TMS320TCl6487/8 Semaphore

User's Guide

Literature Number: SPRUEF6C
December 2006—Revised February 2008
Contents

Preface ............................................................................................................................ 5

1 Module Overview ........................................................................................................ 6
  1.1 Introduction ....................................................................................................... 6
  1.2 Semaphore Architecture ................................................................................... 6

2 Semaphore Operation .............................................................................................. 8
  2.1 Semaphore Access Modes .................................................................................. 8
  2.2 Interrupt Handling ............................................................................................ 8
  2.3 Error Handling .................................................................................................. 9
  2.4 Status Query ...................................................................................................... 9
  2.5 Releasing Semaphore Resources ...................................................................... 9

3 Emulation Considerations ....................................................................................... 9

4 Semaphore Usage Examples .................................................................................... 10
  4.1 CSL API Calls .................................................................................................. 10
  4.2 Accessing Semaphore Resource When Free .................................................... 12
  4.3 Accessing Semaphore Resource When Not Free ............................................. 13
  4.4 Accessing Semaphore Resource in Combined Mode ......................................... 14

5 Semaphore Registers .............................................................................................. 15
  5.1 Register Memory Map ........................................................................................ 15
  5.2 Register Descriptions ........................................................................................ 16
    5.2.1 Peripheral ID Register (PID) ....................................................................... 16
    5.2.2 End-of-Interrupt Register (EOI) ................................................................. 16
    5.2.3 Direct Register (DIRECTx) .......................................................................... 17
    5.2.4 Indirect Register (INDIRECTx) ..................................................................... 18
    5.2.5 Query Register (QUERYx) .......................................................................... 19
    5.2.6 Flag Register (FLAGx) ............................................................................... 20
    5.2.7 Flag Set Register (FLAG_SETx) ................................................................. 20
    5.2.8 Error Register (ERR) ................................................................................ 21
    5.2.9 Error Clear Register (ERROR_CLR) ........................................................... 22
List of Figures

1  TMS320TCI6487/8 Semaphore Block Diagram ................................................................. 7
2  Peripheral ID Register (PID) ............................................................................................. 16
3  End-of-Interrupt Register (EOI) ....................................................................................... 16
4  Direct Register (DIRECTx) ............................................................................................... 17
5  Indirect Register (INDIRECTx) ......................................................................................... 18
6  Query Register (QUERYx) ............................................................................................... 19
7  Flag Register (FLAGx) .................................................................................................... 20
8  Flag Set Register (FLAG_SETx) ...................................................................................... 20
9  Error Register (ERR) ...................................................................................................... 21
10 Error Clear Register (ERROR_CLR) .................................................................................. 22

List of Tables

1  Register Memory Map ....................................................................................................... 15
2  Peripheral ID Register (PID) Field Descriptions ............................................................ 16
3  End-of-Interrupt Register (EOI) Field Descriptions ......................................................... 16
4  Direct Register (DIRECTx) Field Descriptions ............................................................... 17
5  Indirect Register (INDIRECTx) Field Descriptions ......................................................... 18
6  Query Register (QUERYx) Field Descriptions ............................................................... 19
7  Flag Register (FLAGx) Field Descriptions .................................................................... 20
8  Flag Set Register (FLAG_SETx) Field Descriptions ....................................................... 20
9  Error Register (ERR) Field Descriptions ...................................................................... 21
10 Error Register (ERROR_CLR) Field Descriptions .......................................................... 22
About This Manual
The TMS320TCI6487/8 semaphore module is used to support atomic arbitration among multiple CPUs for shared resources/peripherals. This document describes the usage of the semaphore and some of the CSL calls used to configure/use semaphore module.

Terms and Abbreviations
- **CPU**— Central Processing Unit of the C64x+ Core
- **DSP**— Digital Signal Processor
- **CSL**— Chip Support Library
- **API**— Application Programmer Interface

Trademarks
All trademarks are the property of their respective owners.
1 Module Overview

1.1 Introduction

In the TMS320TCI6487/8 device, multiple CPUs attempt to access shared resources simultaneously. To avoid resource conflict, the semaphore module is used to access shared resources in mutual exclusion. Unlike the software flag checking mechanism, the semaphore module is atomic, to satisfy the read-modify-write operation successfully.

The semaphore module allows acquisition of a semaphore resource/peripheral through read operations and also by posting a write request.

The number of resources the semaphore can handle is parameterized and, for the TMS320TCI6487/8 device, the following parameters are pre-programmed:

- **NUM_SEM**: Number of semaphore resources. In the device, the value is 32.
- **QUEUE_DEPTH**: Number of entries in each request queue. In the device, the value is 2.

1.2 Semaphore Architecture

The semaphore module has unique interrupts for each of the DSP cores (masters) to identify when that master has acquired the requested resource. Likewise, there are unique error interrupts to each DSP core when a particular master attempts to access the semaphore resource that is already locked by the same/other master. For each semaphore peripheral there are three different registers associated with it to acquire the resource: direct, indirect, or combined mechanism.

Semaphore resources are not directly connected within the module. Through software programming hardware resources can be allocated to any of the semaphore resources. Figure 1 shows the basic building blocks of semaphore module.

In Figure 1, \( m \) is the number of semaphore resources (i.e., 32) and \( n \) is the number of the master (i.e., 3). There are three 32-bits registers associated with each semaphore resource: DIRECT, INDIRECT, and QUERY registers. For each DSP core there is a FLAG register to notify the status of all 32 semaphore resources. The ERR register shows the different types of errors and the semaphore peripheral ID and master ID that caused the specific error. Upon receiving the error or interrupt event, the DSP core clears the particular bit field of the FLAG register by programming the FLAG register and writing to the semaphore end-of-interrupt (EOI) register to re-arm or re-enable a particular master's error/interrupt line.
Figure 1. TMS320TCI6487/8 Semaphore Block Diagram
2 Semaphore Operation

Any of the DSP cores can use the shared semaphore resources.

2.1 Semaphore Access Modes

Any semaphore resource can be accessed by three different modes:

- **Direct Mode**
  In direct access mode, a resource is accessed by reading the DIRECTx (x is the semaphore peripheral number) register. Since there is no physical connection between the semaphore resources and the registers, any register can be mapped to any resource by software programming.

  If the resource is free, access is immediately granted to the requested core. The FREE bit of the DIRECTx register is set to 1 signifying that the resource is free and access is granted to the requested core. The OWNER field of the DIRECTx register is set to the requested core ID.

  If the resource is not available, no further action is taken by the semaphore module. The FREE bit of the DIRECTx register is set to 0 signifying that the resource is in use and the requested master should start a fresh request. In any case, there is no interrupt issued and there is no specific bit field set in the FLAGy (y is the granted core ID) register.

- **Indirect Mode**
  In indirect access mode, a resource is accessed by writing to the DIRECTx, INDIRECTx, or QUERYx (x is the semaphore peripheral number) register. Since there is no physical connection between the semaphore resources and the registers, any register can be mapped to any resource by software programming.

  If the resource is free, access is immediately granted to the requested core. The FREE bit of the DIRECTx, INDIRECTx, or QUERYx register is set to 1 signifying that the resource is free and access is granted to the requested core. The OWNER field of the DIRECTx, INDIRECTx, or QUERYx register is set to the requested core ID. An interrupt is immediately issued to the requested core.

  If the resource is not available, the request is added to the request queue assigned to the semaphore resource. When the resource becomes free, an interrupt is issued and the FREE bit of the DIRECTx, INDIRECTx, or QUERYx register is set to 1 signifying that access is now granted to the requested core. An interrupt is issued in this case. The specific bit field for the semaphore resource of the FLAGy (y is the granted core ID) register is set.

- **Combined Mode**
  In combined access mode, a resource is accessed by reading the INDIRECTx (x is the semaphore peripheral number) register. Since there is no physical connection between the semaphore resources and the registers, any register can be mapped to any resource by software programming.

  If the resource is free, access is immediately granted to the requested core. The FREE bit of the INDIRECTx register is set to 1 signifying that the resource is free and access is granted to the requested core. The OWNER field of the INDIRECTx register is set to the requested core ID. No grant interrupt or flag register settings happen in this case.

  If the resource is not available, the request is added to the request queue assigned to the semaphore resource. When the resource becomes free, an interrupt is issued and the FREE bit of the INDIRECTx register is set to 1 signifying that access is now granted to the requested core. An interrupt is issued in this case. The specific bit field for the semaphore resource of the FLAGy (y is the granted core ID) register is set.

2.2 Interrupt Handling

The first interrupt occurs when the pending queued request is serviced. To ensure that a re-arm of the master's interrupt occurs prior to the next access by the same or other resource, the master should:

- read the FLAGx (x is the particular master id) register
- clear the flag bit by programming the specific bit field for the semaphore resource of the FLAGx register
- write to the EOI register.
2.3 Error Handling

The first error interrupt occurs when there is a semaphore access error. To ensure that a re-arm of the master's error interrupt occurs prior to the next access violation, the master should:

• read the ERR register to determine the error type
• clear the error by programming the ERR_CLEAR register
• write to the EOI register.

2.4 Status Query

Before acquiring any semaphore resource, the requested core checks the availability of the resource by reading the QUERYx (x is the semaphore peripheral number) register. The FREE bit signifies whether a particular resource is free or not; when FREE=0 the resource is not free, when FREE=1 the resource is free. The OWNER field has a master ID that currently holds the resource (when FREE=0) or shows 0x00 (when FREE=1). Before taking any action on a resource, check the resource status.

2.5 Releasing Semaphore Resources

When the master that is granted access is finished with the shared resource, it must free the resource so that another master can access it. Writing 1 to the FREE bit of the DIRECTx, INDIRECTx, or QUERYx register releases the specific resource.

3 Emulation Considerations

During debug, when using the emulator, the CPU(s) may be halted. During emulation halt, the debugger reads to certain semaphore registers is ignored to avoid changing semaphore state. In other words, the debugger read of semaphore registers (DIRECTx, INDIRECTx) does not change the state of the semaphore peripheral. But, during emulation halt, the debugger checks the semaphore peripheral status via the QUERYx register.
Semaphore Usage Examples

4 Semaphore Usage Examples

This section provides some of the API call lists and examples to access semaphore resources.

4.1 CSL API Calls

This section provides the basic usage of the semaphore module using different CSL API calls. Three typical modes of accessing peripherals are illustrated with examples. Example 1 shows some of the API call lists.

Example 1. Sample API Call Lists

```c
#ifndef CSL_sem_API
#define HW_SEM_RELEASE 1
#define HW_SEM_REQUEST 0

// ENUMERATIONS

typedef enum {
    CSL_SEM_ID0 = 0,
    CSL_SEM_ID1 = 1,
    CSL_SEM_ID2 = 2,
} CSL_SemOwnerId;

typedef enum {
    CSL_SEM_ERR0 = 0,
    CSL_SEM_ERR1 = 1,
    CSL_SEM_ERR2 = 2,
    CSL_SEM_ERR3 = 3,
    CSL_SEM_ERR4 = 4,
} CSL_SemError;

typedef enum {
    CSL_SEM_NOTFREE = 0, /* Semaphore is not available */
    CSL_SEM_FREE = 1 /* Semaphore is available */
} CSL_SemFlag;

typedef enum {
    CSL_SEM_REARM_SEMINT0 = 0,
    CSL_SEM_REARM_SEMINT1 = 1,
    CSL_SEM_REARM_SEMINT2 = 2,
    CSL_SEM_REARM_SEMINT_ALL = 0x10
} CSL_SemEOISet

typedef enum {
    CSL_SEM_QUERY_REVISION,
    CSL_SEM_QUERY_ERROR,
    CSL_SEM_QUERY_FLAGS,
    CSL_SEM_QUERY_STATUS,
    CSL_SEM_QUERY_DIRECT,
    CSL_SEM_QUERY_INDIRECT
} CSL_SemHwStatusQuery;
```
Example 1. Sample API Call Lists (continued)

typedef enum {
    CSL_SEM_CMD_EOI_WRITE,
    CSL_SEM_CMD_FLAG_SET,
    CSL_SEM_CMD_FREE_DIRECT,
    CSL_SEM_CMD_WRITE_POST_DIRECT,
    CSL_SEM_CMD_FREE_INDIRECT,
    CSL_SEM_CMD_WRITE_POST_INDIRECT,
    CSL_SEM_CMD_FREE_QUERY,
    CSL_SEM_CMD_WRITE_POST_QUERY,
    CSL_SEM_CMD_CLEAR_ERR,
    CSL_SEM_CMD_CLEAR_FLAGS
} CSL_SemHwControlCmd;
/*******************/
/* DATA STRUCTURES */
/********************/
typedef struct _CSL_SemFlagClear_Arg{
    CSL_BitMask32        mask;
    CSL_SemOwnerId       masterId;
} CSL_SemFlagSetClear_Arg;

typedef struct {
    CSL_SemRegsOvly     regs;
} CSL_SemBaseAddress;

typedef struct {
    CSL_BitMask16       flags;
} CSL_SemParam;

typedef struct {
    Uint16              contextInfo;
} CSL_SemContext;

typedef struct CSL_SemObj{
    CSL_InstNum         instNum;
    int                 semNum;
    CSL_SemRegsOvly     regs;
} CSL_SemObj;

typedef volatile CSL_SemObj *CSL_SemHandle;

typedef struct {
    int                 semNum;
    CSL_SemOwnerId      semOwner;
    CSL_SemFlag        semFree;
} CSL_SemVal;

typedef struct CSL_SemFaultStatus {
    int                 semNum;
    CSL_SemError        errorMask;
    Uint16              faultID;
} CSL_SemFaultStatus;
**4.2 Accessing Semaphore Resource When Free**

A CPU accesses a shared peripheral (semaphore resource) by reading the semaphore direct (DIRECTx) register when the resource is free to use. Checking the semaphore query (QUERYx) register is the way to verify whether a particular resource is free or still in use by other CPU. If the FREE bit of QUERYx (where x is the semaphore peripheral ID) register is 1, the requested CPU directly locks the peripheral by reading the DIRECTx register and the requested peripheral is immediately granted access. After finishing the access, the CPU must release the resource by writing 1 to the FREE bit of the DIRECTx register so that another CPU can access the same resource.

Example 2 shows a semaphore resource (NUM = 4) acquired using the direct-read mode.

**Example 2. Sample Code to Access Semaphore in Direct-Read Mode**

```c
CSL_SemVal query;
CSL_SemHandle mySemHandle;
CSL_SemObj mySemObj;
CSL_SemContext SemContext;
CSL_SemParam mySemParam;

mySemParam.flags = 4;
CSL_semInit(&SemContext);

mySemHandle = CSL_semOpen(&mySemObj, 0,&mySemParam, NULL);
if(mySemHandle != NULL) {
    CSL_semGetHwStatus(mySemHandle, CSL_SEM_QUERY_STATUS,&query);
}

if(query.semFree == CSL_SEM_FREE) {
    CSL_semGetHwStatus(mySemHandle, CSL_SEM_QUERY_DIRECT,&query);
} else {
    // Do peripheral access
}

CSL_semHwControl(mySemHandle, CSL_SEM_CMD_FREE_DIRECT,NULL);

/* END OF Main code */
```
4.3 Accessing Semaphore Resource When Not Free

In certain situations when a shared resource is in use by another master, the requesting master/CPU can still post a request indirectly so that when the resource becomes available the requested master/CPU gets the access. Writing to the semaphore indirect (INDIRECTx) register when the resource is not free is the way to post a request in the request queue for the particular semaphore peripheral. Checking the semaphore query (QUERYx) register is the way to verify whether a particular resource is free or still in use by other CPU. If the FREE bit of the QUERYx (where x is the semaphore peripheral ID) register is 0 (indicating resource is in use), then the requested CPU can post a request by writing to the INDIRECTx register. When the resource becomes free, the master is notified by the interrupt and the requested peripheral is granted access. After finishing the access the CPU must release the resource by writing 1 to the FREE bit of the INDIRECTx register so that another CPU can access the same resource.

Example 3 shows a semaphore resource (NUM = 22) acquired using indirect-write mode.

**Example 3. Sample Code to Access Semaphore in Indirect-Write Mode**

```c
CSL_SemVal   query;
CSL_SemHandle mySemHandle;
CSL_SemObj   mySemObj;
CSL_SemContext SemContext;
CSL_SemParam mySemParam;

mySemParam.flags = 22;
CSL_semInit(&SemContext);

mySemHandle = CSL_semOpen(&mySemObj, 0,&mySemParam, NULL);
if(mySemHandle != NULL) {
   CSL_semGetHwStatus(mySemHandle,CSL_SEM_QUERY_STATUS,&query);
}
if(query.semFree == CSL_SEM_FREE) {
   // Direct access
   CSL_semGetHwStatus(mySemHandle,CSL_SEM_QUERY_DIRECT,&query);
}
CSL_semGetHwStatus(mySemHandle,CSL_SEM_QUERY_STATUS,&query);
if (query.semFree != CSL_SEM_FREE) {
   // Indirect access
   CSL_semHwControl(mySemHandle, CSL_SEM_CMD_WRITE_POST_INDIRECT,NULL);
   // Release the previously locked semaphore
   CSL_semHwControl(mySemHandle, CSL_SEM_CMD_FREE_DIRECT,NULL);
}
// Get the semaphore status.Still lock. This time for indirect pending access
CSL_semGetHwStatus(mySemHandle,CSL_SEM_QUERY_STATUS,&query);
if (query.semFree != CSL_SEM_FREE) {
   // do semaphore access
} else {
   // Return error
}
CSL_semHwControl(mySemHandle, CSL_SEM_CMD_FREE_INDIRECT,NULL);
/* END Of Main code */
```
4.4 Accessing Semaphore Resource in Combined Mode

A semaphore resource can be accessed in combination of direct-read mode and indirect-write mode. When the resource is free, access is immediately granted and when the resource is not free, it posts a request in the request queue.

Example 4 shows a semaphore resource (NUM = 0) acquired using combined-access mode.

Example 4. Sample Code to Access Semaphore in Combined-Access Mode

```c
CSL_SemVal query;
CSL_SemHandle mySemHandle;
CSL_SemObj mySemObj;
CSL_SemContext SemContext;
CSL_SemParam mySemParam;

mySemParam.flags = 0;
CSL_semInit(&SemContext);

mySemHandle = CSL_semOpen(&mySemObj, 0, &mySemParam, NULL);
if (mySemHandle != NULL) {
    CSL_semGetHwStatus(mySemHandle, CSL_SEM_QUERY_STATUS, &query);
}
if (query.semFree == CSL_SEM_FREE) {
    CSL_semGetHwStatus(mySemHandle, CSL_SEM_QUERY_INDIRECT, &query);
}
CSL_semGetHwStatus(mySemHandle, CSL_SEM_QUERY_STATUS, &query);
if (query.semFree != CSL_SEM_FREE) {
    // Do peripheral access
} else {
    // return error.
}

CSL_semHwControl(mySemHandle, CSL_SEM_CMD_FREE_QUERY, NULL);
/* END OF Main code */
```
5 Semaphore Registers

This section provides the semaphore memory map and descriptions of the peripheral registers.

5.1 Register Memory Map

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Description</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>PID</td>
<td>Peripheral Revision ID Register</td>
<td>Section 5.2.1</td>
</tr>
<tr>
<td>0x00C</td>
<td>EOI</td>
<td>EOI Register</td>
<td>Section 5.2.2</td>
</tr>
<tr>
<td>0x100 - 0x17C</td>
<td>DIRECT0-31</td>
<td>Direct Registers 0-31</td>
<td>Section 5.2.3</td>
</tr>
<tr>
<td>0x200 - 0x27C</td>
<td>INDIRECT0-31</td>
<td>Indirect Registers 0-31</td>
<td>Section 5.2.4</td>
</tr>
<tr>
<td>0x300 - 0x37C</td>
<td>QUERY0-31</td>
<td>Query Registers 0-31</td>
<td>Section 5.2.5</td>
</tr>
<tr>
<td>0x400</td>
<td>FLAG0</td>
<td>Flag0 Register (for C64x+ Core0)</td>
<td>Section 5.2.6</td>
</tr>
<tr>
<td>0x404</td>
<td>FLAG1</td>
<td>Flag1 Register (for C64x+ Core1)</td>
<td>Section 5.2.6</td>
</tr>
<tr>
<td>0x408</td>
<td>FLAG2</td>
<td>Flag2 Register (for C64x+ Core2)</td>
<td>Section 5.2.6</td>
</tr>
<tr>
<td>0x480</td>
<td>FLAG_SET0</td>
<td>Flag Set0 Register (for C64x+ Core0)</td>
<td>Section 5.2.7</td>
</tr>
<tr>
<td>0x484</td>
<td>FLAG_SET1</td>
<td>Flag Set1 Register (for C64x+ Core1)</td>
<td>Section 5.2.7</td>
</tr>
<tr>
<td>0x488</td>
<td>FLAG_SET2</td>
<td>Flag Set2 Register (for C64x+ Core2)</td>
<td>Section 5.2.7</td>
</tr>
<tr>
<td>0x500</td>
<td>ERR</td>
<td>Error Register</td>
<td>Section 5.2.8</td>
</tr>
<tr>
<td>0x504</td>
<td>ERROR_CLR</td>
<td>Error Clear Register</td>
<td>Section 5.2.9</td>
</tr>
</tbody>
</table>
Semaphore Registers

5.2 Register Descriptions

The following are some sample semaphore peripheral registers.

5.2.1 Peripheral ID Register (PID)

The semaphore peripheral ID (PID) register is shown in Figure 2 and described in Table 2.

![Figure 2. Peripheral ID Register (PID)]

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-30</td>
<td>SCHEME</td>
<td>01b</td>
<td>Used to distinguish which ID scheme is used.</td>
</tr>
<tr>
<td>29-28</td>
<td>RSVD</td>
<td>00</td>
<td>Reserved. Read returns 0. Write has no effect.</td>
</tr>
<tr>
<td>27-16</td>
<td>FUNC</td>
<td>0x802</td>
<td>Specifies module family</td>
</tr>
<tr>
<td>15-11</td>
<td>RTL</td>
<td>0000b</td>
<td>RTL Version</td>
</tr>
<tr>
<td>10-8</td>
<td>MAJOR</td>
<td>01b</td>
<td>Major Revision</td>
</tr>
<tr>
<td>7-6</td>
<td>CUSTOM</td>
<td>00b</td>
<td>Special/Custom Revision</td>
</tr>
<tr>
<td>5-0</td>
<td>MINOR</td>
<td>00000b</td>
<td>Minor Revision</td>
</tr>
</tbody>
</table>

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 2. Peripheral ID Register (PID) Field Descriptions

5.2.2 End-of-Interrupt Register (EOI)

The semaphore end-of-interrupt (EOI) register is used for re-arming the error/interrupt line after serving the existing error/interrupt. The EOI register is shown in Figure 3 and described in Table 3.

![Figure 3. End-of-Interrupt Register (EOI)]

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>INTERRUPT/ERROR SELECT</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>W</td>
<td>W</td>
<td></td>
</tr>
</tbody>
</table>

LEGEND: W = Write only; -n = value after reset

Table 3. End-of-Interrupt Register (EOI) Field Descriptions

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-8</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7-0</td>
<td>INTERRUPT/ERROR SELECT</td>
<td>0x00</td>
<td>Re-enable semint0 (For C64x+ Core0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x01</td>
<td>Re-enable semint1 (For C64x+ Core1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x02</td>
<td>Re-enable semint2 (For C64x+ Core2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x10</td>
<td>Re-enable All Error interrupt (For all C64x+ Cores)</td>
</tr>
</tbody>
</table>

(1) Reading of the EOI register will result in a memory-read exception being generated.
5.2.3 Direct Register (DIRECTx)

The semaphore direct (DIRECTx) register acquires the semaphore resource in direct-read mode as well as indirect-write mode. If the resource is free, the FREE field of the DIRECTx (x is the semaphore peripheral ID being requested) signifies whether the resource is granted or not. The DIRECTx register is shown in Figure 4 and described in Table 4.

Figure 4. Direct Register (DIRECTx)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-16</td>
<td>Reserved</td>
<td>0x0000</td>
<td>Reserved. Read returns 0. Write has no effect.</td>
</tr>
<tr>
<td>15-8</td>
<td>OWNER</td>
<td>0x00</td>
<td>When FREE = 0: Semaphore resource owner ID When FREE = 1: The value returns 0x00</td>
</tr>
<tr>
<td>7-1</td>
<td>Reserved</td>
<td>0x00</td>
<td>Reserved. Read returns 0. Write has no effect.</td>
</tr>
<tr>
<td>0</td>
<td>FREE</td>
<td>0</td>
<td>Read Operation: 0 Semaphore is not granted. 1 Semaphore is granted to the Master. FREE is cleared by the hardware at the end of the access. Write Operation: 0 Request is posted in the queue (indirect mode). 1 Semaphore is freed.</td>
</tr>
</tbody>
</table>
### Semaphore Registers

#### 5.2.4 Indirect Register (INDIRECTx)

The semaphore indirect (INDIRECTx) register acquires the semaphore resource in indirect-write mode. If the resource is free, the FREE bit of the INDIRECTx (x is the semaphore peripheral ID being requested) signifies whether the resource is granted or not. In addition, an interrupt is sent to the requested master. The INDIRECT register is shown in Figure 5 and described in Table 5.

![Figure 5. Indirect Register (INDIRECTx)](image)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-16</td>
<td>Reserved</td>
<td>0x0000</td>
<td>Reserved. Read returns 0. Write has no effect.</td>
</tr>
<tr>
<td>15-8</td>
<td>OWNER</td>
<td>0x00</td>
<td>When FREE = 0: Semaphore resource owner ID. When FREE = 1: The value returns 0x00.</td>
</tr>
<tr>
<td>7-1</td>
<td>Reserved</td>
<td>0x00</td>
<td>Reserved. Read returns 0. Write has no effect.</td>
</tr>
<tr>
<td>0</td>
<td>FREE</td>
<td>0</td>
<td>Read Operation: Semaphore is not granted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Semaphore is granted to the Master. FREE is cleared by the hardware at the end of the access. Write Operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Request is posted in the queue (indirect mode).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Semaphore is freed.</td>
</tr>
</tbody>
</table>
5.2.5 Query Register (QUERYx)

Each semaphore query register (QUERYx) checks the current semaphore resource status and also can be used for indirect-write mode to access the particular resource. Reading the QUERYx (x is the semaphore peripheral ID being accessed) register does not affect the status of the particular peripheral. The QUERY register is shown in Figure 6 and described in Table 6.

Figure 6. Query Register (QUERYx)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-16</td>
<td>Reserved</td>
<td>0x0000</td>
<td>Reserved. Read returns 0. Write has no effect.</td>
</tr>
<tr>
<td>15-8</td>
<td>OWNER</td>
<td>0x00</td>
<td>When FREE = 0: Semaphore resource owner ID When FREE = 1: The value returns 0x00</td>
</tr>
<tr>
<td>7-1</td>
<td>Reserved</td>
<td>0x00</td>
<td>Reserved. Read returns 0. Write has no effect.</td>
</tr>
<tr>
<td>0</td>
<td>FREE</td>
<td></td>
<td>Read Operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Semaphore is not granted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Semaphore is not granted to the Master</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write Operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Request is posted in the queue (indirect mode)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Semaphore is freed</td>
</tr>
</tbody>
</table>

Table 6. Query Register (QUERYx) Field Descriptions

LEGEND: R/W = Read/Write; R = Read only; \(-n\) = value after reset
Semaphore Registers

5.2.6 Flag Register (FLAGx)

Each semaphore flag (FLAGx, where x is the Master ID) register checks whether the particular master/core is currently holding the semaphore resource or not. There is one register for each core and the particular bit field signifies the holding status of the particular semaphore peripheral by the requested master. Once access to the particular semaphore resource is complete, the corresponding flag bit is cleared by writing a 1 to the particular bit of the FLAG register. The FLAGx register is shown in Figure 7 and described in Table 7.

Figure 7. Flag Register (FLAGx)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>FY</td>
<td>0x0</td>
<td>Semaphore Flag Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x1</td>
<td>Semaphore resource y is not owned by the master x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x1</td>
<td>Semaphore resource y is owned by the master x</td>
</tr>
</tbody>
</table>

5.2.7 Flag Set Register (FLAG_SETx)

Each semaphore flag set (FLAG_SETx, where x is the Master ID) register sets the flag bit of the particular semaphore peripheral. This register is implemented to check, by software programming, whether any resource can be accessed by the any of the masters. There is one register for each core and writing to the particular bit field sets the semaphore flag registers (FLAGx) corresponding bit-field value. The FLAG_SETx register is shown in Figure 8 and described in Table 8.

Figure 8. Flag Set Register (FLAG_SETx)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>FY</td>
<td>0x0</td>
<td>Semaphore Flag Set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x1</td>
<td>Semaphore flag y is cleared</td>
</tr>
</tbody>
</table>
5.2.8 Error Register (ERR)

The semaphore error (ERR) register updates any kind of error that occurs while acquiring a semaphore resource by any core. By reading the ERR register the particular master/core should clear the particular error and program the EOI register so that a re-arm occurs for the next semaphore access by the same master. The ERR register is shown in Figure 9 and described in Table 9.

**Figure 9. Error Register (ERR)**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-16</td>
<td>Reserved</td>
<td>0x0000</td>
<td>Reserved. Read returns 0. Write has no effect.</td>
</tr>
<tr>
<td>15-8</td>
<td>FAULTID</td>
<td>0x00</td>
<td>Master ID number that caused the error</td>
</tr>
<tr>
<td>7-3</td>
<td>SEM_NUM</td>
<td>0x00</td>
<td>Semaphore peripheral ID number (0 to 31)</td>
</tr>
<tr>
<td>2-0</td>
<td>ERR</td>
<td>000</td>
<td>Semaphore error code. 000 No semaphore access error has occurred. 001 Master ID FAULTID attempted to free semaphore NUM when it was already free. 010 Master ID FAULTID attempted to free semaphore NUM while not currently owned by FAULTID. 011 Master ID FAULTID attempted to acquire semaphore NUM while it was already owned by FAULTID. 100 Master ID FAULTID attempted to acquire semaphore NUM while FAULTID already had a request pending.</td>
</tr>
</tbody>
</table>

**Table 9. Error Register (ERR) Field Descriptions**

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset
Semaphore Registers

5.2.9 Error Clear Register (ERROR_CLR)

The semaphore error clear (ERROR_CLR) register clears the existing error code. The master should reprogram the EOI register after clearing the error so that a re-arm occurs for the next error event. The ERROR_CLR register is shown in Figure 10 and described in Table 10.

Figure 10. Error Clear Register (ERROR_CLR)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-16</td>
<td>Reserved</td>
<td>0x0000</td>
<td>Reserved. Read returns 0. Write has no effect.</td>
</tr>
<tr>
<td>15-8</td>
<td>FAULTID</td>
<td>0x00</td>
<td>Master ID number that caused the error</td>
</tr>
<tr>
<td>7-3</td>
<td>SEM_NUM</td>
<td>0x00</td>
<td>Semaphore peripheral ID number (0 to 31)</td>
</tr>
<tr>
<td>2-0</td>
<td>ERR</td>
<td></td>
<td>Semaphore error code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000</td>
<td>No semaphore access error has occurred.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>001</td>
<td>Master ID FAULTID attempted to free semaphore NUM when it was already free.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>010</td>
<td>Master ID FAULTID attempted to free semaphore NUM while not currently owned by FAULTID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>011</td>
<td>Master ID FAULTID attempted to acquire semaphore NUM while it was already owned by FAULTID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>Master ID FAULTID attempted to acquire semaphore NUM while FAULTID already had a request pending.</td>
</tr>
</tbody>
</table>
### 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 test 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:

<table>
<thead>
<tr>
<th>Products</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amplifiers</td>
<td>Audio</td>
</tr>
<tr>
<td>Data Converters</td>
<td>Automotive</td>
</tr>
<tr>
<td>DSP</td>
<td>Broadband</td>
</tr>
<tr>
<td>Clocks and Timers</td>
<td>Digital Control</td>
</tr>
<tr>
<td>Interface</td>
<td>Medical</td>
</tr>
<tr>
<td>Logic</td>
<td>Military</td>
</tr>
<tr>
<td>Power Mgmt</td>
<td>Optical Networking</td>
</tr>
<tr>
<td>Microcontrollers</td>
<td>Security</td>
</tr>
<tr>
<td>RFID</td>
<td>Telephony</td>
</tr>
<tr>
<td>RF/IF and ZigBee® Solutions</td>
<td>Video &amp; Imaging</td>
</tr>
<tr>
<td></td>
<td>Wireless</td>
</tr>
</tbody>
</table>

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2008, Texas Instruments Incorporated