IBM

Software Development Kit for Multicore Acceleration, Version 3.0

SPU Timer Library Programmer's Guide and API Reference

Note: Before using this information and the product it supports, read the information in “Notices”.

First Edition (October 2007)
This edition applies to the version 3, release 0, of the IBM Software Development Kit for Multicore Acceleration and to all subsequent releases and modifications until otherwise indicated in new editions.

Table of Contents

PREFACE

About this publication

This book describes the SPU timer library for Cell Broadband Engine™ (Cell BE™) that is shipped as part of the IBM® Software Development Kit for Multicore Acceleration, Version 3.0. It describes the programming interfaces and programming environment for using the timer library on the Cell BE™ Synergistic Processing Unit (SPU).

Intended audience

This document is intended for application or performance tools developers who are interested in using the SPU timer library for performance sampling and measurements. It assumes the reader is familiar with Cell BE™ and SPU programming.

Related Information

The following publications provide most of the required background information needed for using the SPU Timer library:

- *Cell Broadband Engine Programming Handbook*
- *SDK for Multicore Acceleration Version 3.0 Programmer's Guide*

Getting help and technical assistance

If you need help, service, or technical assistance or just want more information about IBM products, you will find a wide variety of sources available from IBM to assist you. This appendix contains information about where to go for additional information about IBM and IBM products and whom to call for service, if it is necessary.

Using the documentation

Information about your IBM hardware or software is available in the documentation that comes with the product. That documentation can include printed documents, online documents, readme files, and help files. See the troubleshooting information in your documentation for instructions for using

diagnostic programs. The troubleshooting information or the diagnostic programs might tell you that you need additional or updated device drivers or other software. IBM maintains pages on the World Wide Web where you can get the latest technical information and download device drivers and updates. To access these pages, go to http://www.ibm.com/bladecenter/, click **Support**, and follow the instructions. Also, some documents are available through the IBM Publications Center at http://www.ibm.com/shop/publications/order/.

Getting help and information from the World Wide Web

You can locate documentation and other resources on the World Wide Web. Refer to the following web sites:

- IBM BladeCenter systems, optional devices, services, and support information at http://www.ibm.com/bladecenter/. For service information, select **Support**.
- developerWorks® Cell/B.E. Resource Center at http://www.ibm.com/developerworks/power/cell/. To access the Cell/B.E. forum on developerWorks, select **Community**.
- The Barcelona Supercomputing Center (BSC) Web site at http://www.bsc.es/projects/deepcomputing/linuxoncell.
- There is also support for the Full-System Simulator and XL C/C++ Compiler through their individual alphaWorks® forums. If in doubt, start with the Cell/B.E. architecture forum.
- The GNU Project debugger, GDB, is supported through many different forums on the Web, but primarily at the GDB Web site http://www.gnu.org/software/gdb/gdb.html.

Contacting IBM support

To obtain telephone assistance, for a fee or on a support contract, contact IBM Support. In the U.S. and Canada, call 1-800-IBM-SERV (1-800-426-7378), or see http://www.ibm.com/planetwide/ for support telephone numbers.

Part I. Overview

Introduction

The SPU timer library provides virtual clock and interval timer services for SPU programs. The virtual clock is a 64-bit software managed, monotonically increasing *Time Base* counter. The interval timers provide the ability to register a user-defined handler to be called on a specified interval.

By virtualizing the *decrementer* register, the SPU timer library provides the ability to perform high-precision time measurements, while simultaneously activating one or more interval timers, which can be used for statistical, sample-based profiling.

Terminology

The following terms are used throughout this document:

FLIH

First-Level Interrupt Handler – code that is branched to by the processor in response to an interrupt. SPU programs that enable interrupts must place their interrupt handler code at fixed address 0x0. By default, code specified in the ".interrupt" section of the object is placed at this address.

SLIH

Second-Level Interrupt Handler – code that by convention services a specific interrupt type and is called by the *FLIH*

Time Base

A hardware register defined by the PowerPC® architecture which represents an elapsed time. It is a monotonically increasing counter that ticks at an implementation-specific *Time Base* frequency.

Decrementer

A hardware register defined by the PowerPC® architecture which is available on the Cell BE™ PPU and SPU. This register counts down from its programmed value at the *Time Base* frequency and generates an interrupt or event when the count has expired. On the SPU, the *decrementer* is a 32-bit, user-programmable register.

Part II. Programming Guide

Using the Virtual Clock

The virtual clock provided by the SPU timer library is a 64-bit software managed *Time Base* counter that represents the elapsed runtime of the calling thread since the start of the clock. The clock increments at the *Time Base* frequency so it can be used to perform relatively high precision time measurements. The clock is started by calling the ***spu_clock_start()*** service, and is read by calling the ***spu_clock_read()*** service.

Using Virtual Timers

The timers provided by the SPU timer library are virtual interval timers. They are virtual in the sense that they are built on the virtual clock, so their expiration interval is based on elapsed runtime, and not wall time. A timer allows an application to register a handler to be called on a specified interval. Up to SPU_TIMER_NTIMERS may be active at a time, each with a different expiration interval.

To use a timer, it must first be allocated using the ***spu_timer_alloc()*** service. Once allocated, the timer may be started and stopped as desired, using the ***spu_timer_start()*** and ***spu_timer_stop()*** services. The ability to start and stop a timer is useful since it allows for profiling of specific blocks of code. When a timer is no longer needed, it can be freed using the ***spu_timer_free()*** service. A timer must in the stopped state for it to be freed. A timer is in the stopped state if it has been allocated and not yet started, or has been previously started and explicitly stopped.

Once started, a timer will call the registered handler on each interval. When called, the timer ID of the expired timer is passed to the handler. This makes it possible to use the same handler for multiple timers, if desired, since the handler has the option to take timer-specific action based on the ID. After the handler is called, the timer is automatically restarted with the same interval and handler, unless the handler has stopped it. The handler may also free the timer after stopping it, in which case it cannot be restarted.

The expiration interval of the timer should be chosen based on the tradeoff between the statistical accuracy of the samples and the performance impact of doing the sampling. A good starting point might be an interval of one millisecond. The timer interval is specified in *Time Base* units, so a conversion from seconds to *Time Base* units needs to be done to determine a reasonable value. The *Time Base* frequency is system dependent but can be determined by reading the value as reported in the */proc/cpuinfo* file.

Interrupt Handlers

The clock and timer services require the use of first and second-level interrupt handlers (*FLIH* and *SLIH*) for servicing timer requests. The library provides both a *FLIH* and a *SLIH* for handling the *decrementer* interrupt. The use of the library-supplied *SLIH* is required for using the clock and timer services. Use of the library-supplied *FLIH* is optional, but recommended.

Applications that wish to use the library-supplied *FLIH* need to just call the provided ***spu_slih_register()*** service to register ***spu_clock_slih()*** as the *SLIH* for the MFC_DECREMENTER_EVENT. This service is part of the library *FLIH* and the symbol reference to it causes it to be linked into the application.

Applications that wish to supply their own *FLIH*, must register ***spu_clock_slih()*** using their own mechanism.

The *SLIH* must be registered before using any of the clock or timer services. Registration of new handlers after the clock and timers have been activated is not supported and will result in undefined behavior.

The contents of the *SPU_RdEventStat* register should be passed to ***spu_clock_slih()***, which will return this status with the *decrementer* event cleared.

A user-provided *FLIH* should not acknowledge the *decrementer* event, since the SPU clock code requires that the *decrementer* continue to run while the timer handlers are running. The clock code acknowledges the event before returning.

Programming Environment

Since the management of the SPU clock and timers is interrupt-driven, application code must be aware that it can be interrupted at any time when using them. This affects the use of events, as well as other global resources. As such, access to global resources must be managed appropriately. This includes:

Hardware Resources – Hardware resources such as the MFC DMA write channels may be used by applications, libraries, and interrupt handlers. To ensure consistency of the state of these channels, application code that runs with interrupts enabled must disable interrupts when performing a DMA, to ensure it does not get interrupted during its sequence of channel writes. This also applies any other channels that may be used by both the application code and the interrupt handler.

Global Application Data – Accesses to global application data that might be shared between the main application thread and the interrupt handler must similarly be protected by disabling interrupts (or the specific event which triggers the access) when manipulating the data.

User-Registered Handlers – Interrupt handlers must also be interrupt-safe, which means they can't acquire any application locks that may be acquired with interrupts enabled, and they can't call any library services that aren't known to be interrupt-safe. Applications can make their services interrupt-safe, as needed, by disabling interrupts when holding locks.

Events – The SPU timer library enables the interrupt facility on the SPU whenever the clock is running. When interrupts are enabled, all events that are of interest to the application will generate an interrupt when they are posted. For this reason, the SPU timer library should not be used with applications that may be using synchronous event notification without being interrupt-safe. Code that may be using synchronous events can be made interrupt-safe by disabling interrupts across the period of time where the event could be posted and when it is recognized.

Decrementer - When the SPU clock is being used, all use of the *decrementer* needs to be through the services provided by the library. Direct reading of the *decrementer* when timers are enabled will provide inconsistent results since it may be reset frequently by the clock's *SLIH*. Writing of the *decrementer* and registration of a user-defined *decrementer SLIH* while the SPU clock is running is not supported and results in undefined behavior.

Debugging

The SPU timer library package includes a non-debug version of the library (**libsputimer.a**) and a debug version of the library (**libsputimer_dbg.a)**. The debug library contains more error checking than the non-debug library and should be used during application development to help catch any potential programming errors. When development is finished, the application can be deployed with the non-debug library for slightly better performance.

Note: At the time of this release, inconsistent behavior may be seen with the clock and timers if the application is stopped for an extended period with an application-level debugger (such as gdb).

Example Using Clock and Timers

The following pseudo-code is an example of SPU clock and timer usage:

```
#include <spu_mfcio.h>
#include <spu_timer.h>

/* use library FLIH and SLIH */
spu_slih_register (MFC_DECREMENTER_EVENT, spu_clock_slih);

/* alloc timer for profiling */
id = spu_timer_alloc (14318, my_prof_handler);

/* start clock before timer */
spu_clock_start ();

/* profile the following block */
spu_timer_start (id);

while (more_work) {
      /* measure total time for work() */
      start = spu_clock_read ();
      work();
      time_working += (spu_clock_read () - start);
      other_work ();
}

/* done profiling */
spu_timer_stop (id);
spu_timer_free (id);

/* more work */
start = spu_clock_read ();

more_work ();

time_working += (spu_clock_read () - start);

/* done profiling and timing */
spu_clock_stop ();
```

Part III. API Reference

spu_clock_start

NAME

spu_clock_start – Start the SPU virtual clock

SYNOPSIS

#include <spu_timer.h>

void spu_clock_start (void)

DESCRIPTION

Starts the SPU virtual clock. After the clock has been started, it can be read using ***spu_clock_read()*** and timer services may be used. The clock *SLIH* (***spu_clock_slih()***) must be registered before starting the clock. The behavior is undefined if the clock is started without having registered the *SLIH*.

Since the SPU clock may be used by applications and libraries without knowledge of each other, the state of the clock must be coordinated among potentially several users. For this reason, the SPU clock maintains and internal start count, to ensure that one requester cannot stop the clock while it is in use by another. This count is incremented on start requests and decremented on stop requests. Attempts to stop the clock will fail if the start count is non-zero after decrementing the count. This failure simply means that someone else is using the clock and can in most cases be ignored.

The clock value is reset to zero whenever it is (re)started.

SEE ALSO

spu_clock_read; spu_clock_stop

spu_clock_stop

NAME

spu_clock_stop – Stop the SPU virtual clock

SYNOPSIS

#include <spu_timer.h>

int spu_clock_stop (void)

DESCRIPTION

Stops the SPU virtual clock. After the clock has been stopped, ***spu_clock_read()*** will return zero, and the ***spu_timer_start()*** and ***spu_timer_stop()*** services will fail.

This service decrements the start count of the clock and stops it if the count becomes zero. If the count was decremented, but the clock was not stopped, it returns an error code to indicate this. If the start count is one and there are active timers, the service fails.

RETURN VALUES

Returns 0 if the clock was successfully stopped. Returns one of the following error codes upon error:

SPU_CLOCK_ERR_NOT_RUNNING	The clock is not running.
SPU_CLOCK_ERR_STILL_RUNNING	The clock start count was decremented but the clock was not stopped.
SPU_CLOCK_ERR_TIMERS_ACTIVE	The clock was not stopped because there are active timers.

SEE ALSO

spu_clock_read; spu_clock_start

spu_clock_read

NAME

spu_clock_read – Read the SPU virtual clock

SYNOPSIS

#include <spu_timer.h>

uint64_t spu_clock_read (void)

DESCRIPTION

Read the SPU virtual clock. Returns the elapsed time since the start of the clock in *Time Base* units. The clock can be read any time after it is started. This service will return zero when the clock is not running.

RETURN VALUES

Returns 0 if the clock is either not running or is running and has not yet ticked, else the non-zero virtual clock value.

SEE ALSO

spu_clock_start; spu_clock_stop

spu_timer_alloc

NAME

spu_timer_alloc – Allocate an SPU timer

SYNOPSIS

#include <spu_timer.h>

int spu_timer_alloc (int tb_intvl, void (*handler)(int))

PARAMETERS

tb_intvl	The number of *timebase* units for the timer expiration interval. Can be any positive integer between 1 and INT_MAX.
handler	Pointer to the expiration handler to be called on each interval. The ID of the expired timer is passed to the handler.

DESCRIPTION

Allocates a new timer. The newly allocated timer remains inactive until started by a call to ***spu_timer_start()***. The timer remains allocated until freed by a call to ***spu_timer_free()***. The clock does not need to be running when allocating a timer.

RETURN VALUES

Upon success, returns the timer ID of the new timer. Valid timer IDs are in the range 0 – (SPU_TIMER_NTIMERS – 1). Upon failure, returns one of the following error codes:

SPU_TIMER_ERR_INVALID_PARM	*tb_intvl* was out of range or handler was NULL
SPU_TIMER_ERR_NONE_FREE	There are no free timers to allocate

spu_timer_free

NAME

spu_timer_free – Free an SPU timer

SYNOPSIS

#include <spu_timer.h>

int spu_timer_free (int id)

PARAMETERS

id	The ID of the timer to free

DESCRIPTION

Frees an allocated timer. This service fails if the specified timer is currently active. This service can be called successfully before a timer is started, after it is stopped, from application code or from the timer's handler. After a timer is freed, no further operations on it are permitted.

The clock does not need to be running when freeing a timer.

RETURN VALUES

Returns 0 upon success. Returns one of the following error codes upon failure:

SPU_TIMER_ERR_INVALID_ID	*id* does not refer to an allocated timer
SPU_TIMER_ERR_NOT_STOPPED	*id* does not refer to a stopped timer

SEE ALSO

spu_timer_alloc

spu_timer_start

NAME

spu_timer_start – Start an SPU timer

SYNOPSIS

#include <spu_timer.h>

int spu_timer_start (int id)

PARAMETERS

id	The ID of the timer to start

DESCRIPTION

Starts the specified timer. When started, a timer remains active until it is stopped. While active, the timer's expiration handler is called on each interval.

RETURN VALUES

Returns 0 upon success. Returns one of the following error codes upon failure:

SPU_TIMER_ERR_INVALID_ID	*id* does not refer to an allocated timer
SPU_TIMER_ERR_NOT_STOPPED	*id* does not refer to a stopped timer
SPU_TIMER_ERR_NOCLOCK	The SPU clock is not running.

SEE ALSO

spu_timer_alloc; spu_timer_stop;

spu_timer_stop

NAME

spu_timer_stop – Stop an SPU timer

SYNOPSIS

#include <spu_timer.h>

int spu_timer_stop (int id)

PARAMETERS

id	The ID of the timer to stop

DESCRIPTION

Stops the specified timer. The timer is put into the stopped state where it remains until it is either restarted or freed.

RETURN VALUES

Returns 0 upon success. Returns one of the following error codes upon failure:

SPU_TIMER_ERR_INVALID_ID	*id* does not refer to an allocated timer
SPU_TIMER_ERR_NOT_ACTIVE	*id* does not refer to an active timer
SPU_TIMER_ERR_NOCLOCK	The SPU clock is not running

spu_slih_register

NAME

spu_slih_register – Register a second level interrupt handler

SYNOPSIS

#include <spu_mfcio.h>

#include <spu_timer.h>

void spu_slih_register (unsigned event_mask, unsigned (*slih)(unsigned)

PARAMETERS

event_mask	The set of events for which to call the registered handler
slih	The pointer to the function to use as the second-level interrupt handler

DESCRIPTION

Register a second-level interrupt handler for the given events. The handler will be called for the events specified in the mask. The values for the event mask can be found in *spu_mfcio.h*.

For library initialization, **spu_clock_slih** should be registered as the handler for the MFC_DECREMENTER_EVENT, using this service.

SEE ALSO

spu_clock_slih

spu_clock_slih

NAME

spu_clock_slih – Second level interrupt handler for SPU clock

SYNOPSIS

#include <spu_timer.h>

unsigned spu_clock_slih (unsigned event_mask)

PARAMETERS

event_mask	The list of pending events at the time of the call.

DESCRIPTION

The second-level interrupt handler for the clock and timer services. This needs to be registered as the handler for the MFC_DECREMENTER_EVENT before starting the SPU virtual clock. It can be registered using the provided ***spu_slih_register()*** service or by a user-provided service. If registered using a service other than that provided by the library, it is the application's responsibility to also provide the first-level interrupt handler to call it as appropriate.

RETURN VALUES

Returns the *event_mask* value that was passed in, with the MFC_DECREMENTER_EVENT cleared and acknowledged.

Part IV. Appendices

Appendix A. Installation

The following packages are available with SDK 3.0 for using the SPU timer library:

Package Name	**Description**
spu-timer-devel-3.0.0-#.ppc.rpm	Development – Header file and static libraries
spu-timer-cross-devel-3.0.0-#.noarch.rpm	Cross Platform Development – Header file and static libraries
spu-timer-devel-debuginfo-3.0.0-#.ppc.rpm	Debug symbols

The installed files are:

<*path*>/prototype/usr/spu/include/spu_timer.h
<*path*>/prototype/usr/spu/lib/libsputimer.a
<*path*>/prototype/usr/spu/lib/libsputimer_dbg.a

Where <*path*> is either:

/opt/cell/sdk (for the "devel" package)
or
/opt/cell/sysroot/opt/cell/sdk (for the "cross-devel" package).

Appendix B. Constants

The header file for the SPU timer library defines the following constants:

Timer Constants

SPU_TIMER_NTIMERS	The number of timers that may be allocated. For SDK 3.0 this is 4.

Clock Error Codes

SPU_CLOCK_ERR_NOT_RUNNING	The clock is not running
SPU_CLOCK_ERR_STILL_RUNNING	The clock start count was decremented but the clock was not stopped.
SPU_CLOCK_ERR_TIMERS_ACTIVE	The clock was not stopped because there are active timers.

Timer Error Codes

SPU_TIMER_ERR_INVALID_PARM	Invalid parameter
SPU_TIMER_ERR_NONE_FREE	There are no free timers to allocate
SPU_TIMER_ERR_INVALID_ID	Invalid timer ID
SPU_TIMER_ERR_NOT_ACTIVE	Specified timer is not active
SPU_TIMER_ERR_NOCLOCK	Clock is not running
SPU_TIMER_ERR_NOT_STOPPED	Specified timer is not stopped

Appendix C. Accessibility

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

The following list includes the major accessibility features:

- Keyboard-only operation
- Interfaces that are commonly used by screen readers
- Keys that are tactilely discernible and do not activate just by touching them
- Industry-standard devices for ports and connectors
- The attachment of alternative input and output devices

IBM® and accessibility

See the IBM Accessibility Center at http://www.ibm.com/able/ for more information about the commitment that IBM has to accessibility.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION ″AS IS″ WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you. This information could include technical inaccuracies or typographical errors.

Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101

11400 Burnet Road
Austin, TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee. The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Trademarks
