Software Development Kit for Multicore Acceleration
Version 3.1

SPU Runtime Library Extensions Programmer's Guide and API Reference

SC34-2593-00

Software Development Kit for Multicore Acceleration
Version 3.1

SPU Runtime Library Extensions Programmer's Guide and API Reference

SC34-2593-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 51.

This edition applies to version 3, release 1, modification 0 of the IBM Software Development Kit for Multicore Acceleration (Product number 5724-S84) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this publication

This publication describes the SPU Runtime Library Extensions in detail and how to program applications using it on the IBM Software Development Kit for Multicore Acceleration (SDK). It contains detailed reference information about the APIs for the library as well as sample applications showing usage of these APIs.

Who should use this book

The target audience for this document is application programmers using the SDK. You are expected to have a basic understanding of programming on the Cell Broadband Engine™ (Cell/B.E.) platform and common terminology used with the Cell/B.E. platform.

Typographical conventions

The following table explains the typographical conventions used in this document.

Table 1. Typographical conventions

Typeface	Indicates	Example
Bold	Lowercase commands, library functions.	**void sscal_spu (float *sx, float sa, int n)**
Italics	Parameters or variables whose actual names or values are to be supplied by the user. Italics are also used to introduce new terms.	The following example shows how a test program, *test_name* can be run
`Monospace`	Examples of program code or command strings.	`int main()`

Related information

For a list of SDK documentation, see Appendix D, “Related documentation,” on page 47.

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this publication, send your comments using IBM Resource Link™ at http://www.ibm.com/servers/resourcelink. Click **Feedback** on the navigation pane. Be sure to include the name of the book, the form number of the book, and the specific location of the text you are commenting on (for example, a page number or table number).

Part 1. SPU timer library

The topics in this section describe how to use the SPU timer library.

The following topics are described:

Chapter 1. Overview of the SPU timer library

The SPU timer library provides a software-managed 64-bit monotonically increasing clock and interval timer services for SPU programs. The 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 at a specified interval.

By managing 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. This is code to which the processor branches 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.

Chapter 2. Installing the SPU timer library

The SPU timer library is available as part of the SDK new library.

Refer to the *SDK Installation Guide* for more information.

Chapter 3. Programming with the SPU timer library

This section describes how to program with SPU timer library.

It covers the following topics:

Using the software clock

The software 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. To start the clock, call the `spu_clock_start()` service, and to read the clock, call the `spu_clock_read()` service.

Using SPU timers

The timers provided by the SPU timer library are interval timers. The timers are interval timers, meaning they will repeatedly expire after the specified interval when the decrementer is enabled for interrupts. A timer allows an application to register a handler to be called on a specified interval. A timer allows an application to register a handler to be called on a specified interval. Up to `SPU_TIMER_NTIMERS` can be active at a time, each with a different expiration interval.

To use a timer, you must first use the `spu_timer_alloc()` service to allocate it. After the timer has been allocated, use the `spu_timer_start()` and `spu_timer_stop()` services to start and stop it as required. The ability to start and stop a timer is useful because it allows specific blocks of code to be profiled. When a timer is no longer needed, use the `spu_timer_free()` service to free it. 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.

After the timer has been started, it calls the registered handler at 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, because 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 can also free the timer after it has stopped it, in which case it cannot be restarted.

You should chose the expiration interval of the timer based on the trade off between the statistical accuracy of the samples and the performance impact of doing the sampling. A good starting point is an interval of one millisecond or 100 tics. 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 or using the spu_timebase() API. See “spu_timebase” on page 35

Interrupt handlers

The clock and timer services require the use of first-level interrupt handlers (FLIH) and second-level interrupt handlers (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 results in undefined behavior.

Notes if using a user-provided FLIH: the contents of the `SPU_RdEventStat` register should be passed to `spu_clock_slih()`, which returns this status with the `decrementer` event cleared. A user-provided FLIH should include the *decrementer* event when acknowledging the events, but exclude the decrementer event when the writing the event mask, The SPU clock code requires that the *decrementer* continue to run while the timer handlers are running. The clock code restarts the decrementer and enables for interrupts before returning.

Programming environment

Because 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 cannot 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. Note that the SPU is disabled for interrupts when the handlers are invoked.

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 provides 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.

Examples using clocks and timers

The first example shows how to use the SPU timer clock API to configure, start, and read the clock. The second example shows how to use the SPU Timer APIs with the clock APIs to configure an interval timer and profile a trivial loop.

Example 1: Using the SPU timer clock API

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

#define MAX_COUNT 0x100000ULL

static uint64_t ctr = 0;

/* profiled work function  */
void work () {
    ++ctr;
}

/* non profiled work function  */
void other_work () {
    ++ctr;
}

/* non profiled work control function */
int more_work () {
  int x = (ctr > MAX_COUNT) ? 0 : 1;
  return x;
}

/* timer interrupt handler counter */
static uint64_t actual_count = 0;

/* timer interrupt handler to collect data */
void my_prof_handler (int id)
{
  ++ actual_count;
}

int main ()
{
```

```
  uint64_t start;
  uint64_t time_working =0;
  int id;

  /* use library FLIS 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);

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

  printf("MAX_COUNT[%lld] ctr[%lld] actual_count[%lld] time_working[%lld]\n",
         MAX_COUNT,ctr,actual_count,time_working);
  return 0;
}
```

Example 2: Using the SPU timer APIs with the clock APIs

```
#include <spu_timer.h>
#include <spu_mfcio.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>


#define SLEEPTIME 10 /* sleep interval in seconds */

#define ITERATIONS 30  /* limit for sleep function calls */

/* QS20 timebase - from command: cat /proc/cpuinfo */
#define TIMEBASE 14318000int main()
{
   int rc;
   int x;
   uint64_t start_tics = 0;
   uint64_t end_tics = 0;
   uint64_t total_tics = 0;
   uint64_t seconds = 0;
   uint64_t remainder = 0;
   uint64_t timebase = spu_timebase()

   printf("Total sleeptime [%d] seconds in [%d] iterations of [%d] seconds\n",
           SLEEPTIME * ITERATIONS,ITERATIONS,SLEEPTIME);

   /* register slih to handle decrementer interval interrupts */
   spu_slih_register(MFC_DECREMENTER_EVENT, spu_clock_slih);
```

```
    spu_clock_start();

    start_tics = spu_clock_read();

    /* iterate sleep time intervals - iteration allows decrementer interrupts */
    for (x=0;x<ITERATIONS ;++x ) {
        printf("interation %d, sleeping for %d seconds\n", x, SLEEPTIME);
        sleep(SLEEPTIME);
    }

    end_tics = spu_clock_read();

    rc = spu_clock_stop();

    total_tics = end_tics - start_tics;

    seconds = total_tics/timebase;

    remainder = total_tics % timebase;

    printf("sleep time in seconds[%d] total tics[%lld] seconds[%lld r%lld]\n",
           SLEEPTIME,total_tics,seconds,remainder);

    return 0;
}
```

Part 2. EA library

The topics in this section describe how to use the EA library.

The following topics are described:

Chapter 4. The _ea address space qualifier language extension

This topic describes the _ea address space qualifier language extension implementations.

The following topics are described:

- "The _ea address space qualifier language extension implementation in GCC"
- For information about the _ea address space qualifier language extension implementation in XLC, refer to:

http://publib.boulder.ibm.com/infocenter/cellcomp/v9v111/index.jsp?topic=/com.ibm.cellptf.doc/effective_address_support.html

The _ea address space qualifier language extension implementation in GCC

When you develop SPE programs using the SDK, you may wish to reference variables in the PPE address space from code running on an SPE. This is achieved through an extension to the C language syntax.

It might be desirable to share data in this way between an SPE and the PPE. This extension makes it easier to pass pointers so that you can use the PPE to perform certain functions on behalf of the SPE. You can readily share data between all SPEs through variables in the PPE address space.

The compiler recognizes an address space identifier `__ea` that can be used as an extra type qualifier like `const` or `volatile` in type and variable declarations. You can qualify variable declarations in this way, but not variable definitions.

The following are examples.

```
/* Variable declared on the PPE side. */
extern __ea int ppe_variable;

/* Can also be used in typedefs. */
typedef __ea int ppe_int;

/* SPE pointer variable pointing to memory in the PPE address space */
__ea int *ppe_pointer;
```

Pointers in the SPE address space can be cast to pointers in the PPE address space. Doing this transforms an SPE address into an equivalent address in the mapped SPE local store (in the PPE address space). The following is an example.

```
int x;
__ea int *ppe_pointer_to_x = &x;
```

These pointer variables can be passed to the PPE process by way of a mailbox and used by PPE code. With this method, you can perform operations in the PPE execution context such as copying memory from one region of the SPE local store to another.

In the same way, these pointers can be converted to and from the two address spaces, as follows:

```
int *spe_x;
spe_x = (int *) ppe_pointer_to_x;
```

References to __ea variables cause decreased performance. The implementation performs software caching of these variables, but there are much higher overheads when the variable is accessed for the first time. Modifications to __ea variables is also cached. The writeback of such modifications to PPE address space may be delayed until the cache line is flushed, or the SPU context terminates.

GCC for the SPU provides the following command line options to control the runtime behavior of programs that use the __ea extension. Many of these options specify parameters for the software-managed cache. In combination, these options cause GCC to link your program to a single software-managed cache library that satisfies those options. Table 2 describes these options.

Table 2. Options

Option	Description
-mea32	Generate code to access variables in 32-bit PPU objects. The compiler defines a preprocessor macro __EA32__ to allow applications to detect the use of this option. This is the default.
-mea64	Generate code to access variables in 64-bit PPU objects. The compiler defines a preprocessor macro __EA64__ to allow applications to detect the use of this option.
-mcache-size=8	Specify an 8 KB cache size.
-mcache-size=16	Specify an 16 KB cache size.
-mcache-size=32	Specify an 32 KB cache size.
-mcache-size=64	Specify an 64 KB cache size.
-mcache-size=128	Specify an 128 KB cache size.
-matomic-updates	Use DMA atomic updates when flushing a cache line back to PPU memory. This is the default.
-mno-atomic-updates	This negates the -matomic-updates option.

Accessing an __ea variable from an SPU program creates a copy of this value in the local storage of the SPU. Subsequent modifications to the value in main storage are not automatically reflected in the copy of the value in local store. It is your responsibility to ensure data coherence for __ea variables that are accessed by both SPE and PPE programs.

A complete example using __ea qualifiers to implement a quick sort algorithm on the SPU accessing PPE memory can be found in the examples/ppe_address_space directory provided by the SDK cell-examples tar ball.

Chapter 5. Installing the EA library APIs

The EA library functions are available as part of the SPU runtime library (newlib).

Refer to the *SDK Installation Guide* for more information.

Chapter 6. Programming with the EA library APIs

This topic describes how to program with the EA APIs.

The following topics are described:

- "PPU Linux system calls"
- "Support for PPE address space on the SPE" on page 20

Refer to Appendix C, "Example: programming with the EA library," on page 45 for sample code for EA library program application.

PPU Linux system calls

This topic describes how to use sys/linux_syscalls.h.

The Linux® Kernel does not implement all system calls for the SPU. Newlibs header file sys/linux_syscalls.h defines symbolic names for each supported system call number. The newlib implements only the most important system calls (see Linux system call functions), but provides a generic interface to allow the developer to implement them. The sys/linux_syscalls.h defines a struct called spu_syscall_block and a function called __linux_syscall. The function takes a pointer to the struct that defines the number of system call to be issued and an array of up to six arguments:

A struct called `spu_syscall_block`:

```
struct spu_syscall_block
 {
   unsigned long long nr_ret;   /* System call nr and return value.  */
   unsigned long long parm[6];  /* System call arguments.  */
 };
```

A function declaration for issuing a system call:

```
int __linux_syscall (struct spu_syscall_block *s)
```

The __linux_syscall function uses the stop and signal mechanism to notify the Kernel to execute that particular system call.

Example

The following is an example for the read system call (that reads data from a file) wrapped in a function called `read_ea`:

```
<snip>
 #include <ea.h>
 #include <sys/linux_syscalls.h>

 ssize_ea_t
 read_ea (int fd, __ea void *buf, size_ea_t count)
 {
  struct spu_syscall_block s = {
    __NR_read,
    {
      fd,
      (size_ea_t) buf,
      count,
      0,
```

```
      0,
      0}
    };
    __cache_flush ();
    return __linux_syscall (&s);
  }
 </snip>
```

Note: This example is already implemented (see newlib/libc/machine/spu/readv_ea.c) and declared in ea.h. Wherever possible, you should use the existing functions instead of implementing your own. See "Linux system call functions" on page 21 for a list of implemented Linux system call functions.

Support for PPE address space on the SPE

The __ea type qualifier is a PPE address namespace identifier. The syntax for using it is the same as for using type qualifiers const and volatile.

For more information, refer to Chapter 4, "The _ea address space qualifier language extension," on page 15

Functions defined in ea.h behave exactly as their non-ea companions except that they are work on ea variables (data in the PPE memory).

The ea.h implements the following:

Memory mapping functions

```
__ea void *mmap_ea (__ea void *start, size_ea_t length, int prot, int flags, int fd, off_t offset);
int munmap_ea (__ea void *start, size_ea_t length);
__ea void *mremap_ea (__ea void *old_address, size_ea_t old_size, size_ea_t new_size, unsigned long flags);
int msync_ea (__ea void *start, size_ea_t length, int flags);
```

EA memory management functions

```
__ea void *calloc_ea (size_ea_t nmemb, size_ea_t length);
void free_ea (__ea void *ptr);
__ea void *malloc_ea (size_ea_t size);
__ea void *realloc_ea (__ea void *ptr, size_ea_t size);
int posix_memalign_ea (__ea void **memptr, size_ea_t alignment, size_ea_t size);
```

String copying functions

```
__ea void *memcpy_ea (__ea void *dest, __ea const void *src, size_ea_t n);
__ea void *memmove_ea (__ea void *dest, __ea const void *src, size_ea_t n);
__ea char *strcpy_ea (__ea char *dest, __ea const char *src);
__ea char *strncpy_ea (__ea char *dest, __ea const char *src, size_ea_t n);
```

Concatenation functions

```
__ea char *strcat_ea (__ea char *dest, __ea const char *src);
__ea char *strncat_ea (__ea char *dest, __ea const char *src, size_ea_t n);
```

Comparison functions

```
int memcmp_ea (__ea void *s1, __ea const void *s2, size_ea_t n);
int strcmp_ea (__ea char *s1, __ea const char *s2);
int strncmp_ea (__ea void *s1, __ea const void *s2, size_ea_t n3);
```

Search functions

```
__ea void *memchr_ea (__ea const void *s, int c, size_ea_t n);
__ea char *strchr_ea (__ea const char *s, int c);
size_ea_t strcspn_ea (__ea const char *s, const char *reject);
__ea char *strpbrk_ea (__ea const char *s, const char *accept);
```

```
__ea char *strrchr_ea (__ea const char *s, int c);
size_ea_t strspn_ea (__ea const char *s, const char *accept);
__ea char * strstr_ea (__ea const char *s1, __ea const char *s2);
```

Miscellaneous functions

```
__ea void *memset_ea (__ea void *dest, int c, size_ea_t n);
size_ea_t strlen_ea (__ea const char *s);
```

Linux system call functions

```
ssize_ea_t read_ea(int fd, __ea void *buf, size_ea_t count);
ssize_ea_t pread_ea(int fd, __ea void *buf, size_ea_t count, off_t offset);
ssize_ea_t readv_ea(int fd, struct iovec_ea *vector, int count);
ssize_ea_t write_ea(int fd, __ea const void *buf, size_ea_t count);
ssize_ea_t pwrite_ea(int fd, __ea const void *buf, size_ea_t count, off_t offset);
ssize_ea_t writev_ea(int fd, struct iovec_ea *vector, int count);
```

Type definitions

ea.h defines the following types:

```
size_ea_t
ssize_ea_t
key_ea_t
iovec_ea
```

Part 3. APIs

This topic describes the APIs.

Chapter 7. SPU timer library APIs

This section provides information about the SPU timer library APIs.

The following APIs are described:

spu_clock_start

NAME

spu_clock_start - Starts the SPU clock.

SYNOPSIS

#include <spu_timer.h>
void spu_clock_start (void)

DESCRIPTION

Starts the SPU clock. After the clock has been started, it is 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`.

Because the SPU clock might 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 when the count is non-zero will result in a non zero return code. This return code means that someone else is using the clock and is not considered a failure. The clock value is reset to zero whenever it is (re)started.

SEE ALSO

spu_clock_read(3); spu_clock_stop(3)

spu_clock_stop

NAME

spu_clock_stop - Stops the SPU clock.

SYNOPSIS

#include <spu_timer.h>
int spu_clock_stop (void)

DESCRIPTION

Stops the SPU clock. After the clock stops, `spu_clock_read()` returns zero, and the `spu_timer_start()` and `spu_timer_stop()` services 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(3); spu_clock_start(3)

spu_clock_read

NAME

spu_clock_read - Reads the SPU clock.

SYNOPSIS

```
#include <spu_timer.h>
uint64_t spu_clock_read (void)
```

DESCRIPTION

Reads the SPU 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 returns 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, or else the non-zero clock value.

SEE ALSO

spu_clock_start(3); spu_clock_stop(3)

spu_timer_alloc

NAME

spu_timer_alloc - Allocates 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. The recommended minimum is 1 microsecond or 100 tics.
`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 to allocate 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 - Frees 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 to free a timer.

RETURN VALUES

Returns 0 upon success. Valid timer IDs are in the range 0 - (SPU TIMER NTIMERS - 1).

Returns one of the following error codes upon failure:

`SPU_TIMER_ERR_INVALID_ID`	*id* does not refer to a valid timer id
`SPU_TIMER_ERR_ACTIVE`	*id* refers to an active timer
`SPU_TIMER_ERR_FREE`	*id* refers to an already free timer

SEE ALSO

spu_timer_alloc(3)

spu_timer_start

NAME

spu_timer_start - Starts 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. Valid timer IDs are in the range 0 - (SPU TIMER NTIMERS - 1).

Returns one of the following error codes upon failure:

`SPU_TIMER_ERR_INVALID_ID`	*id* does not refer to a valid timer ID
`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(3); spu_timer_stop(3)

spu_timer_stop

NAME

spu_timer_stop - Stops an SPU timer.

SYNOPSIS

#include <spu_timer.h>
int_spu_timer_start (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. Valid timer IDs are in the range 0 - (SPU TIMER NTIMERS - 1).

Returns one of the following error codes upon failure:

`SPU_TIMER_ERR_INVALID_ID`	*id* does not refer to a valid timer id
`SPU_TIMER_ERR_NOT_ACTIVE`	*id* does not refer to an inactive timer
`SPU_TIMER_ERR_NOCLOCK`	The SPU clock is not running.

spu_slih_register

NAME

spu_slih_register - Registers a second level interrupt handler.

SYNOPSIS

#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.
`handler`	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 is called for the events specified in the mask. The values for the event mask can be found in `spu_mfcio.h`. If the slih value is NULL then the original default slih handler will be restored.

For library initialization, `spu_clock_slih` should be registered as the handler for the MFC_DECREMENTER_EVENT, using this service. To restore the original slih handler pass in a value of NULL for the slih. The original default slih only resets the interrupt, the spu clock slih must be registered for the handlers to be invoked.

SEE ALSO

spu_clock_slih(3)

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 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 VALUE

Returns the `event_mask` value that was passed in, with the MFC_DECREMENTER_EVENT cleared and the decrementer restarted.

spu_timebase

NAME

spu_timebase - Returns the SPU time base in hertz.

SYNOPSIS

```
#include <spu_timer.h>
unsigned spu_timebase (void);
```

DESCRIPTION

This function queries /proc/cpuinfo and returns the SPU time base in hertz. The time base varies on different hardware platforms. You can use this function to set up a timer that triggers your code with the same frequency on all hardware platforms.

Part 4. Appendixes

Appendix A. SPU constants

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

Table 3. Timer constants

SPU_TIMER_NTIMERS	The number of timers that may be allocated.
SPU_TIMER_MIN_INTERVAL	Recommended minimum SPU Timer interval time.

Table 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.

Table 5. 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_ACTIVE	Specified timer is active
SPU_TIMER_ERR_NOT_ACTIVE	Specified timer is not active
SPU_TIMER_ERR_NOCLOCK	Clock is not running
SPU_TIMER_ERR_FREE	Specified timer is free
SPU_TIMER_ERR_NOT_STOPPED	Specified timer is not stopped

Appendix B. Example: programming with the SPU timer library

Profile sample code for SPU timer library.

This sample demonstrates using the SPU timer APIs for profiling. This sample consists of 2 files, prof_test.c and sprof.h. The file prof_test.c is shown first and includes the file sprof.h. The file sprof.h contains the profiling logic. This makes it easy to add profiling to an existing SPU program.

```
#include "sprof.h"

int
main()
{
    int i = 0,z=0;
    sprof_enable(14318);

    while (++i < 1000000) {
        unsigned was_enabled;
        was_enabled = spu_readch(SPU_RdMachStat);
        int y = i+1;
        z = y/i + y%i;
    }

    sprof_disable();
    sprof_dump();
    return z;
}


/* sprof.h
 *
 * Copyright (C) 2008 IBM Corp.
 *
 * Simple profile & hot-spot tracking
 * for SPE.
 */

#ifndef __SPROF_H__
#define __SPROF_H__

#include <stdio>
#include <spu_intrinsics_h>
#include <asset.h>
#include <spu.timer.h>

volatile unsigned int event_count = 0;

#define SPU_RDCH_INSTR          0x01a00000
#define SPU_WRCH_INSTR          0x21a00000
#define SPU_INSTR_MASK          0xfff00000

volatile unsigned int sprof_rdch_cntr __attribute ((aligned (16))) = 0;
volatile unsigned int sprof_wrch_cntr __attribute ((aligned (16))) = 0;
volatile unsigned int sprof_misc_cntr __attribute ((aligned (16))) = 0;

#if 0
#define PROF_TABLE_SHIFT        13
#define PROF_TABLE_SZ           (1 << PROF_TABLE_SHIFT)
#define PROF_TABLE_MASK         (PROF_TABLE_SZ - 1)
#else
```

```
#define PROF_TABLE_SHIFT        10
#define PROF_TABLE_SZ           (1 << PROF_TABLE_SHIFT)
#define PROF_TABLE_MASK         (PROF_TABLE_SZ - 1)
#endif

volatile unsigned int sprof_tagid_table[32];
volatile unsigned int sprof_ch_table[32];
volatile unsigned int sprof_ticks = 0;


static inline unsigned int
sprof_hash (unsigned int npc)
{
    // instructions are
    // always word addressable.
    //return (npc >> 9) & PROF_TABLE_MASK;
    return ((npc >> 8) & PROF_TABLE_MASK);
}


static void
sprof_tick (int id __attribute__ ((unused)))
{
    unsigned int npc = spu_readch (SPU_RdSRR0);
    unsigned int tagmask = spu_readch (MFC_RdTagMask);
    unsigned int *npc_ptr = (unsigned int *) npc;
    unsigned int instr = *npc_ptr & SPU_INSTR_MASK;
    unsigned int ch = (*npc_ptr >> 7) & 0x1f;
    unsigned int r = sprof_rdch_cntr;
    unsigned int w = sprof_wrch_cntr;
    unsigned int m = sprof_misc_cntr;
    unsigned int r1 = r + 1;
    unsigned int w1 = w + 1;
    unsigned int m1 = m + 1;
    unsigned int rdch = (instr == SPU_RDCH_INSTR);
    unsigned int wrch = (instr == SPU_WRCH_INSTR);
    unsigned int misc = (!(rdch || wrch));
    int i;

    sprof_ticks += 1;
    sprof_rdch_cntr = (rdch) ? r1 : r;
    sprof_wrch_cntr = (wrch) ? w1 : w;
    sprof_misc_cntr = (misc) ? m1 : m;


    if (rdch & (ch == 24))
    {

        for (i = 0; i < 32; i++)
        {
            if (tagmask & (1 << i))
                sprof_tagid_table[i] += 1;
        }
    }


    if (rdch | wrch)
    {
        /* ISA reserves 7b but only 5b are generally used. */
        sprof_ch_table[ch] += 1;
    }
}

int sprof_hz;
```

```
static void
sprof_dump (void)
{
    float percent_rdch =
        ((float) sprof_rdch_cntr / (float) sprof_ticks) * 100.0f;
    float percent_wrch =
        ((float) sprof_wrch_cntr / (float) sprof_ticks) * 100.0f;
    float percent_misc =
        ((float) sprof_misc_cntr / (float) sprof_ticks) * 100.0f;
    int i;

    printf ("\n");
    printf ("SPU Hot-Spot Dump.\n");
    printf ("  Profile frequency (HZ):    %10d\n", sprof_hz);
    printf ("  Total ticks:               %10d\n", sprof_ticks);
    printf ("  Ticks for rdch:            %10d\t (%2.2f %%)\n",
            sprof_rdch_cntr, percent_rdch);
    printf ("  Ticks for wrch:            %10d\t (%2.2f %%)\n",
            sprof_wrch_cntr, percent_wrch);
    printf ("  Ticks for compute:         %10d\t (%2.2f %%)\n",
            sprof_misc_cntr, percent_misc);
    printf ("\n");
    printf("\n");

    printf ("          Ticks by channel id:\n");
    printf ("       ticks\t         %%\t         CH\n");
    printf ("  ==========\t ==========\t ==========\n");
    for (i = 0; i < 32; i++)
    {
        if (sprof_ch_table[i])
        {
            float ratio = (float) sprof_ch_table[i] / (float) sprof_ticks;

            printf ("  %10d\t %10.2f\t %10d\n",
                    sprof_ch_table[i], ratio * 100.0f, i);
        }
    }
    printf("\n");
    printf("\n");

    printf ("          Ticks by tag group id:\n");
    printf ("       ticks\t         %%\t      tagid\n");
    printf ("  ==========\t ==========\t ==========\n");
    for (i = 0; i < 32; i++)
    {
        if (sprof_tagid_table[i])
        {
            float ratio = (float) sprof_tagid_table[i] / (float) sprof_ticks;

            printf ("  %10d\t %10.2f\t %10d\n",
                    sprof_tagid_table[i], ratio * 100.0f, i);
        }
    }
    printf("\n");
    printf("\n");

}

static unsigned sprof_timerid;

static inline void
sprof_enable (int hz)
{
    int i;


    for (i = 0; i < 32; i++)
```

```
    {
        sprof_tagid_table[i] = 0;
    }

    /* just for summary reporting */
    sprof_hz = hz;

    spu_slih_register(MFC_DECREMENTER_EVENT, spu_clock_slih);
    /* start the virtual clock */
    spu_clock_start ();

    /* alloc/start our profiling timer */
    sprof_timerid = spu_timer_alloc (hz, sprof_tick);
    assert ((int)sprof_timerid != -1);
    spu_timer_start (sprof_timerid);
}

static inline void
sprof_disable (void)
{
    spu_timer_stop (sprof_timerid);
    spu_timer_free (sprof_timerid);
    spu_clock_stop ();
}

#endif
```

Appendix C. Example: programming with the EA library

The following code shows how to program with the EA library APIs

```
#include  <ea.h>
#include  <string.h>
#include  <sys/types.h>
#include  <sys/stat.h>
#include  <fcntl.h>

#define  FILE_NAME "/tmp/ea_example"

int
main ()
{
  int  fd = 0;
  int  i;
  char  s[] = "hello world\n";
  __ea char *buf;

  /* Allocate PPE memory.  */
  buf = (char *) malloc_ea (strlen (s));

  /* Fill buf with our local char array using the __ea cache.  */
  for  (i = 0; i < strlen (s); ++i)
    buf[i] = s[i];

  /* Write the data of buf into a file.  */
  fd = open (FILE_NAME, O_WRONLY | O_CREAT | O_TRUNC, 0644);
  write_ea (fd, (__ea void *) buf, strlen (s));
  close (fd);

  free_ea (buf);
  return  0;
}
```

Appendix D. Related documentation

This topic helps you find related information.

Document location

Links to documentation for the SDK are provided on the IBM® developerWorks® Web site located at:

`http://www.ibm.com/developerworks/power/cell/`

Click the **Docs** tab.

The following documents are available, organized by category:

Architecture

- *Cell Broadband Engine Architecture*
- *Cell Broadband Engine Registers*
- *SPU Instruction Set Architecture*

Standards

- *C/C++ Language Extensions for Cell Broadband Engine Architecture*
- *Cell Broadband Engine Linux Reference Implementation Application Binary Interface Specification*
- *SIMD Math Library Specification for Cell Broadband Engine Architecture*
- *SPU Application Binary Interface Specification*
- *SPU Assembly Language Specification*

Programming

- *Cell Broadband Engine Programmer's Guide*
- *Cell Broadband Engine Programming Handbook*
- *Cell Broadband Engine Programming Tutorial*

Library

- *Accelerated Library Framework for Cell Broadband Engine Programmer's Guide and API Reference*
- *Basic Linear Algebra Subprograms Programmer's Guide and API Reference*
- *Data Communication and Synchronization for Cell Broadband Engine Programmer's Guide and API Reference*
- *Example Library API Reference*
- *Fast Fourier Transform Library Programmer's Guide and API Reference*
- *LAPACK (Linear Algebra Package) Programmer's Guide and API Reference*
- *Mathematical Acceleration Subsystem (MASS)*
- *Monte Carlo Library Programmer's Guide and API Reference*
- *SDK 3.0 SIMD Math Library API Reference*
- *SPE Runtime Management Library*
- *SPE Runtime Management Library Version 1 to Version 2 Migration Guide*
- *SPU Runtime Extensions Library Programmer's Guide and API Reference*

- *Three dimensional FFT Prototype Library Programmer's Guide and API Reference*

Installation

- *SDK for Multicore Acceleration Version 3.1 Installation Guide*

Tools

- *Getting Started - XL C/C++ for Multicore Acceleration for Linux*
- *Compiler Reference - XL C/C++ for Multicore Acceleration for Linux*
- *Language Reference - XL C/C++ for Multicore Acceleration for Linux*
- *Programming Guide - XL C/C++ for Multicore Acceleration for Linux*
- *Installation Guide - XL C/C++ for Multicore Acceleration for Linux*
- *Getting Started - XL Fortran for Multicore Acceleration for Linux*
- *Compiler Reference - XL Fortran for Multicore Acceleration for Linux*
- *Language Reference - XL Fortran for Multicore Acceleration for Linux*
- *Optimization and Programming Guide - XL Fortran for Multicore Acceleration for Linux*
- *Installation Guide - XL Fortran for Multicore Acceleration for Linux*
- *Performance Analysis with the IBM Full-System Simulator*
- *IBM Full-System Simulator User's Guide*
- *IBM Visual Performance Analyzer User's Guide*

IBM PowerPC Base

- *IBM PowerPC Architecture™ Book*
  - *Book I: PowerPC User Instruction Set Architecture*
  - *Book II: PowerPC Virtual Environment Architecture*
  - *Book III: PowerPC Operating Environment Architecture*
- *IBM PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual*

Appendix E. Accessibility features

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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

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 may 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.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester; MIN 55901
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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A complete and current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml.

Adobe, Acrobat, Portable Document Format (PDF), PostScript, and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc., in the United States, other countries, or both and is used under license therefrom

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of the manufacturer.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of the manufacturer.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any data, software or other intellectual property contained therein.

The manufacturer reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by the manufacturer, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

THE MANUFACTURER MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THESE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Index

Printed in USA

SC34-2593-00