate branches (0x0D).

Event 0x86, 0x91 and 0x92 May Be Different from Expected

Because of ARM hardware defect, the DMB instruction is configured to be decoded and executed like a DSB instruction. As a consequence, executing DMB instruction will be counted as a DSB instruction.

Implications

The event 0x86 (the number of stall cycle due to DMB), 0x91 (the number of DSB) and 0x92 (the number of DMB) may be different from expected.

MRC and MCR Instructions Are Not Counted in Event 0x68

Event 0x68 counts the total number of instructions passing through the Register rename pipeline stage. The erratum is that MRC and MCR instructions are not counted in this event.

Implications

The value of event 0x68 is imprecise, omitting the number of MRC and MCR instructions. The inaccuracy of the total count depends on the rate of MRC and MCR instructions in the code.