KeyStone Architecture
Semaphore2 Hardware Module

User Guide

Literature Number: SPRUGS3A
April 2012

Release History

Release	Date	Description/Comments
SPRUGS3A	April 2012	Updated graphics to generalize document (Page 1-3) Edited text throughout to make descriptions more generic. (Page 2-2)
SPRUGS3	November 2010	Initial Release

Contents

List of Tables

List of Figures

Preface

About This Manual

This document describes the operation of the Semaphore2 hardware module. The semaphore2 module is accessible across all the cores on a multicore DSP. The module supports up to 64 independent semaphores that help the application implement shared-resource protection across multiple cores. Each of the semaphores can be accessed by the cores in direct, indirect, or combined modes.

Notational Conventions

This document uses the following conventions:

- Commands and keywords are in **boldface** font.
- Arguments for which you supply values are in *italic* font.
- Terminal sessions and information the system displays are in `screen font`.
- Information you must enter is in **`boldface screen font`**.
- Elements in square brackets ([ ]) are optional.

Notes use the following conventions:

Note—Means reader take note. Notes contain helpful suggestions or references to material not covered in the publication.

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

CAUTION—Indicates the possibility of service interruption if precautions are not taken.

WARNING—Indicates the possibility of damage to equipment if precautions are not taken.

Related Documentation from Texas Instruments

C66x CorePac User Guide	SPRUGW0
Hardware Design Guide for KeyStone Devices	SPRABI2
Interrupt Controller (INTC) for KeyStone Devices User Guide	SPRUGW4
Memory Protection Unit (MPU) for KeyStone Devices User Guide	SPRUGW5
Multicore Navigator for KeyStone Devices User Guide	SPRUGR9
Multicore Programming Guide	SPRAB27
Multicore Shared Memory Controller (MSMC) for KeyStone Devices User Guide	SPRUGW7
Network Coprocessor (NETCP) *for KeyStone Devices User Guide*	SPRUGZ6

Trademarks

Chapter 1

Introduction

This document describes the operation of the Semaphore2 hardware module. The semaphore2 module is accessible across all the cores on a multicore DSP. The module supports up to 64 independent semaphores that help the application implement shared-resource protection across multiple cores. Each of the semaphores can be accessed by the cores in direct, indirect, or combined modes.

1.1 Purpose of the Semaphore2 Hardware Module

In a multicore environment—where system resources must be shared—it is important to control simultaneous accesses to the available resources. To ensure correct system operation, it is necessary to limit access to a resource by one—and only one—core at a time; that is, it is necessary to provide mutual exclusion for resources shared across multiple cores.

The Semaphore2 module provides a mechanism that applications can use to implement mutual exclusion of shared resources across multiple cores.

1.2 Features

The Semaphore2 module supports the following features:

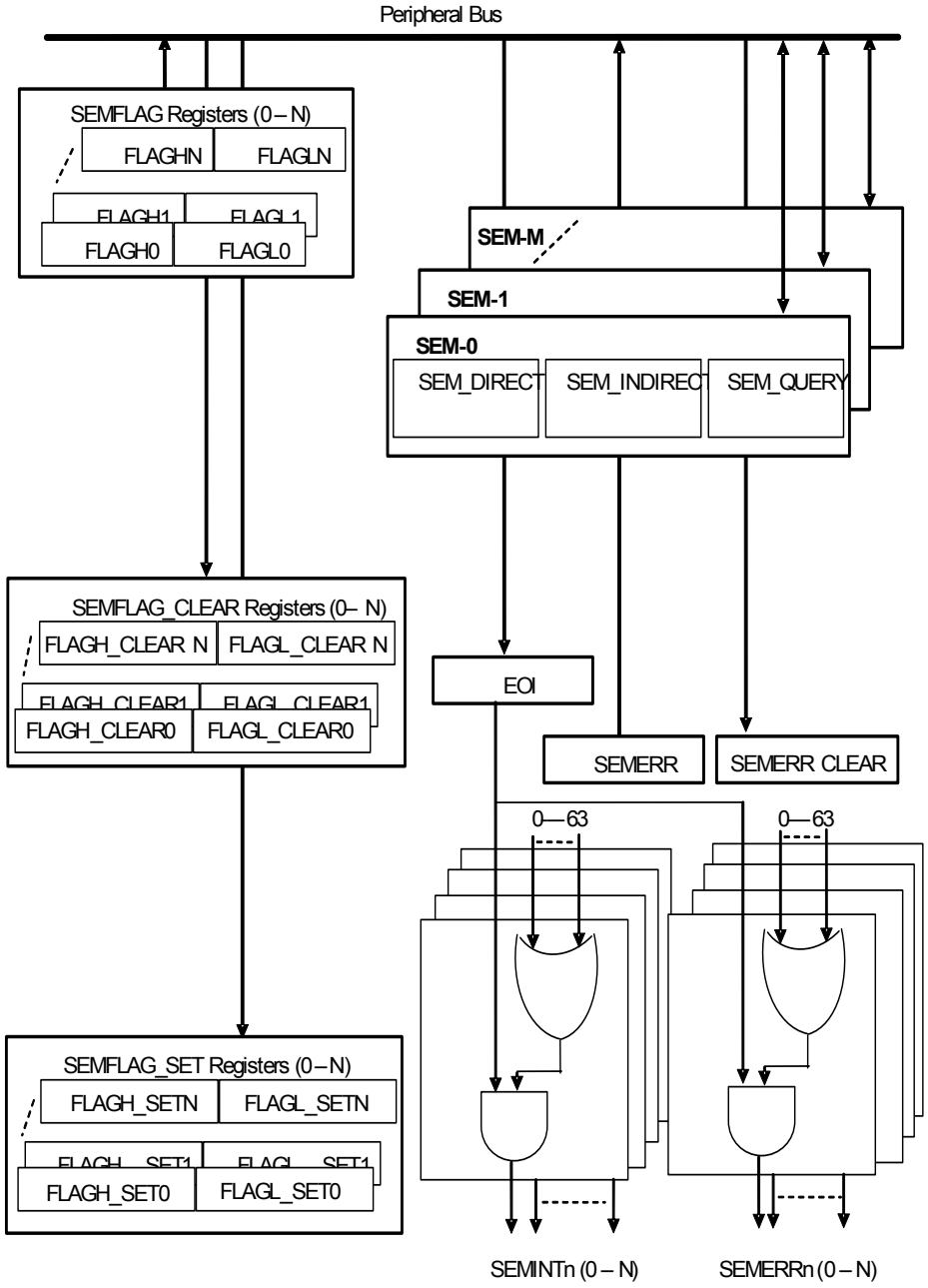
- Provides mutual exclusion for a shared resource
- A maximum of 64 independent semaphores
- Semaphore request methods
  - Direct request
  - Indirect request
  - Combined request
- Endian independent
- Atomic semaphore access
- Lock-out mechanism for used semaphores
- Queued requests for used semaphores
- Semaphores access grant interrupt for queued requests
- Allows the application to check the status of any of the semaphores
- Error detection and interrupts

1.3 Functional Block Diagram

Figure 1-1 shows the block diagram of the semaphore module. The module supports up to 64 semaphores that can be accessed by all four cores. The module generates two sets of interrupts: semaphore grant interrupt (SEMINTn) and error interrupt (SEMERRn). In the figure, 'N' denotes the number of cores available and 'M' denotes the number of semaphores (maximum of 64) available on a device.

Note—Refer to the device-specific datasheet for information on how many cores and semaphores are available on a specific device.

Figure 1-1 Semaphore Module Block Diagram



1.4 Industry Standard(s) Compliance Statement

This peripheral does not conform to any specific industry standard.

Chapter 2

Overview

The semaphore module can be used as an arbiter to ensure mutual exclusivity when sharing resources over multiple cores in a multicore device. It provides up to 64 independent semaphores that can be acquired and released by the application.

2.1 Architecture

Each of the semaphores is controlled by either reading from or writing to one of the three registers: SEM_DIRECT, SEM_INDIRECT, or SEM_QUERY. These three registers form a set and each semaphore is attached to one such set as shown in Figure 1-1 "Semaphore Module Block Diagram" on page 1-3. The semaphore module can service requests using one of three methods: the direct request method, the indirect request method, or the combined request method. These three methods provide flexibility to the application to either implement a polling-based system or an interrupt-based callback mechanism for acquiring or locking a semaphore.

None of the semaphores on the module are mapped to any resource or module on the device. The application can choose to map any of the available semaphores to protect any resource. There is no support provided by the semaphore module to store the semaphore to resource mapping information. The mapping information must be maintained by the software application.

2.2 Reset Considerations

2.2.1 Software Reset Considerations

The semaphore module can be reset through software by writing to the reset bit of the *SEM_RST_RUN* register. On reset, all semaphores are in 'FREE' state and all the errors cleared.

2.2.2 Hardware Reset Considerations

On reset, all semaphores are in 'FREE' state and all the errors cleared.

2.3 Interrupt Support

2.3.1 Interrupt Events and Requests

The semaphore module can generate an interrupt to any of the cores on the device either to signal semaphore grant or an error. The module generates two sets of interrupts: semaphore grant (SEMINTn) and error (SEMERRn). Each set has the capability to interrupt any core (0-N) on the device.

The *SEMINTn* interrupt is routed to the interrupt controller (INTC) of each core while the *SEMERRn* is routed to the respective chip interrupt controller (CIC[n]).

2.4 DMA Event Support

All *'N'* semaphore grant events (SEMINTn) can be used to trigger a DMA channel.

The error events (SEMERRn) may be used to trigger DMA.

2.5 Emulation Considerations

During debug when using the emulator, the DSP(s) may be halted for debugging. During emulation halt, the debugger read to semaphore registers is ignored. That is, debugger read of semaphore registers like SEM_DIRECT or SEM_INDIRECT will not change the state of the semaphore peripheral.

However, during emulation halt, the debugger can always see the SEM_QUERY register to check the semaphore peripherals status.

Chapter 3

Direct Semaphore Request

Direct request is the simplest method used to request a semaphore. The request behaves as an atomic read and set operation. The result of a request is either to grant the semaphore to the requesting core or deny the request because the semaphore has already been granted to another core.

3.1 Issuing a Direct Request

A direct request is issued by ***reading*** the ***SEM_DIRECT*** register corresponding to the semaphore requested. As seen in Figure 1-1 "Semaphore Module Block Diagram" on page 1-3 there are up to 64 DIRECT registers, one corresponding to each of the semaphores.

If the semaphore is free, the read returns "0x1"; if the request is rejected, the read returns the ID of the core that currently owns the semaphore. See "Direct Register (SEM_DIRECTn)" on page 10-6 for a detailed description of the read value.

3.2 Interrupt Events

No semaphore grant (SEMINTn) interrupts are generated when the direct request method is used. However, an error interrupt may be generated. For more information, about semaphore access errors, see "Error Generation and Handling" on page 8-1.

Chapter 4

Indirect Semaphore Request

The indirect request method acquires a semaphore with one request. If a semaphore was not free using the direct request method, the application has to keep trying until it is granted a semaphore. With the indirect method, the request for a semaphore is submitted to a queue. The request queue is processed sequentially by the semaphore module on a "first-in-first-out" basis. The semaphore is granted to the core whose request is at the top of the request queue and is marked busy until it is released, at which point the next request (if any) is processed.

4.1 Issuing an Indirect Request

An indirect request is issued by ***writing*** '***0x0***' to the ***SEM_DIRECT***, ***SEM_INDIRECT***, or the ***SEM_QUERY*** registers corresponding to the semaphore requested. As shown in Figure 1-1 "Semaphore Module Block Diagram" on page 1-3, there are up to 64 sets of the three registers—one corresponding to each of the semaphores.

When a semaphore becomes available or is free, the semaphore is granted to the core that has its request at the top of the queue.

4.2 Interrupt Events

When a semaphore is granted to a core via a request made using the indirect method, a *SEMINTn* interrupt to that core is generated.

Chapter 5

Combined Semaphore Request

A combined semaphore request is a hybrid version of the direct and indirect methods. It behaves like a direct request if the semaphore is free; otherwise it behaves like an indirect request and posts a request in the queue.

5.1 Issuing an Indirect Request

A combined request is issued by ***reading*** the ***SEM_INDIRECT*** register corresponding to the semaphore being requested.

If the semaphore is free, it is granted to the core making the request and the read returns a "0x1".

If the semaphore it is not free, the resulting read returns the ID of the core that currently owns the semaphore. A request for the semaphore is also posted to the queue for that semaphore. Refer to "Indirect Register (SEM_INDIRECTn)" on page 10-7 for a detailed description of the read value. The semaphore is granted whenever the posted request is processed.

5.2 Interrupt Events

For a combined request where a semaphore is free at the time the request is made, a *SEMINTn* interrupt is not generated when the semaphore is granted to the requestor. However, if it is posted to the request queue because the semaphore was not free at the time of request, a *SEMINT* interrupt is generated to that core when the semaphore is granted to it.

Chapter 6

Releasing Semaphores

Whenever a core is finished using a shared resource, it must free the associated semaphore so that other processes waiting to use it can get access. To release a semaphore and set its state to FREE, the application must ***write*** a ***'0x1'*** to one of ***SEM_DIRECT***, ***SEM_INDIRECT,*** or ***SEM_QUERY*** register corresponding to that semaphore.

Note—A semaphore can be released only by a process that is running on the core that is currently its owner; any other core trying to release the semaphore will generate an error.

Chapter 7

Querying Semaphore Status

It is reasonable that the application would need to check the status of a semaphore without acquiring it. The semaphore module provides the ability to query the status of any of the semaphores.

To query the status of a semaphore, the application needs to **read** the ***SEM_QUERY*** register *(*"Query Register (SEM_QUERYn)" on page 10-8*)* corresponding to the semaphore. The query returns information about whether the semaphore is free; if it is not free, the application returns the ID of the current owner.

Chapter 8

Error Generation and Handling

There are four types of errors the semaphore module can generate:

- ***Already Free Error***: This error is generated when an attempt is made to free a semaphore that is already free.
- ***Illegal Free Error***: This error is generated when an attempt is made to free a semaphore by a core that is not the current owner.
- ***Already Own Error***: This error is generated when a core attempts to acquire a semaphore that it already owns.
- ***Already Requested Error***: This error is generated when a core attempts to acquire a semaphore for which it already has a pending request in the queue.

When an error is generated by the module, the *SEMERR* register is updated with the error code semaphore number (0-63) and the ID (0-N) of the core that caused the error. Refer to "Error Register (SEMERR)" on page 10-12 for a more detailed description.

The *SEMERR* register can be cleared by writing a '0x1' to the *SEMERR_CLEAR* register.

Note—The SEMERR register is influenced by the SEMERR interrupt state. To capture and register errors continuously besides clearing the SEMERR register, the application must also reset the SEMERR interrupt state each time a new error is detected. Refer to the interrupt handling section for a description of the steps for resetting the interrupt ("Servicing SEMERRn" on page 9-2).

Chapter 9

Interrupts

The semaphore module can generate two sets of interrupts (events):

- Semaphore Grant Interrupt (SEMINTn)
- Semaphore Error Interrupt (SEMERRn)

Each set has independent lines capable of addressing any specific core on the device.

9.1 Semaphore Grant Interrupt (SEMINTn)

The *SEMINTn* is generated whenever a pending request is granted for any of the semaphores. The interrupt is directed to the specific core that posted the original request. The bit marking the semaphore that was granted is set in either the ***SEMFLAGLn*** (semaphore < 32) or the ***SEMFLAGHn*** (semaphore >= 32) register corresponding to that core. Each time a *SEMINTn* is generated, the *SEMINTn* line to that core is disabled to prevent further semaphore interrupts from being generated.

9.1.1 Servicing SEMINTn

Whenever *SEMINTn* is received by a core, the following steps are performed by the application to clear the event and reenable the interrupt:

1. Read the *SEMFLAGLn* and/or the *SEMFLAGHn* register to determine the semaphore that was granted.
2. Write a '1' to the *SEMFLAGL_CLEAR* or the *SEMFLAGH_CLEAR* register bit corresponding to the semaphore that was granted.

   This clears the corresponding bit in the flag register.
3. Write the core ID number (0 to N-1) to the "End-of-Interrupt Register (SEM_EOI)" on page 10-5.

   This reenables the SEMINTn line to that core and enables the generation of further SEMINTn interrupts.

9.2 Semaphore Error Interrupt (SEMERRn)

The *SEMERRn* interrupt is generated whenever an error condition is detected. Refer to "Error Generation and Handling" on page 8-1 for a more detailed description of the error conditions. The ***SEMERR*** register is updated with the error condition and core that caused the condition. Refer to the register description in "Error Register (SEMERR)" on page 10-12 for a description of the fields. Each time a *SEMERRn* is generated, all the *SEMERRn* lines are disabled to prevent any further semaphore error interrupts from being generated.

9.2.1 Servicing SEMERRn

Whenever *SEMERRn* is received by a core, the following steps are performed by the application to clear the event and reenable the error interrupts.

1. Read the *SEMERR* register ("Error Register (SEMERR)" on page 10-12) to determine the error condition.
2. Clear the *SEMERR* register by writing a '0x1' to the *SEMERR_CLEAR* register ("Error Clear Register (SEMERR_CLEAR)" on page 10-13).
3. Write ***0x10*** to the *SEM_EOI* register ("End-of-Interrupt Register (SEM_EOI)" on page 10-5) to rearm the semaphore error interrupts to all cores.

Chapter 10

Registers

Table 10-1 lists the memory mapped registers on the semaphore module. See the device-specific data sheet for the exact address of the registers.

Table 10-1 Register Map (Part 1 of 2)

Offset	Acronym	Description	Section
0000h	SEM_PID	Peripheral Revision ID Register	Section 10.1 "Peripheral Revision ID Register (SEM_PID)" on page 10-3
0004h	RSVD	Reserved	
0008h	SEM_RST_RUN	Reset/Run Status Register	Section 10.2 "Reset/Run Status Register (SEM_RST_RUN)" on page 10-4
000Ch	SEM_EOI	End of Interrupt Register	Section 10.3 "End-of-Interrupt Register (SEM_EOI)" on page 10-5
0010h – 00FFh	RSVD	Reserved	
0100h	SEM_DIRECT0[1]	Semaphore-0 Direct Register	Section 10.4 "Direct Register (SEM_DIRECTn)" on page 10-6
0104h	SEM_DIRECT1[1]	Semaphore-1 Direct Register	Section 10.4 "Direct Register (SEM_DIRECTn)" on page 10-6
⋮	⋮	⋮	
01FCh	SEM_DIRECT63[1]	Semaphore-63 Direct Register	Section 10.4 "Direct Register (SEM_DIRECTn)" on page 10-6
0200h	SEM_INDIRECT0	Semaphore-0 Indirect Register	Section 10.5 "Indirect Register (SEM_INDIRECTn)" on page 10-7
0204h	SEM_INDIRECT1[1]	Semaphore-1 Indirect Register	Section 10.5 "Indirect Register (SEM_INDIRECTn)" on page 10-7
⋮	⋮	⋮	
02FCh	SEM_INDIRECT63[1]	Semaphore-63 Indirect Register	Section 10.5 "Indirect Register (SEM_INDIRECTn)" on page 10-7
0300h	SEM_QUERY0[1]	Semaphore-0 Query Register	Section 10.6 "Query Register (SEM_QUERYn)" on page 10-8
0304h	SEM_ QUERY1[1]	Semaphore-1 Query Register	Section 10.6 "Query Register (SEM_QUERYn)" on page 10-8
⋮	⋮	⋮	
03FCh	SEM_ QUERY63[1]	Semaphore-63 Query Register	Section 10.6 "Query Register (SEM_QUERYn)" on page 10-8
0400h	SEMFLAGL0[2]	Flag Register Low for Core-0	Section 10.7 "Flag Registers (SEMFLAGLm, SEMFLAGHm)" on page 10-9
0404h	SEMFLAGL1[2]	Flag Register Low for Core-1	Section 10.7 "Flag Registers (SEMFLAGLm, SEMFLAGHm)" on page 10-9
⋮	⋮	⋮	Section 10.7 "Flag Registers (SEMFLAGLm, SEMFLAGHm)" on page 10-9
043Ch	SEMFLAGL15[2]	Flag Register Low for Core-15	Section 10.7 "Flag Registers (SEMFLAGLm, SEMFLAGHm)" on page 10-9
0400h	SEMFLAGL_CLEAR0[2]	Flag Clear Register Low for Core-0	Section 10.8 "Flag Clear Registers (SEMFLAGL_CLEARm, SEMFLAGH_CLEARm)" on page 10-10
0404h	SEMFLAGL_CLEAR1[2]	Flag Clear Register Low for Core-1	Section 10.8 "Flag Clear Registers (SEMFLAGL_CLEARm, SEMFLAGH_CLEARm)" on page 10-10
⋮	⋮	⋮	
043Ch	SEMFLAGL_CLEAR15[2]	Flag Clear Register Low for Core-15	Section 10.8 "Flag Clear Registers (SEMFLAGL_CLEARm, SEMFLAGH_CLEARm)" on page 10-10
0440h	SEMFLAGH0[2]	Flag Register High for Core-0	Section 10.7 "Flag Registers (SEMFLAGLm, SEMFLAGHm)" on page 10-9

Table 10-1 Register Map (Part 2 of 2)

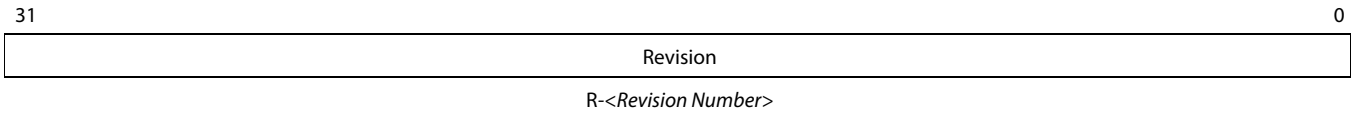
Offset	Acronym	Description	Section
0444h	SEMFLAGH1[2]	Flag Register High for Core-1	Section 10.7 "Flag Registers (SEMFLAGLm, SEMFLAGHm)" on page 10-9
⋮	⋮	⋮	
047Ch	SEMFLAGH15[2]	Flag Register High for Core-15	Section 10.7 "Flag Registers (SEMFLAGLm, SEMFLAGHm)" on page 10-9
0440h	SEMFLAGH_CLEAR0[2]	Flag Clear Register High for Core-0	Section 10.8 "Flag Clear Registers (SEMFLAGL_CLEARm, SEMFLAGH_CLEARm)" on page 10-10
0444h	SEMFLAGH_CLEAR1[2]	Flag Clear Register High for Core-1	Section 10.8 "Flag Clear Registers (SEMFLAGL_CLEARm, SEMFLAGH_CLEARm)" on page 10-10
⋮	⋮	⋮	
047Ch	SEMFLAGH_CLEAR15[2]	Flag Clear Register High for Core-15	Section 10.8 "Flag Clear Registers (SEMFLAGL_CLEARm, SEMFLAGH_CLEARm)" on page 10-10
0480h	SEMFLAGL_SET0[2]	Flag Set Register Low for Core-0	Section 10.9 "Flag Set Registers (SEMFLAGL_SETm, SEMFLAGH_SETm)" on page 10-11
0484h	SEMFLAGL_SET1[2]	Flag Set Register Low for Core-1	Section 10.9 "Flag Set Registers (SEMFLAGL_SETm, SEMFLAGH_SETm)" on page 10-11
⋮	⋮	⋮	
04BCh	SEMFLAGL_SET15[2]	Flag Set Register Low for Core-15	Section 10.9 "Flag Set Registers (SEMFLAGL_SETm, SEMFLAGH_SETm)" on page 10-11
04C0h	SEMFLAGH_SET0[2]	Flag Set Register High for Core-0	Section 10.9 "Flag Set Registers (SEMFLAGL_SETm, SEMFLAGH_SETm)" on page 10-11
04C4h	SEMFLAGH_SET1[2]	Flag Set Register High for Core-1	Section 10.9 "Flag Set Registers (SEMFLAGL_SETm, SEMFLAGH_SETm)" on page 10-11
⋮	⋮	⋮	
04FCh	SEMFLAGH_SET15[2]	Flag Set Register High for Core-15	Section 10.9 "Flag Set Registers (SEMFLAGL_SETm, SEMFLAGH_SETm)" on page 10-11
0500h	SEMERR	Error Register	Section 10.10 "Error Register (SEMERR)" on page 10-12
0504h	SEMERR_CLEAR	Error Clear Register	Section 10.11 "Error Clear Register (SEMERR_CLEAR)" on page 10-13
0508h	SEMERR_SET	Error Set Register	Section 10.12 "Error Set Register (SEMERR_SET)" on page 10-14
End of Table 10-1			

1. There are only as many SEM_DIRECTn, SEM_INDIRECTn and SEM_QUERYn registers as the number of semaphores that the device supports. Refer to the device datasheet for the details. The rest of the address space is reserved space.
2. There are only as many SEMFLAGLn, SEMFLAGHn, SEMFLAGL_CLEARn, SEMFLAGH_CLEARn, SEMFLAGL_SETn and SEMFLAGH_SETn registers as the number of cores on the device. The rest of the address space is reserved space.

10.1 Peripheral Revision ID Register (SEM_PID)

The Peripheral ID register contains the revision number and identification data of the semaphore2 peripheral. Refer to the device-specific datasheet for the register value for your device.

Figure 10-1 Peripheral Revision ID Register (SEM_PID)



R-<*Revision Number*>

Table 10-2 SEM_PID Field Description

Bit	Field	Access	Value	Description
31- 0	Revision	R	<Revision Number>	Module Revision number

10.2 Reset/Run Status Register (SEM_RST_RUN)

This register contains the reset status of the semaphore module. This register is also used to soft reset the module.

Figure 10-2 Reset/Run Status Register (SEM_RST_RUN)

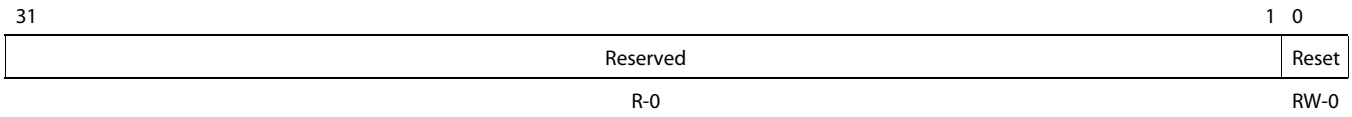


Table 10-3 SEM_RST_RUN Field Description

Bit	Field	Access	Value	Description
31–1	Reserved	R	0	Reserved field.
0	Reset	RW	0	R: Module is not ready. W: No affect.
			1	R: Module ready. W: Resets the module

10.3 End-of-Interrupt Register (SEM_EOI)

This register is used to rearm the interrupts.

Figure 10-3 End-of-Interrupt Register (SEM_EOI)

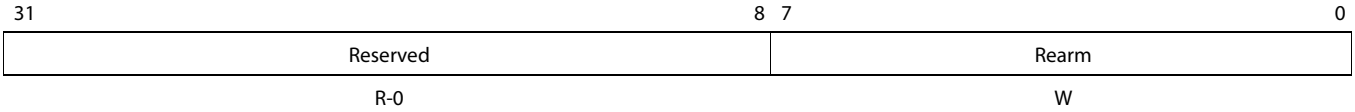


Table 10-4 SEM_EOI Field Description

Bit	Field	Access	Value	Description
31–8	Reserved	R	0	Reserved field.
7-0	Rearm	W	00h 01h : : : 10h	Re-arm SEMINT0 Re-arm SEMINT1 : : : Re-arm SEMERR

10.4 Direct Register (SEM_DIRECTn)

There are *n* direct registers, where *n* is the number of semaphores supported by the device. Each one is associated with and used to manage the respective semaphore.

Figure 10-4 Direct Register (SEM_DIRECTn)

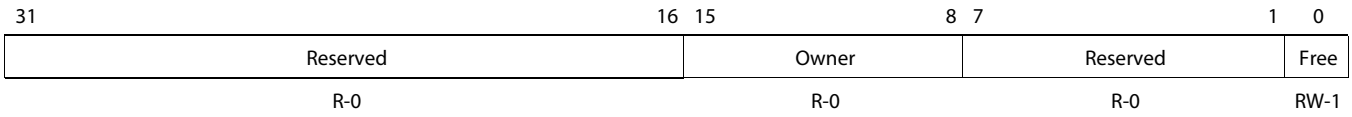


Table 10-5 SEM_DIRECTn Field Description

Bit	Field	Access	Value	Description
31–16	Reserved	R	0	Reserved field.
15 - 8	Owner	R	00h – 0Fh	FREE = 0: Core ID of current owner FREE = 1: Zero.
7-1	Reserved	R	0	Reserved field
0	Free	RW	00h	R: Semaphore is not granted W: Request posted to queue
			01h	R: Semaphore is granted to the reader. W: Semaphore is set free[1]

1. - Only if writer is the current owner

10.5 Indirect Register (SEM_INDIRECTn)

There are *n* indirect registers where *n* is the number of semaphores supported by the device. Each one is associated with and used to manage the respective semaphore.

Figure 10-5 Indirect Register (SEM_INDIRECTn)

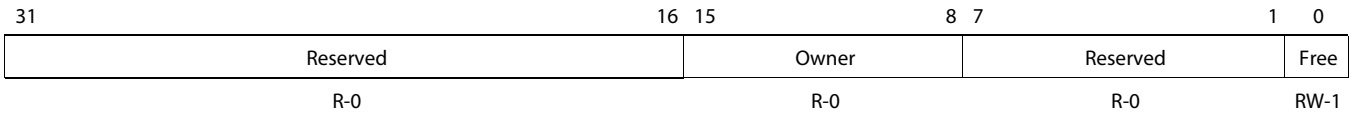


Table 10-6 SEM_INDIRECTn Field Description

Bit	Field	Access	Value	Description
31–16	Reserved	R	0	Reserved field.
15 - 8	Owner	R	00h – 0Fh	FREE = 0: Core ID of current owner FREE = 1: Zero.
7-1	Reserved	R	0	Reserved field
0	Free	RW	00h	R: Semaphore is not granted, request posted to queue. W: Request posted to queue
			01h	R: Semaphore is granted to the reader. W: Semaphore is set free[1]

1. Only if writer is the current owner

10.6 Query Register (SEM_QUERYn)

There are *n* query registers, where *n* is the number of semaphores supported by the device. Each one is associated with and used to manage the respective semaphore.

Figure 10-6 Query Register (SEM_QUERYn)

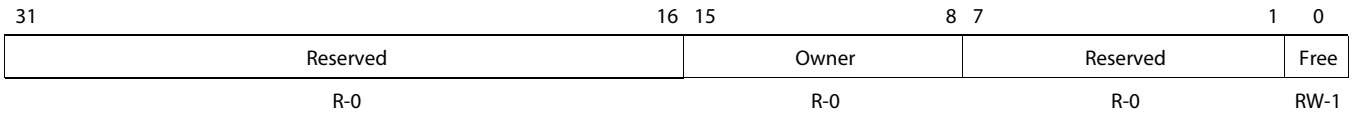


Table 10-7 SEM_QUERYn Field Description

Bit	Field	Access	Value	Description
31–16	Reserved	R	0	Reserved field.
15 - 8	Owner	R	00h – 0Fh	FREE = 0: Core ID of current owner FREE = 1: Zero.
7-1	Reserved	R	0	Reserved field
0	Free	RW	00h	R: Semaphore is not available. W: Request posted to queue
			01h	R: Semaphore is available. W: Semaphore is set free[1]

1. Only if writer is the current owner

10.7 Flag Registers (SEMFLAGLm, SEMFLAGHm)

There are M SEMFLAGL and M SEMFLAGH registers. M denotes the number of cores on the device. Refer to the device-specific datasheet for information regarding the number of cores on the device. The bits flag that an interrupt was generated to that core granting it the semaphore. If a bit is set, it does not particularly mean that the core still owns the semaphore.

10.7.1 Flag Register SEMFLAGLm

Figure 10-7 Flag Register SEMFLAGLm

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
F31	F30	F29	F28	F27	F26	F25	F24	F23	F22	F21	F20	F19	F18	F17	F16
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F15	F14	F13	F12	F11	F10	F9	F8	F7`	F6	F5	F4	F3	F2	F1	F0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Table 10-8 SEM_FLAGLm Field Description

Bit	Field	Access	Value	Description
n	FLAG n	R	0	No interrupt generated to core-m.
			1	Interrupt was generated core-m to signal that Semaphore-n was granted to it.

10.7.2 Flag Register SEMFLAGHm

Figure 10-8 Flag Register SEMFLAGHm

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
F63	F62	F61	F60	F59	F58	F57	F56	F55	F54	F53	F52	F51	F50	F49	F48
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F47	F46	F45	F44	F43	F42	F41	F40	F39	F38	F37	F36	F35	F34	F33	F32
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Table 10-9 SEM_FLAGHm Field Description

Bit	Field	Access	Value	Description
n	FLAG n+32	R	0	No interrupt generated to core-m.
			1	Interrupt was generated core-m to signal that Semaphore-n+32 was granted to it.

10.8 Flag Clear Registers (SEMFLAGL_CLEARm, SEMFLAGH_CLEARm)

There are M SEMFLAGL_CLEAR and M SEMFLAGH_CLEAR registers. M denotes the number of cores on the device. Refer to the device-specific datasheet for information regarding the number of cores on the device. The SEMFLAGL_CLEARm and the SEMFLAGH_CLEARm registers are used to clear the flag bits in the SEMFLAGL and SEMFLGH registers, respectively.

10.8.1 Flag Clear Register SEM_FLAGL_CLEARm

Figure 10-9 Flag Clear Register SEM_FLAGL_CLEARm

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
F31	F30	F29	F28	F27	F26	F25	F24	F23	F22	F21	F20	F19	F18	F17	F16
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F15	F14	F13	F12	F11	F10	F9	F8	F7`	F6	F5	F4	F3	F2	F1	F0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Table 10-10 SEM_FLAGL_CLEARm Field Description

Bit	Field	Access	Value	Description
n	FLAG n	W	0	No affect
			1	Flag bit n is cleared

10.8.2 Flag Clear Register SEM_FLAGH_CLEARm

Figure 10-10 Flag Clear Register SEM_FLAGH_CLEARm

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
F63	F62	F61	F60	F59	F58	F57	F56	F55	F54	F53	F52	F51	F50	F49	F48
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F47	F46	F45	F44	F43	F42	F41	F40	F39	F38	F37	F36	F35	F34	F33	F32
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Table 10-11 SEM_FLAGH_CLEARm Field Description

Bit	Field	Access	Value	Description
n	FLAG n+32	W	0	No effect
			1	Flag bit n+32 is cleared

10.9 Flag Set Registers (SEMFLAGL_SETm, SEMFLAGH_SETm)

There are M SEMFLAGL_SET and M SEMFLAGH_SET registers. M denotes the number of cores on the device. Refer to the device-specific datasheet for information regarding the number of cores on the device. The SEMFLAGL_SETm and the SEMFLAGH_SETm registers are used to set the flag bits in the SEMFLAGL and SEMFLGH registers respectively and also generate the associated SEMINTm interrupt. This can be used for testing and debugging purposes.

10.9.1 Flag Set Register SEM_FLAGL_SETm

Figure 10-11 Flag Set Register SEM_FLAGL_SETm

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
F31	F30	F29	F28	F27	F26	F25	F24	F23	F22	F21	F20	F19	F18	F17	F16
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F15	F14	F13	F12	F11	F10	F9	F8	F7`	F6	F5	F4	F3	F2	F1	F0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Table 10-12 SEM_FLAGL_SETm Field Description

Bit	Field	Access	Value	Description
n	FLAG n	W	0	No affect
			1	Flag bit n is set, SEMINTm is generated

10.9.2 Flag Set Register SEM_FLAGH_SETm

Figure 10-12 Flag Set Register SEM_FLAGH_SETm

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
F63	F62	F61	F60	F59	F58	F57	F56	F55	F54	F53	F52	F51	F50	F49	F48
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F47	F46	F45	F44	F43	F42	F41	F40	F39	F38	F37	F36	F35	F34	F33	F32
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Table 10-13 SEM_FLAGH_SETm Field Description

Bit	Field	Access	Value	Description
n	FLAG n+32	W	0	No effect
			1	Flag bit n+32 is set, SEMINTm is generated

10.10 Error Register (SEMERR)

The SEMERR register contains information pertaining to the last generated SEMERRm interrupt.

Figure 10-13 Error Register (SEMERR)

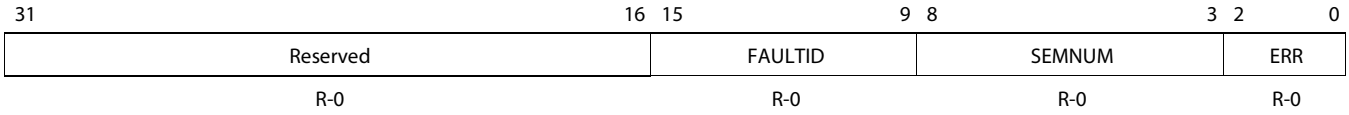


Table 10-14 SEMERR Field Description

Bit	Field	Access	Value	Description
31-16	Reserved	R	0	Reserved field.
15-9	FaultId	R	m	ID of the core that is responsible for the error
8-3	SEMNUM	R	n	Semaphore number associated with error.
2-0	ERR	R		Error Code:
			00h	No Error.
			01h	Already Free Error
			02h	Illegal Free Error
			03h	Already Own Error
			04h	Already Requested Error

10.11 Error Clear Register (SEMERR_CLEAR)

This SEMERR_CLEAR register is used to clear the error condition in the SEMERR register.

Figure 10-14 Error Clear Register (SEMERR_CLEAR)

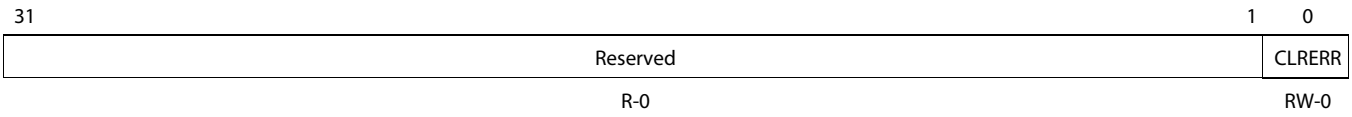


Table 10-15 SEMERR_CLEAR Field Description

Bit	Field	Access	Value	Description
31–1	Reserved	R	0	Reserved field.
0	CLRERR	W	0	No effect
			1	Clears the SEMERR register.

10.12 Error Set Register (SEMERR_SET)

The SEMERR_SET register is used to generate an error and the associated SEMERRm interrupt by setting the SEMERR register values. It can be used for testing and debug purposes.

Figure 10-15 Error Set Register (SEMERR_SET)

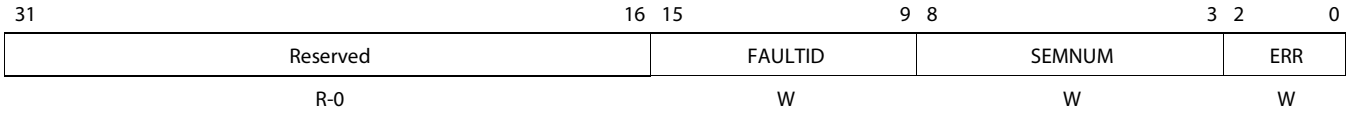


Table 10-16 SEMERR_SET Field Description

Bit	Field	Access	Value	Description
31-16	Reserved	R	0	Reserved field.
15-9	FAULTID	W	m	ID of the core that is responsible for the error
8-3	SEMNUM	W	n	Semaphore number associated with error.
2-0	ERR	W		Error Code:
			00h	No Error.
			01h	Already Free Error
			02h	Illegal Free Error
			03h	Already Own Error
			04h	Already Requested Error

Index

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Mobile Processors	www.ti.com/omap		
Wireless Connectivity	www.ti.com/wirelessconnectivity		
	TI E2E Community Home Page		e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265