



Systems and Technology Group

Developing Code for Cell - Mailboxes

Course Code: L3T2H1-55
Cell Ecosystem Solutions Enablement

Course Objectives – Things you will learn

- Cell communication mechanisms
 - mailboxes (this course) and DMA (another course)
- Mailbox queues for PPE and SPU
- Mailbox features and characteristics
- How to read and write mailboxes
- SPU outbound mailboxes and example
- SPU inbound mailbox and example
- PPE mailbox queue – PPE and SPU calls
- SPU mailbox queue – PPE and SPU calls

Course Agenda

- **Cell Communication Mechanisms**
- **Mailboxes**
 - How Mailboxes Were Sent
 - Mailbox Channels and their Associated MMIO Registers
- **Reading and Writing Mailboxes**
- **SPU Outbound Mailboxes**
 - Writing SPU Write Outbound Mailbox Data
 - Waiting to Write SPU Write Outbound Mailbox Data
 - How to Poll for or Block on an SPU Write Outbound Mailbox Available Event
 - How PPE Software can read from the SPU Write Outbound Mailbox of an SPE
 - Example shows how PPE software can read from the SPU Write Outbound Mailbox of an SPE
- **SPU Inbound Mailbox**
 - SPU Read Inbound Mailbox Channel
 - SPU Read Inbound Mailbox vs. SPU Write Outbound Mailboxes
 - How to write four 32-bit Messages to the PPU Read Inbound Mailbox of a Particular SPU from the PPE
- **PPE Mailbox Queue – PPE Calls, SPU Calls**
 - PPE Interrupting Mailbox Queue – PPE Calls
- **SPU Mailbox Queue – PPE Calls, SPU Calls**
- **Using mailboxes with macros defined in `libspe.h` and `spu_mfcio.h`**

Trademarks - Cell Broadband Engine [™] is a trademark of Sony Computer Entertainment, Inc.

Overview Cell Communication Mechanisms

- **Mailboxes**
 - between PPE and SPEs
- **DMA**
 - between PPE and SPEs
 - between one SPE and another
- **Used for**
 - Synchronization
 - Error reporting
 - Communication
 - Monitor SPU status
- **Some aspects of what is described are architecture and some aspects are implementation; e.g.:**
 - queues are architecture
 - queue sizes are implementation

Mailboxes

Each MFC provides three mailbox queues of 32 bit each:

1. PPE (“SPU outbound”) mailbox queue

- SPE writes, PPE reads
- 1 deep
- SPE stalls writing to full mailbox

2. PPE (“SPU outbound”) interrupt mailbox queue

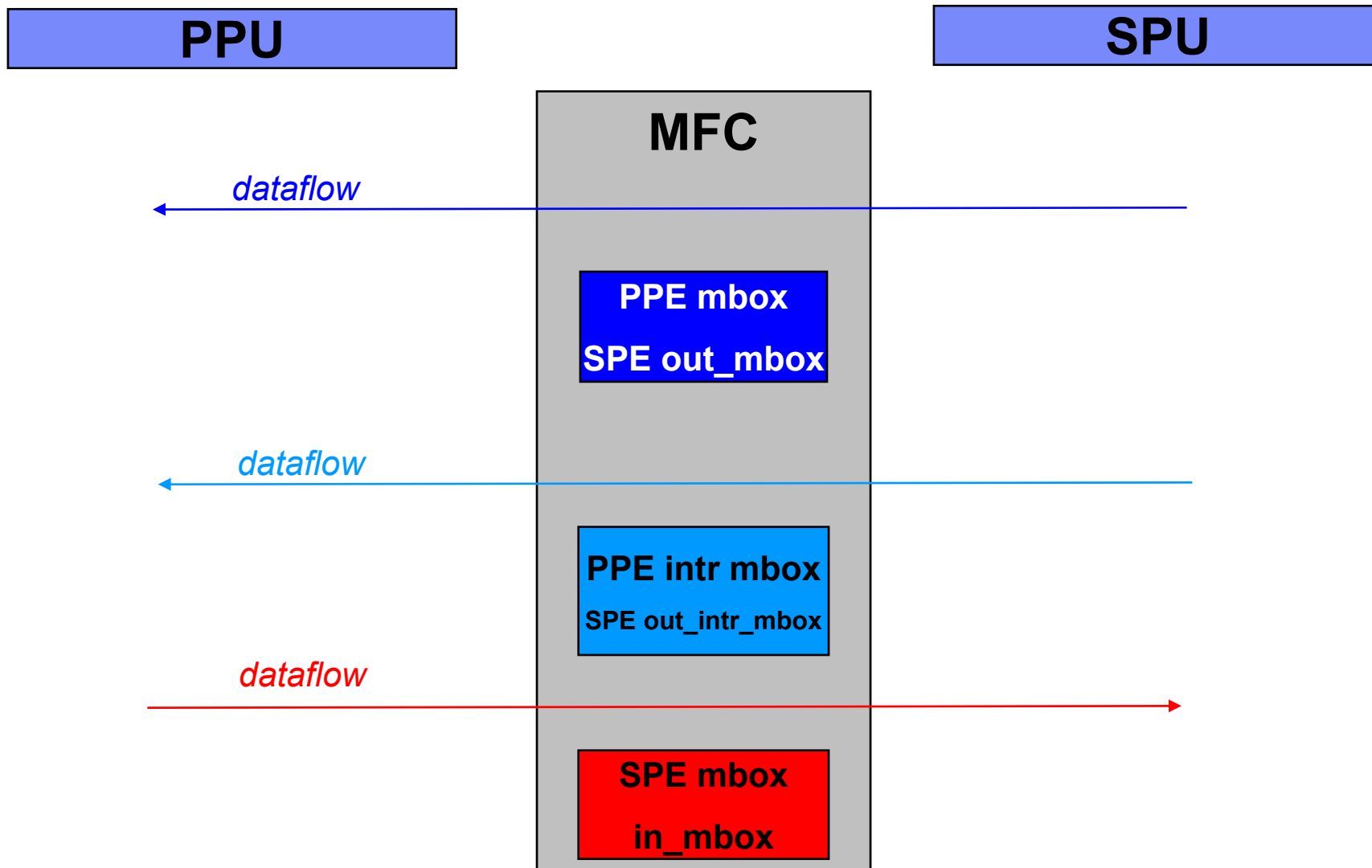
- like PPE mailbox queue, but an interrupt is posted to the PPE when the mailbox is written

3. SPU (“SPU inbound”) mailbox queue

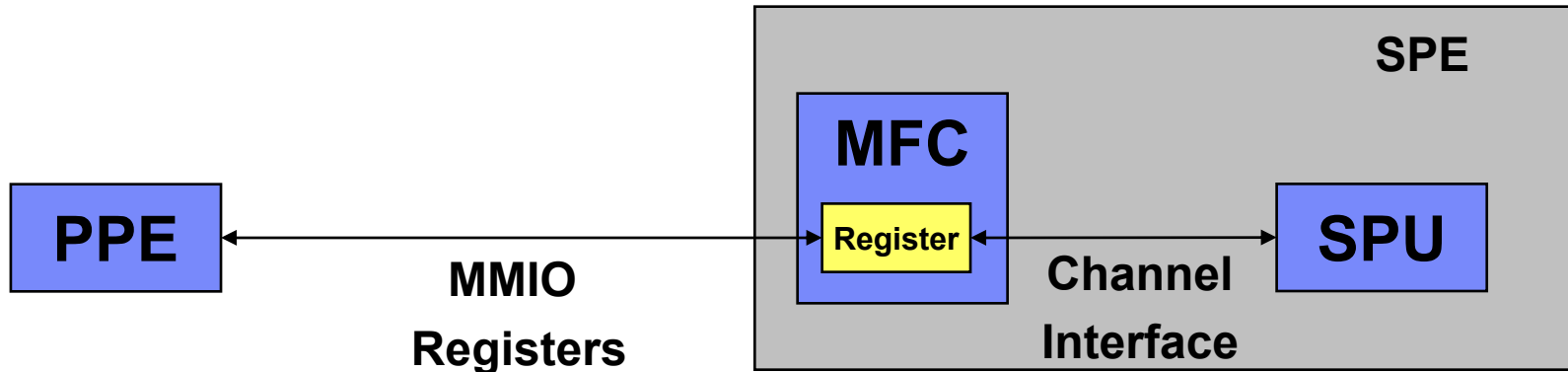
- PPE writes, SPE reads
- 4 deep
- can be overwritten

➤ **Each mailbox entry is a fullword**

Mailboxes Overview



Mailbox Architecture



Find problem state address and
offsets to memory mapped registers
Write directly into memory

Channel Instructions
`spu_writch(<channel>, <value>)`
`spu_readch(<channel>)`
`spu_readchcnt(<channel>)`

How Mails Are Sent

Channel

- SPE (outgoing)
 - write the 32-bit message value to either its two outbound mailbox channels
- SPE (incoming)
 - reads a message in the inbound mailbox

MMIO Reg

- PPE and other devices (incoming)
 - read message in outbound mailbox by reading the MMIO register in the SPE's MFC
- PPE and other devices (outgoing)
 - send by writing the associated MMIO register
- For interrupts associated with the SPU Write Outbound Interrupt Mailbox,
 - **no ordering of the interrupt and previously issued MFC commands**

Mailbox Channels and their Associated MMIO Registers

SPE Channel #	Name	Channel Interface					MMIO Register Interface				
		Mnemonic	Max. Entries	Blocking	R/W	Width (bits)	Offset From Base	Mnemonic	Max. Entries	R/W	Width (bits)
28	SPU Write Outbound Mailbox	SPU_WrOutMbox	1	yes	W	32	X'04004'	SPU_Out_Mbox	1	R	32
29	SPU Read Inbound Mailbox	SPU_RdInMbox	4	yes	R	32	X'0400C'	SPU_In_Mbox	4	W	32
30	SPU Write Outbound Interrupt Mailbox ¹	SPU_WrOutIntrMbox	1	yes	W	32	X'04000'	SPU_Out_Intr_Mbox	1	R	64
—	SPU Mailbox Status	—	—	—	—	—	X'04014'	SPU_Mbox_Stat	1	R	32

1. Access to this MMIO register is available only to privileged PPE software.

Functions of Mailbox Channels (SPU)

Channel Interface	SPU Read or Write	Functions
SPU_WrOutMbox	W	Writes message data to the outbound mailbox.
SPU_RdInMbox	R	Returns the next message data from the inbound mailbox
SPU_WrOutIntrMbox	W	Writes message data to the outbound interrupt mailbox.

Functions of Mailbox MMIO Registers (PPU)

MMIO Register	PPE Read or Write	Functions
SPU_Out_Mbox	R	Returns the message data from the corresponding SPU outbound mailbox.
SPU_In_Mbox	W	Writes message data to the SPU inbound mailbox.
SPU_Out_Intr_Mbox ¹	R	Returns the message data from the corresponding SPU outbound interrupt mailbox.
SPU_Mbox_Stat	R	Returns the number of available mailbox entries.

1. Access to the SPU_Out_Intr_Mbox MMIO register is available only to privileged PPE software.

Reading and Writing Mailboxes

SPU Outbound Mailboxes

SPU Outbound Mailboxes

■ SPU Write Outbound Mailbox Channel

- The value **written** to the SPU Write Outbound Mailbox channel SPU_WrOutMbox is entered into the outbound mailbox in the MFC if the mailbox has capacity to accept the value.
- If the mailbox can **accept** the value, the channel count for SPU_WrOutMbox is decremented by '1'.
- If the outbound mailbox is **full**, the channel count will read as '0'.
- If SPE software writes a value to SPU_WrOutMbox when the channel count is '0', the SPU will **stall** on the write.
- The SPU **remains stalled** until the PPE or other device reads a message from the outbound mailbox by reading the MMIO address of the mailbox.
- When the mailbox is **read** through the MMIO address, the channel count is incremented by '1'.

SPU Outbound Mailboxes (Cont'd)

▪ SPU Write Outbound Interrupt Mailbox Channel

- The value **written** to the SPU Write Outbound Interrupt Mailbox channel (SPU_WrOutIntrMbox) is entered into the outbound interrupt mailbox if the mailbox has capacity to accept the value.
- If the mailbox can **accept** the message, the channel count for SPU_WrOutIntrMbox is decremented by '1', and an **interrupt is raised** in the PPE or other device, depending on interrupt enabling and routing.
- There is no ordering of the interrupt and previously issued MFC commands.
- If the outbound interrupt mailbox is **full**, the channel count will read as '0'.
- If SPE software writes a value to SPU_WrOutIntrMbox when the channel count is '0', the SPU will **stall** on the write.
- The SPU **remains stalled** until the PPE or other device reads a mailbox message from the outbound interrupt mailbox by reading the MMIO address of the mailbox.
- When this is done, the channel count is **incremented** by '1'.

Writing SPU Write Outbound Mailbox Data

- SPE software can write to the SPU Write Outbound Mailbox channel to put a mailbox message in the SPU Write Outbound Mailbox.
- Sufficient space
 - **Yes : immediate return**
 - **No: SPU will stall until the PPE reads from this mailbox**
- How to write to the SPU Write Outbound Mailbox.

```
unsigned int mb_value;  
spu_writetech(SPU_WrOutMbox, mb_value);
```

Waiting to Write SPU Write Outbound Mailbox Data

- To avoid SPU stall, SPU can use the read-channel-count instruction on the SPU Write Outbound Mailbox channel to determine if the queue is empty before writing to the channel.
- If the read-channel-count instruction returns '0', the SPU Write Outbound Mailbox Queue is full.
- If the read channel-count instruction returns a non-zero value, the value indicates the number of free entries in the SPU Write Outbound Mailbox Queue.
- When the queue has free entries, the SPU can write to this channel without stalling the SPU.
- How to poll SPU Write Outbound Mailbox or SPU Write Outbound Interrupt Mailbox.

```
/*  
 * To write the value 1 to the SPU Write Outbound Interrupt Mailbox instead  
 * of the SPU Write Outbound Mailbox, simply replace SPU_WrOutMbox  
 * with SPU_WrOutIntrMbox in the following example.  
 */  
unsigned int mb_value;  
do {  
/*  
 * Do other useful work while waiting.  
 */  
} while (!spu_readchcnt(SPU_WrOutMbox)); // 0 → full, so something useful  
spu_writetech(SPU_WrOutMbox, mb_value);
```


How to Poll for or Block on an SPU Write Outbound Mailbox Available Event

```
#define MBOX_AVAILABLE_EVENT 0x00000080
unsigned int event_status;
unsigned int mb_value;
spu_writetech(SPU_WrEventMask, MBOX_AVAILABLE_EVENT);
do {
    /*
     * Do other useful work while waiting.
     */
} while (!spu_readchcnt(SPU_RdEventStat));
event_status = spu_readch(SPU_RdEventStat); /* read status */
spu_writetech(SPU_WrEventAck, MBOX_AVAILABLE_EVENT); /* acknowledge event */
spu_writetech(SPU_WrOutMbox, mb_value); /* send mailbox message */
```

- **NOTES:** To block, instead of poll, simply delete the do-loop above.

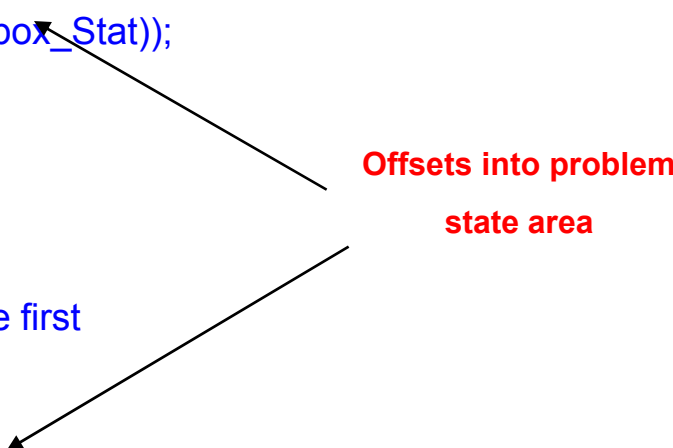
PPU reads SPU Outbound Mailboxes

- PPU must check Mailbox Status Register first
 - **check that unread data is available in the SPU Outbound Mailbox or SPU Outbound Interrupt Mailbox**
 - **otherwise, stale or undefined data may be returned**
- To determine that unread data is available
 - **PPE reads the Mailbox Status register**
 - **extracts the count value from the SPU_Out_Mbox_Count field**
- count is
 - **non-zero → at least one unread value is present**
 - **zero → PPE should not read but poll the Mailbox Status register**

Example: PPE reads from the SPU Outbound Mailbox

```
void *ps = spe_get_ps(speid); // system call assumed to return base address of problem state are
                               // might vary, depending on libspe version

unsigned int mb_status;
unsigned int new;
unsigned int mb_value;
do {
    mb_status = *((volatile unsigned int *) (ps + SPU_Mbox_Stat));
    new = mb_status & 0x000000FF;
} while ( new == 0 );
/*
 * Issue an eieio instruction to ensure that the last
 * Mailbox Status Register read is performed prior to the first
 * SPU Write Outbound Mailbox Register read.
 */
__asm__("eieio");
mb_value = *((volatile unsigned int *) (ps + SPU_Out_Mbox));
```



Offsets into problem state area

SPU Inbound Mailbox

SPU Inbound Mailbox

- The MFC provides one mailbox for a PPE to send information to an SPU
 - the SPU Read Inbound Mailbox.
- This mailbox has **four** entries
 - i.e. PPE can have up to four 32-bit messages pending at a time in the SPU Read Inbound Mailbox

SPU Read Inbound Mailbox Channel

- Mailbox is FIFO queue
 - If the SPU Read Inbound Mailbox channel (SPU_RdInMbox) has a message, the value read from the mailbox is the oldest message written to the mailbox.
- Mailbox Status (empty: channel count =0)
 - If the inbound mailbox is empty, the SPU_RdInMbox channel count will read as '0'.
- SPU stalls on reading empty mailbox
 - If SPE software reads from SPU_RdInMbox when the channel count is '0', the SPU will stall on the read. The SPU remains stalled until the PPE or other device writes a message to the mailbox by writing to the MMIO address of the mailbox.
- When the mailbox is written through the MMIO address, the channel count is incremented by '1'.
- When the mailbox is read by the SPU, the channel count is decremented by '1'.

SPU Read Inbound Mailbox vs. SPU Write Outbound Mailboxes

- The SPU Read Inbound Mailbox can be overrun by a PPE.
- A PPE writing to the SPU Read Inbound Mailbox will not stall when this mailbox is full.
- When a PPE overruns the SPU Read Inbound Mailbox, mailbox message data will be lost.

How to write four 32-bit Messages to the PPU Read Inbound Mailbox of a Particular SPU from the PPE

```
void *ps = spe_get_ps(speid); // see previous example !!
unsigned int j,k = 0;
unsigned int mb_status;
unsigned int slots;
unsigned int mb_value[4] = {0x1, 0x2, 0x3, 0x4};
do {
    /*
    * Poll the Mailbox Status Register until the
    * SPU_In_Mbox_Count field indicates there is at
    * least one slot available in the SPU Read Inbound
    * Mailbox.
    */
    do {
        mb_status = *((volatile unsigned int *) (ps +
            SPU_Mbox_Stat));
        slots = (mb_status & 0x0000FF00) >> 8;
    } while ( slots == 0 ); // as long as full
```

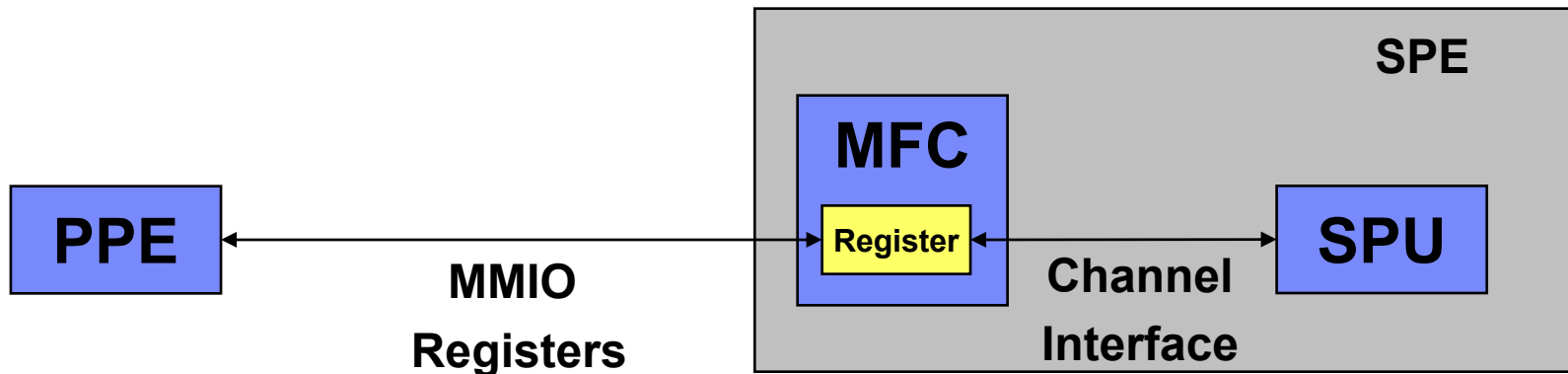
```
/* Issue an eieio instruction to ensure
that the last Mailbox Status Register
read is performed prior to the first
SPU Read Inbound Mailbox Register
write.
*/
__asm__("eieio");
for (j=0; j<slots && k < 4; j++) {
    *((volatile unsigned int *) (ps +
        SPU_In_Mbox)) = mb_value[k++];
}
} while ( k < 4 );
```


How SPU Reads From the Incoming Mailbox

```
unsigned int mb_value;
do {
    /*
     * Do other useful work while waiting.
     */
} while (!spu_readcnt(SPU_RdInMbox));
mb_value = spu_readch(SPU_RdInMbox);
```

Access to Mailboxes using macros defined in `libspe.h` (PPU) and `spu_mfcio.h` (SPU)

Mailbox Architecture



Find problem state address and offsets to memory mapped registers
Write directly into memory

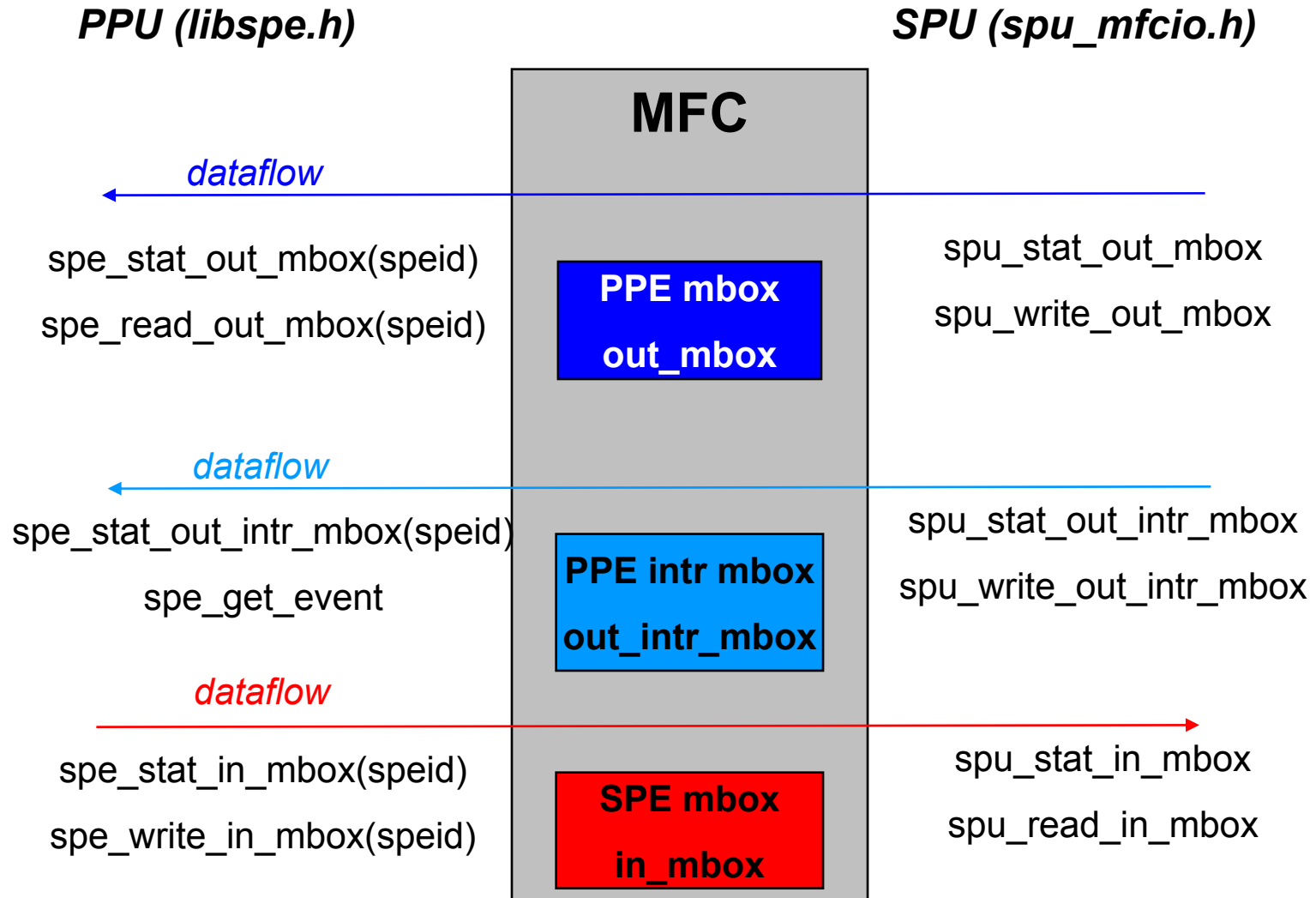
Macros defines in `libspe.h`
(requiring system calls)

→ easier, but slower

Channel Instructions
`spu_writch(<channel>, <value>)`
`spu_readch(<channel>)`
`spu_readchcnt(<channel>)`

Macros defined in `cbe_mfcio.h`

Mailboxes Overview



PPE Access to Mailboxes

- PPE can derive “addresses” of mailboxes from spe thread id
- First, create SPU thread, e.g.:

```
speid_t spe_id;  
spe_id = spe_create_thread(0,spu_load_image,NULL,NULL,-1,0);
```

 - spe_id has type speid_t (normally an int)
- PPE mailbox calls use **spe_id** to identify desired SPE’s mailbox
- Functions are in **libspe.a**

PPE Mailbox Queue – PPE Calls (libspe.h)

- **“SPU outbound” mailbox**
- **Check mailbox status:**
 - unsigned int count;
 - count = spe_stat_out_mbox(spe_id);
 - count = 0 → no data in the mailbox
 - otherwise, count = number of incoming 32-bit words in the mailbox
- **Get mailbox data:**
 - unsigned int data;
 - data = spe_read_out_inbox(spe_id);
 - data contains next 32-bit word from mailbox
 - routine is non-blocking
 - routine returns MFC_ERROR (0xFFFFFFFF) if no data in mailbox

PPE Mailbox Queues – SPU Calls (spu_mfcio.h)

- **“SPU outbound” mailbox**
- **Check mailbox status:**
 - `unsigned int count;`
 - `count = spu_stat_out_mbox();`
 - `count = 0` → mailbox is full
 - otherwise, `count` = number of available 32-bit entries in the mailbox
- **Put mailbox data:**
 - `unsigned int data;`
 - `spu_write_out_mbox(data);`
 - data written to mailbox
 - routine blocks if mailbox contains unread data

PPE Interrupting Mailbox Queue – PPE Calls

- **“SPU outbound” interrupting mailbox**
- **Check mailbox status:**
 - unsigned int count;
 - `count = spe_stat_out_intr_mbox(spe_id);`
 - count = 0 → no data in the mailbox
 - otherwise, count = number of incoming 32-bit words in the mailbox
- **Get mailbox data:**
 - interrupting mailbox is a privileged register
 - user PPE applications read mailbox data via `spe_get_event`

PPE Interrupting Mailbox Queues – SPU Calls

- **“SPU outbound” interrupting mailbox**
- **Put mailbox data:**
 - unsigned int data;
 - `spe_write_out_intr_mbox(data);`
 - data written to interrupting mailbox
 - routine blocks if mailbox contains unread data
- **defined in `spu_mfcio.h`**

SPU Mailbox Queue – PPE Calls (libspe.h)

- **“SPU inbound” mailbox**
- **Check mailbox status:**
 - `unsigned int count;`
 - `count = spe_stat_in_mbox(spe_id);`
 - `count = 0` → mailbox is full
 - otherwise, `count` = number of available 32-bit entries in the mailbox
- **Put mailbox data:**
 - `unsigned int data, result;`
 - `result = spe_write_in_mbox(spe_id,data);`
 - data written to next 32-bit word in mailbox
 - mailbox can overflow
 - routine returns `0xFFFFFFFF` on failure

SPU Mailbox Queue – SPU Calls (spu_mfcio.h)

- **“SPU inbound” mailbox**
- **Check mailbox status:**
 - unsigned int count;
 - count = spu_stat_in_mbox();
 - count = 0 → no data in the mailbox
 - otherwise, count = number of incoming 32-bit words in the mailbox
- **Get mailbox data:**
 - unsigned int data;
 - data = spu_read_in_mbox();
 - data contains next 32-bit word from mailbox
 - routine blocks if no data in mailbox

(c) Copyright International Business Machines Corporation 2005.
All Rights Reserved. Printed in the United States September 2005.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both.

IBM	IBM Logo	Power Architecture
-----	----------	--------------------

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

All information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in applications such as implantation, life support, or other hazardous uses where malfunction could result in death, bodily injury, or catastrophic property damage. The information contained in this document does not affect or change IBM product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All information contained in this document was obtained in specific environments, and is presented as an illustration. The results obtained in other operating environments may vary.

While the information contained herein is believed to be accurate, such information is preliminary, and should not be relied upon for accuracy or completeness, and no representations or warranties of accuracy or completeness are made.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN "AS IS" BASIS. In no event will IBM be liable for damages arising directly or indirectly from any use of the information contained in this document.

IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY 12533-6351

The IBM home page is <http://www.ibm.com>
The IBM Microelectronics Division home page is
<http://www.chips.ibm.com>